# Domain Adaptation and Domain Generalization with Representation Learning

by

Muhammad Ghifary

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Doctor of Philosophy
in Computer Science.

Victoria University of Wellington
2016

## Abstract

Machine learning has achieved great successes in the area of computer vision, especially in object recognition or classification. One of the core factors of the successes is the availability of massive *labeled* image or video data for training, collected manually by human. Labeling source training data, however, can be expensive and time consuming. Furthermore, a large amount of labeled source data may not always guarantee traditional machine learning techniques to generalize well; there is a potential *bias* or *mismatch* in the data, *i.e.*, the training data do not represent the target environment.

To mitigate the above dataset bias/mismatch, one can consider *domain adaptation*: utilizing labeled training data and *unlabeled* target data to develop a well-performing classifier on the target environment. In some cases, however, the unlabeled target data are nonexistent, but multiple labeled sources of data exist. Such situations can be addressed by *domain generalization*: using multiple source training sets to produce a classifier that generalizes on the unseen target domain. Although several domain adaptation and generalization approaches have been proposed, the domain mismatch in object recognition remains a challenging, open problem – the model performance has yet reached to a satisfactory level in real world applications.

The overall goal of this thesis is to progress towards solving dataset bias in visual object recognition through *representation learning* in the context of domain adaptation and domain generalization. Representation learning is concerned with finding proper data representations or features via learning rather than via engineering by human experts. This thesis proposes several representation learning solutions based on deep learning and kernel methods.

This thesis introduces a *robust-to-noise* deep neural network for handwritten digit classification trained on "clean" images only, which we name

*Deep Hybrid Network* (DHN). DHNs are based on a particular combination of sparse autoencoders and restricted Boltzmann machines. The results show that DHN performs better than the standard deep neural network in recognizing digits with Gaussian and impulse noise, block and border occlusions.

This thesis proposes the *Domain Adaptive Neural Network* (DaNN), a neural network based domain adaptation algorithm that minimizes the classification error and the domain discrepancy between the source and target data representations. The experiments show the competitiveness of DaNN against several state-of-the-art methods on a benchmark object dataset.

This thesis develops the *Multi-task Autoencoder* (MTAE), a domain generalization algorithm based on autoencoders trained via multi-task learning. MTAE learns to transform the original image into its analogs in multiple related domains simultaneously. The results show that the MTAE's representations provide better classification performance than some alternative autoencoder-based models as well as the current state-of-the-art domain generalization algorithms.

This thesis proposes a fast kernel-based representation learning algorithm for both domain adaptation and domain generalization, *Scatter Component Analysis* (SCA). SCA finds a data representation that trades between maximizing the separability of classes, minimizing the mismatch between domains, and maximizing the separability of the whole data points. The results show that SCA performs much faster than some competitive algorithms, while providing state-of-the-art accuracy in both domain adaptation and domain generalization.

Finally, this thesis presents the *Deep Reconstruction-Classification Network* (DRCN), a deep convolutional network for domain adaptation. DRCN learns to classify labeled source data and also to reconstruct unlabeled target data via a shared encoding representation. The results show that DRCN provides competitive or better performance than the prior state-of-the-art model on several cross-domain object datasets.

# Acknowledgments

It has been such a pleasure working towards my PhD thesis and being part of the School of ECS, VUW. I cannot resist expressing my gratitude to individuals without whom this thesis would not be able to finish.

First I would like to acknowledge my PhD advisers, Prof. Bastiaan Kleijn and Prof. Mengjie Zhang, who provided me the best academic supervision I have ever had. Bastiaan is a discoverer-minded, genuine intellectual who has changed my view from being "too pragmatic" to being "more thoughtful" on research. He has brought me that deep understanding, developing intuitions and insights when tinkering a research problem should come first and be the highest priority before anything else. Meng is a perfect example of determination, diligence, tenacity, and leadership; I have never met anyone else better than him in these aspects. He uses his infinite enthusiasm to motivate me and his other students to work at their best, and it is quite infectious! All in all, they are top-notch advisers; both are very generous and expect nothing but best for me. I feel grateful and privileged to work with them.

I would also like to thank Dr. David Balduzzi, who has been my most frequent collaborator during my PhD (well, David is more like my unofficial adviser). I learned a lot from him and admire his brilliance, intuition, curiosity, and enthusiasm. Thank you David! I hope we can always keep collaborating in the future.

Furthermore, I would like to acknowledge all my PhD examiners: Dr. Marcus Frean, Prof. Stephen Marsland, and Dr. Cheng Soon Ong for providing me comprehensive feedback and valuable suggestions.

I am very grateful for the Victoria Doctoral Scholarship, the Faculty

Strategic Research Grant (211215), the School of ECS Grant (E1745), and the Marsden Grant of New Zealand (E2401) for the financial support. My PhD project would not be possible without those supports.

During my PhD, I have been attached with three different research/study groups: Communication and Signal Processing (CaSP), Evolutionary Computation Research Group (ECRG), and Marcus's Festival of Doubt (FoD). I would like to thank people in those groups for creating a stimulating research environment. Special thanks also to my PhD mates: Wenyu, Harsh, Ramoni, Fatih, Callum, Tim, Lakhsmi, Aryan, and other colleagues I cannot mention here, for the friendship, collaboration, discussions, and jokes.

I wish to thank all my Indonesian folks living in Wellington from (but not limited to) Indonesian Student Association (PPIW), the Indonesian embassy in New Zealand (KBRI), KAMASI, and UMI. Gathering with those folks and communities makes me feel never far away from my home country.

I wholeheartedly thank all my former school teachers and lecturers in Indonesia without whom my "PhD trajectory" would not exist. A special thank to my former undergraduate/graduate adviser from Institut Teknologi Bandung (ITB), Dr. Dwi H. Widyantoro, with whom I learned about machine learning and began my research journey.

Last but foremost, I would like to thank my family, especially my dearest wife Syva (she is my soulmate, best friend, and means the world to me) and my lovely 3-year-old daughter Naisha. They sacrificed tremendously as a consequence of the time I spent too much in doing my research. I genuinely apologize and am looking forward to spending more time with them after PhD! Finally, I wish to express my greatest gratitude to the best and wisest men, my parents (dr. H. Thala'at Z.A. and Hj. Cholilah). Their love, courage, and wisdom have shaped me into what I am today. I would like to also thank my parents-in-law (Ir. H. Goenarso and Hj. Winny Widalia) for their support.

# List of Publications

1. M. Ghifary, W. Bastiaan Kleijn, M. Zhang, "Sparse representations in deep learning for noise-robust digit classification", *in Proceedings of the 28th International Conference on Image and Vision Computing New Zealand (IVCNZ)*, Wellington, New Zealand, 2013. pp. 340–345.

2. M. Ghifary, W. Bastiaan Kleijn, M. Zhang, "Deep Hybrid Network with good *out-of-sample* recognition", *in Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Florence, Italy, 2014. pp. 5437–5441.

3. M. Ghifary, W. Bastiaan Kleijn, M. Zhang, "Domain Adaptive Neural Networks for Object Recognition" *in Proceedings of Pacific Rim International Conference on Artificial Intelligence (PRICAI)*, Gold Coast, Australia, 2014. pp. 898–904.

4. M. Ghifary, W. Bastiaan Kleijn, M. Zhang, "Domain Generalization for Object Recognition with Multi-task Autoencoders", *in Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2551–2559.

5. M. Ghifary, D. Balduzzi, W. Bastiaan Kleijn, M. Zhang, "Scatter Component Analysis: A Unified Framework for Domain Adaptation and Domain Generalization", *submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence on 22 October 2015.*

6. D. Balduzzi, M. Ghifary, "Compatible Value Gradients for Reinforcement Learning of Continuous Deep Policies", *submitted to Machine Learning Journal.*

7. D. Balduzzi, M. Ghifary, "Strongly-typed Recurrent Neural Networks", *in Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016. pp. 1292–1300.

8. M. Ghifary, W. Bastiaan Kleijn, M. Zhang, D. Balduzzi, "Deep Reconstruction-Classification Networks for Unsupervised Domain Adaptation", *submitted to European Conference on Computer Vision (ECCV) 2016.*

# Contents

# List of Tables

# List of Figures

# 1 Introduction

*This chapter introduces the thesis. It consists of the problem statement, the motivations, the research goals and objectives, the major contributions and the organization of the thesis.*

## 1.1 Problem Statement

Supervised learning is perhaps the most popular task in machine learning and has recently achieved dramatic successes in many applications such as object recognition [118, 198], object detection [77], speech recognition [50], and machine translation [207]. These successes derive in large part from the availability of massive *labeled* datasets such as PASCAL VOC2007 [62] and ImageNet [117]. Unfortunately, manually obtaining labels is often a time consuming and costly task that requires human experts. A shortage of labels may prevent the standard supervised learning algorithms from reaching their true potential. Furthermore, collecting more labeled samples does not necessarily lead to a better generalization ability [215]. In object recognition, for example, training images may be collected under specific conditions involving camera viewpoints, backgrounds, human selection preference, and object transformations that may be different from those of the target environment. The fore-mentioned issues result in a problem that is commonly referred to as *domain shift* or *dataset bias* [171, 215], *i.e.*, a learning algorithm trained on a particular dataset/domain generalizes poorly across datasets.

This thesis is about **domain adaptation** [26] and **domain generalization** [25]; both are concerned with reducing dataset bias. In this context, a *domain* represents a probability distribution from which the samples are drawn and is often equated with a dataset. The domain is usually divided into two different types: the *source domain* and the *target domain*, to distinguish between a domain with labeled samples and a domain without labeled samples. These two domains are related but different, which limits the applicability of standard supervised learning models on the target domain. Specifically, the basic assumption in standard supervised learning that training and test data come from the same distribution is violated. The goal of both domain adaptation and domain generalization is to produce good models on a target domain by training on labeled samples from the source domain(s). The difference between domain adaptation and domain generalization is *the availability of the unlabeled samples from the target domain*. In domain generalization no samples of any kind are available for the target domain.

It is widely believed that the success of supervised learning algorithms strongly depends on data representation. A good representation should be discriminative, *i.e.*, one that is useful as an input to a supervised predictor. In the context of classification, good representations can be interpreted as ones that are linearly separable – if the representations form two sets of points in a high dimensional space, there exists a hyperplane that separates those sets. A traditional procedure to obtain such a representation is by feature engineering, *i.e.*, manually handcrafting features by means of human expertise in the domain of interest. Examples of handcrafted features in computer vision applications are such as LBP [161], SIFT [136], and HoG [51]. It would be highly desirable in practice if a discriminative representation can be *learned from data as an alternative to manual feature engineering*. The latter approach is referred to as **representation learning** [19].

This thesis focuses on applying representation learning to develop good domain adaptation and domain generalization models for visual object recognition problems. Although several representation learning approaches have been proposed, dataset bias remains essentially unsolved since model recognition accuracy has yet to reach a level that is satisfactory for real-world applications. Therefore, a learning algorithm that can produce a discriminative representation and also can highly reduce dataset bias is required. This thesis studies the effectiveness of neural networks and kernel feature extraction methods, which are considered as non-linear representation learning approaches, to deal with dataset bias. They have been successfully applied to a wide range of applications, but their potential for solving the dataset bias problem has not been fully investigated.

## 1.2 Motivations

This section first elaborates the importance of the main problem addressed in this thesis: *dataset bias* in visual object recognition. It is then followed by the rationale of using neural networks and kernel methods as the approaches to solving the problem.

### 1.2.1 Challenges of Dataset Bias

Domain shift or dataset bias in object recognition is a challenging problem to solve. The visual world is so complex that any finite set of images ends up describing only specific aspects of it. Thus, any standard supervised learning that learns from these labeled images is not likely to generalize well on another domain. An ideal learning algorithm should be able to capture a "general concept" of the visual world from finite samples so that the bias towards a domain could be avoided. Human is remarkably capable of learning a concept after experience with only a small number of samples [65].

To progress towards a methodology to extract the "general concept", one may reduce the problem into overcoming domain shift or dataset bias between domains. Suppose that a learning algorithm is initially provided with a set of finite labeled samples, which we refer to as a source domain. The learned model is expected to perform well on a target domain, which is different from but related to the source domain. Using only the (labeled) source samples as the training set might not be sufficient to provide good generalization on the target domain. It is thus natural to seek information other than the source domain as auxiliary knowledge that might bridge the gap between the source and target domain.

The auxiliary information can possibly come from the target domain itself. This is where domain adaptation plays a role. *Domain adaptation algorithms aim to mitigate dataset bias by learning from labeled source samples and leveraging some knowledge from the target domain.* We can utilize either of at least two types of information: i) *human knowledge* or ii) *unlabeled samples*. An example of using human knowledge is as follows. Consider a problem of handwritten digit recognition where the source images is in the form of "clean" digits with a particular style. We desire a model learned from only the "clean" source images that can recognize "noisy" digits. In this case, the noise acts as the human knowledge, which guides us in designing an algorithm to cancel out the damage caused by the noise.

Another form of the auxiliary information is the unlabeled samples from the target domain, which is relatively easy to obtain. This approach is the standard way performed by existing domain adaptation algorithms. The challenge is how to properly utilize the unlabeled target samples during training such that the generalization onto the target domain can be maximized. An improper use of unlabeled target samples may even reduce the generalization ability.

If the target domain is completely unknown or the unlabeled target samples even cannot be accessed, domain adaptation is not suitable. One may seek other related domains as additional sources with the hope that the algorithm would still generalize well on the target domain. Such a learning setting is referred to as *domain generalization: learning a good model on the target domain from (labeled) source domains.* The task is how to extract commonalities among those source domains used to generalize a model to the target domain. An ad-hoc strategy of using the additional source domains, that is, treating them as just extra training samples for supervised learning may not maximize or even hurts the generalization.

### 1.2.2 Why Neural Networks and Kernel Methods

Neural networks and kernel methods have long been known as being among the most powerful families of machine learning algorithms. Both are equipped with a *nonlinear* transform to provide useful data representations. In a classification task, for example, training an algorithm from real-world data such as images and video is often problematic. Such data is usually complex, redundant, and highly variable. Neural networks and kernel methods are more capable of extracting discriminative representations of complex data than other (linear) representation learning algorithms.

Both neural networks and kernel methods have appealing properties on their own. Neural networks are able to learn deep hierarchical layers, which can lead to more abstract representations. These representations are generally *invariant* to most local changes of the input. In addition, learning a deep network architecture, which is currently known as *deep learning*, is highly scalable since it runs in linear time, can handle streaming data, and can be parallelized on GPUs. Such abstract representations and scalability has led deep neural networks to achieve some breakthroughs in visual object recognition [42, 93, 97, 118].

On the other hand, kernel feature extraction aims to produce useful representations in a slightly different way. It maps inputs into high dimensional (possibly infinite) representations that facilitates *linear separation*. In classification, linear separation can be interpreted as follows. Suppose that there are two sets of high dimensional points. The points should establish a configuration such that those two sets can be easily separated by a single hyperplane; each set represents a class of points. Several members of kernel feature extraction are kernel principal component analysis [192], kernel discriminant analysis [146], and kernel independent component analysis [9].[1] The most appealing properties of kernel feature extraction relate to computation. Firstly, kernel feature extraction does not explicitly compute the high dimensional representations, but it uses the so-called *kernel trick* by simply computing the inner products between the images of all pairs of data in the representation space. Secondly, kernel algorithms are theoretically well founded and mostly based on convex optimization or eigenproblems that admit an exact solution.

This thesis extends the capability of neural networks and kernel feature extraction in the context of domain adaptation and/or domain generalization. Although several domain adaptation/generalization algorithms based on those methods exists, more powerful algorithms to reduce dataset bias are still highly required in practice.

## 1.3 Research Goals

The overall goal of this thesis is to progress towards solving domain shift / dataset bias in visual object recognition through representation learning in the context of domain adaptation and domain generalization. This thesis proposes several representation learning algorithms based on deep learning and kernel methods. To achieve the overall goal, we establish the following research objectives.

1. Develop a new deep learning-based algorithm to deal with a cross-domain handwritten digit recognition problem, where the source do-

[1]In general, any algorithms based on the inner products can be kernerlized.

main contains "clean" digit images and the target domain contains "noisy" digit images. The algorithm is expected to be robust to the following types of noise appearing in the target domain: border, block, Gaussian, impulse noise, and background noise, and provide better accuracy than the standard deep neural networks, particularly on small training sets.

Traditionally, a common approach to improve generalization over noisy observations is by adding some noisy examples in the training set. This study investigates a strategy from a different perspective, *i.e.*, designing a robust-to-noise algorithm that learns from clean examples only. Standard learning algorithms, including deep neural networks, will suffer from dataset bias under such settings since the training and test examples are always assumed to be drawn from a same domain. A domain adaptation strategy to tackle the *clean-noisy* problem above, which is a particular form of dataset bias, is thus required.

2. Develop a new regularization technique for the supervised neural network training to establish a good domain adaptation model. The algorithm is expected to reduce the distribution difference between the source and target data representations, and to perform well on cross-domain object recognition tasks.

   A standard approach to developing neural network-based domain adaptation methods is by unsupervised pretraining on all unlabeled source and target data and then supervised fine-tuning on labeled source data [38,41,79]. While the unsupervised pretraining is effective in some cases, it is unclear whether it indeed reduces the distribution mismatch between the source and target domains during training. An explicit set up to reduce the distribution mismatch by means of a particular criterion is worthwhile. We seek a single-step supervised label training in which the unlabeled target samples can be also considered as a regularization. The regularization criterion associated with the distribution difference minimization needs to be explicitly specified in the learning objective.

3. Develop a novel multi-task representation learning algorithm that provides good *shared representations* for domain generalization. The algorithm is expected to learn multiple *image reconstruction* tasks over many domains and to approximate the underlying "transformation" among the domains. This strategy results in discriminative representations useful for an unseen target domain.

   In many circumstances, there exists labeled samples coming from many (source) domains that can be used to train a classifier. However, the classifier may operate on another different but related domain, which is unseen during training. A possible approach to overcoming dataset bias induced by such a setting is by learning the commonality among the source domains with the hope that it can help generalize the model on the target domain. Multi-task learning (MTL) [35] is a well suited approach to doing so. While MTL usually performs the discriminative tasks, *i.e.*, using labels as the output signals, our proposed algorithm looks into a different perspective, that is, performing multiple image reconstruction tasks, where the output signals are the unlabeled images.

4. Develop a fast, simple, and theoretically motivated representation learning algorithm based on the *kernel trick* for both domain adaptation and domain generalization. The proposed algorithm is supposed to run much faster than the prior state-of-the-art algorithms on several benchmark image datasets and to provide competitive performance in accuracy.

   Typical state-of-the-art domain adaptation and domain generalization algorithms for object recognition result in optimization problems that are inefficient to solve [133, 134, 195, 236]. In addition, while domain adaptation and domain generalization are closely related settings, the approaches are generally not compatible to each other – domain adaptation approaches cannot be directly applied to domain generalization or vice versa. A fast algorithm equipped with domain adaptation-generalization compatibility is highly preferable in practice.

5. Develop a new algorithm to train deep convolutional networks for domain adaptation that approximates the *true correspondence* between the source and target domains. The model is expected to provide the state-of-the-art performance on several cross-domain object recognition tasks and also be scalable to the growth of the training data.

   Deep convolutional networks are considered to be the best object recognition models to date [93]. They have also been applied to domain adaptation problems that provide significant accuracy improvements [56, 73, 132]. However, the performance of prior state-of-the-art models in the context of domain adaptation has not reached a level that is satisfactory for real-world applications. This objective will further improve the domain adaptation performance of deep convolutional neural networks by designing a new multi-task algorithm, where the tasks are the classification and the image reconstruction trained simultaneously. It will also investigate whether the proposed algorithm can indeed seek the between-domain correspondence by observing the reconstruction behavior.

## 1.4 Major Contributions

This section summarizes the major contributions of this thesis. Each contribution below is presented in detail in a chapter, from Chapters 3 to 7.

1. The thesis proposes Deep Hybrid Networks (DHN) that are robust to recognize “noisy" handwritten digits given only “clean" digits as the training images. This networks are based on a particular combination of an autoencoder with sparse regularization on the activation (SAE) and restricted Boltzmann machines (RBMs). SAE is used to extract

sparse features, which are expected to be *noise-invariant* in the observations. The stacked RBMs then observe the sparse features as inputs to learn the top hierarchical features. The use of RBMs is motivated by the fact that the stacked RBMs typically provide good generalization for deep architectures [97]. To improve the robustness against local block noise, a variant of DHN is proposed by imposing sparse connections in addition to the sparse activation in the auto-encoder layer. The experiments show that the proposed deep networks provide good performance in both the *in-domain* and cross-domain tasks, especially when trained from a small sample set.

Part of this contribution has been published in:

Muhammad Ghifary, W. Bastiaan Kleijn, Mengjie Zhang. "Sparse Representations in Deep Learning for Noise-Robust Digit Classification". Proceedings of International Conference on Image and Vision Computing New Zealand (IVCNZ). Wellington, New Zealand, November 27-29, 2013. pp. 340-345.

Muhammad Ghifary, W. Bastiaan Kleijn, Mengjie Zhang. "Deep Hybrid Networks with Good Out-of-sample Object Recognition". Proceedings of 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). Florence, Italy, May 4-0, 2014. pp. 5437-5441.

2. The thesis proposes a neural network that utilizes Maximum Mean Discrepancy (MMD) [28] as a regularization embedded in the supervised back-propagation training. We refer to this model as Domain Adaptive Neural networks (DaNN). To the best of our knowledge, this contribution is the first study of utilizing MMD in the context of neural networks. MMD empirically measures the distribution mismatch between the source and target domains in the latent space, which is minimized during neural network training. The experiments demonstrate that the MMD regularization is an effective technique to establish a neural net-

work as a good domain adaptation model on the Office dataset [187]. Furthermore, DaNN preceded by denoising autoencoder training [225] achieves better performance than recent benchmark models on the same dataset.

Part of this contribution has been published in:

Muhammad Ghifary, W. Bastiaan Kleijn, Mengjie Zhang, "Domain Adaptive Neural Networks for Object Recognition". Proceedings of 13th Pacific Rim International Conference on Artificial Intelligence (PRICAI). Gold Coast, Australia, December 1-5, 2014. pp. 898-904.

3. The thesis proposes a novel representation learning algorithm, *Multi-task Autoencoder* (MTAE), that provides good domain generalization performance for object recognition. MTAE extends the standard denoising autoencoder framework by substituting artificially induced corruption with naturally occurring inter-domain variability in the appearance of objects. Instead of reconstructing images from noisy versions, MTAE learns to transform the original image into its analogs in multiple related domains. It thereby learns features that are robust to variations across domains. The algorithm was evaluated on several benchmark image recognition datasets, where the task is to learn feature from multiple datasets and to then predict image label from unseen datasets. We found that (denoising) MTAE outperforms alternative autoencoder-based models as well as the current state-of-the-art algorithms for domain generalization.

   Part of this contribution has been published in:

   Muhammad Ghifary, W. Bastiaan Kleijn, Mengjie Zhang, David Balduzzi. "Domain Generalization for Object Recognition with Multi-task Autoencoders". Proceedings of IEEE International Conference on Computer Vision (ICCV). Santiago, Chile, December 11-18, 2015.

4. The thesis proposes a fast kernel-based representation learning algo-

rithm referred to as *Scatter Component Analysis* (SCA), which can be applied to both domain adaptation and domain generalization. SCA is based on a simple geometrical measure, *i.e.*, *scatter*, which operates on reproducing kernel Hilbert space. SCA finds a representation that trades between maximizing the separability of classes, minimizing the mismatch between domains, and maximizing the separability of data; each of which is quantified through scatter. The optimization problem of SCA can be reduced to a generalized eigenvalue problem, which results in a fast and exact solution. Comprehensive experiments on benchmark cross-domain object recognition datasets verify that SCA performs much faster than several state-of-the-art algorithms, while provides competitive classification accuracy in both domain adaptation and domain generalization. We also show that domain scatter, a simple consequence when taking two domains into scatter, can be used to establish a theoretical generalization bound in the case of domain adaptation.

Part of this contribution has been submitted to:

Muhammad Ghifary, David Balduzzi, W. Bastiaan Kleijn, Mengjie Zhang. "Scatter Component Analysis: A Unified Framework for Domain Adaptation and Domain Generalization". IEEE Transaction on Pattern Analysis and Machine Intelligence (TPAMI).

5. The thesis proposes Deep Reconstruction-Classification Network (DRCN), a novel deep convolutional network that provides state-of-the-art domain adaptation performance in object recognition. DRCN jointly learns two tasks through the standard back-propagation: i) supervised classification of labeled source data and ii) unsupervised reconstruction of unlabeled target data. This strategy aims to approximate the *true correspondence* between the source and target domains, which is encoded through a shared representation. Such a representation endows the model with a *label prediction pipeline* that generalizes onto

the target domain. The DRCN's performance is evaluated on a series of cross-domain object recognition tasks, where DRCN provides higher accuracy than prior state-of-the-art model in almost all cases. An interesting observation is that DRCN's reconstruction pipeline transforms images from the source domain into images whose appearance resembles those from the target domain. We also theoretically demonstrate that the learning objective of DRCN is approximately equivalent to solving a semi-supervised learning problem on the target domain.

Part of this contribution has been submitted to:

Muhammad Ghifary, W. Bastiaan Kleijn, Mengjie Zhang, David Balduzzi. "Deep Reconstruction-Classification Networks for Unsupervised Domain Adaptation". European Conference on Computer Vision (ECCV). 2016.

## 1.5 Organization of The Thesis

This remainder of the thesis is organized as follows. Chapter 2 presents the background and review of related work. Chapters 3-7 presents the main contributions of the thesis, each of which addresses one of the research objective. Figure 1.1 depicts the structure of the thesis contributions, which can be classified into two groups: domain adaptation and domain generalization. Chapter 8 summarizes the thesis.

Chapter 2 presents essential background and basic concepts of machine learning, learning theory, deep learning, and kernel-based feature learning. It then reviews existing work in domain adaptation and domain generalization that focuses on the application to visual object recognition. Finally, it discusses open questions and current challenges that form the motivations of this thesis.

Chapter 3 presents the first contribution of this thesis: Deep Hybrid Network (DHN), a deep learning model based on a particular combination of sparse autoencoders and stacked restricted Boltzmann machines for robust-

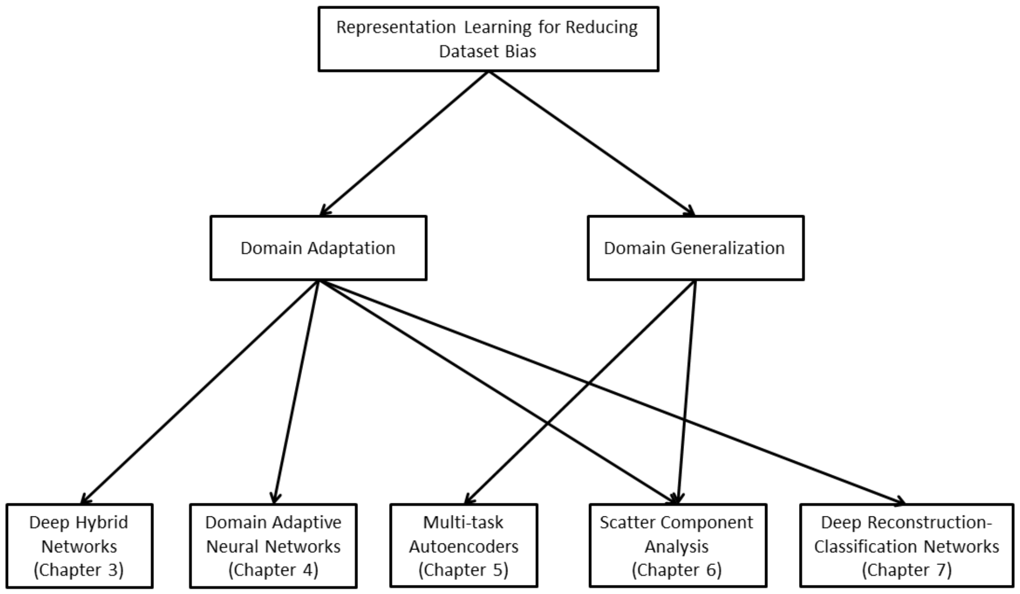

Representation Learning for Reducing Dataset Bias
Domain Adaptation
Domain Generalization
Deep Hybrid Networks (Chapter 3)
Domain Adaptive Neural Networks (Chapter 4)
Multi-task Autoencoders (Chapter 5)
Scatter Component Analysis (Chapter 6)
Deep Reconstruction-Classification Networks (Chapter 7)

Figure 1.1: Overall structure of the thesis contributions.

to-noise handwritten digit recognition. This model addresses a specific domain adaptation setting, where the source domain contains the “clean” digits and the target domain contains the “noisy” digits. The chapter evaluates DHN’s performance accuracy against traditional deep learning methods over various types of noise applied in the target domain, *i.e.*, border, block, Gaussian, impulse, and background noise. The results are then presented and analyzed.

Chapter 4 presents Domain Adaptive Neural Networks (DaNN), models trained using back-propagation with a regularization that aims to minimize the distribution difference in the hidden/latent space. Maximum Mean Discrepancy (MMD) is used as the distribution difference measure embedded in the DaNN’s learning objective. DaNN is examined and compared with standard multi-layer perceptrons, SVMs, and recent domain adaptation algorithms. The effect of using denoising autoencoder pretraining in DaNN is also analyzed. The evaluation results in accuracy on the Office dataset are then presented.

Chapter 5 presents Multi-task Autoencoders (MTAE), a novel multi-task representation learning algorithm for domain generalization. The tasks are in the form of *self-domain* or *between-domain* image reconstructions. The chapter compares MTAE with standard autoencoder-based models and also with recent domain generalization algorithms on several modern benchmark object datasets. The evaluation results in performance accuracy of MTAE are then presented. Analyses of MTAE's effectiveness by observing the weight visualization and also the local dimensionality of the manifold in the feature space are also established.

Chapter 6 presents Scatter Component Analysis (SCA), a fast kernel-based representation learning algorithm for both domain adaptation and domain generalization. It discusses the SCA's capability to seek a representation that satisfies four criteria: i) the source and target domains are similar, ii) elements with the same label are similar, iii) elements with different labels are separated, and iv) the variance of the whole data is maximized. The optimization problem of SCA can be reduced to a generalized eigenvalue problem that admits a fast and exact solution. The chapter evaluates the SCA's cross-domain object recognition performance in both domain adaptation and domain generalization settings in comparison to prior state-of-the-art algorithms. This chapter also establishes a theoretical generalization bound of SCA in the context of domain adaptation.

Chapter 7 presents the final contribution of this thesis: Deep Reconstruction-Classification Networks (DRCN), domain adaptation models based on convolutional neural networks equipped endowed with classification and reconstruction pipelines. The chapter discusses the DRCN algorithm: jointly learning the (supervised) classification on the source images and the (unsupervised) reconstruction on the target images to produce a domain adaptive shared representation. The cross-domain recognition performance of DRCN is evaluated on several large-scale object datasets in comparison to the standard and also state-of-the-art deep convolutional networks. A property of

DRCN in relation to learning the correspondence between the source and target domain is also investigated in this chapter.

Chapter 8 summaries the thesis and draws overall conclusions. Several possible future research directions, especially in domain adaptation and domain generalization, are also discussed.

## 1.6 Benchmark Datasets

The proposed representation learning algorithms in this thesis were evaluated on a number of benchmark visual object datasets. Table 1.1 lists the datasets with their *original* configuration ranging over handwritten digits, objects in the office environment, and general objects in nature.

Unlike the standard machine learning setting, the actual task performed on these datasets is the cross-dataset classification for evaluating the performance of domain adaptation or domain generalization algorithms. For example, an algorithm is trained on the (labeled) MNIST training set and is then applied to recognize the USPS test digits. Furthermore, the configuration of the datasets is often customized to fit the designated cross-domain recognition task. In details, the modified cross-dataset settings and configurations can be found in the experiment sections of the contribution chapters (Ch. 3 - 7).

Table 1.1: The summary of benchmark datasets. The '*' indicates the sub-datasets/domains contained in the Office dataset [187].

| Dataset | Type | Dimension | #Instances | #Classes | Chapter |
|---|---|---|---|---|---|
| MNIST [125] | Grayscale Digits | $28 \times 28$ | $70,000$ ($60,000$ for training and validation, $10,000$ for test) | 10 | 3, 5, 6, 7 |
| USPS [106] | Grayscale Digits | $16 \times 16$ | $9,298$ ($7,291$ for training, $2,007$ for test) | 10 | 6, 7 |
| SVHN [156] | RGB Digits | $32 \times 32$ | $99,289$ ($73,257$ for training, $26,032$ for test) | 10 | 7 |
| CIFAR-10 [117] | RGB Objects | $32 \times 32$ | $60,000$ ($50,000$ for training, $10,000$ for test) | 10 | 7 |
| STL-10 [44] | RGB Objects | $96 \times 96$ | $13,000$ ($5,000$ for training, $8,000$ for test) | 10 | 7 |
| MSRC [235] | RGB Objects | $480 \times 640$ | $4,323$ | 21 | 6 |
| VOC 2007 [62] | RGB Objects | diverse | $12,608$ | 20 | 5, 6 |
| LabelMe [186] | RGB Objects | diverse | $30,369$ images with $111,490$ annotations | 183 | 5, 6 |
| Caltech-101 [88] | RGB Objects | $\sim 300 \times 200$ | $9,146$ images | 101 | 5, 6 |
| SUN09 [40] | RGB Objects | diverse | $> 152,000$ | $> 200$ | 5, 6 |
| Amazon*<br>Webcam*<br>Dslr* | RGB Objects<br>RGB Objects<br>RGB Objects | $300 \times 300$<br>diverse<br>$\sim 1000 \times 1000$ | 958<br>295<br>157 | 31 | 5, 6 |
| IXMAS [229] | Action Videos | $64 \times 48$ | $1,148$ sequences | 11 | 6 |

# 2 Background and Literature Review

*This chapter presents comprehensive backgrounds this thesis including PAC-learning theory, representation learning, deep learning, and kernel methods. It is followed by literature review on the topic of domain adaptation and domain generalization.*

## 2.1 Introduction to Machine Learning

*Machine learning* is a branch of computer science that studies how computer can learn from data without being explicitly programmed [189]. The long-term goal of machine learning is to mimic the learning ability of human and apply it to computers. It is strongly related to *statistics* when using probabilistic or statistical tools to do the analysis. The underlying formal theory behind machine learning is mainly discussed in *computational learning* or *statistical learning* theory. Machine learning is inherently multidisciplinary involving many fields such as artificial intelligence, optimization, information theory, philosophy, psychology, and neurobiology [149]. It has been applied to an enormous range of applications, such as data mining, computer vision, natural language processing, and speech recognition . Some prominent approaches are $k$-nearest neighbor ($k$-NN), decision tree, neural networks, evolutionary computation (including genetic algorithm, genetic programming, particle swarm optimization, etc), probabilistic graphical models, and kernel methods (*e.g.*, support vector machines).

Mitchell [149] formally describes machine learning by such a definition that consists of three main components: 1) *experience*, 2) *task*, and 3) *performance measure*, as follows.

**Definition 1.** *A computer program is said to learn from experience $E$ with respect to some class of tasks $T$ and performance measure $P$, if its performance at tasks in $T$, as measured by $P$, improves with experience $E$.*

To place the definition into an illustrative context, let us consider a problem of object detection. In this case, the task $T$ is to detect objects of interest given an image. The experience $E$ may be obtained by detecting objects in a set of training data; the training set can be provided by human. The performance measure $P$ may be the detection rate, i.e., the ratio of the number of correctly detected objects over the total number of detection.

A simple learning task can be explained by a notion referred to as *concept learning task* [149]. More specifically, let $x \in \mathcal{X}$ be an input data point where $\mathcal{X}$ is the input space. Here we only consider the case of classification. Let $f : \mathcal{X} \to \{-1, 1\}$ be any binary-valued function called the *target concept*,[1] which is a concept or function to be learned. Thus, the target concept $f(\cdot)$ is unknown. Let $h \in \mathcal{H}$ be the *model* or *hypothesis* where $\mathcal{H}$ is the set of all possible hypotheses. Here a hypothesis $h$ can be viewed as one of the possible solutions for estimating $f$. Hence, it naturally has the same mapping scenario

[1]As in [149], the concept learning task is represented by binary-valued problem. However, we can easily extend this formulation into multivalued function. Here we consider only the case of binary-valued for the sake of notation simplicity.

as the target function, i.e., $h : \mathcal{X} \rightarrow \{0, 1\}$. The concept learning task is to find a hypothesis $\hat{h}$ given a *training set* $D = \{(x_1, f(x_1)), \ldots, (x_N, f(x_n))\}$ such that $\hat{h}(x_i) \approx f(x_i), \forall i = 1, ..., n$. The process of obtaining $\hat{h}$ is referred to as the *training phase*. After the training is completed, it can then determine the target of unseen instances, which are said to comprise a *test set*. The ability to determine a target correctly for each unseen instance is known as *generalization*, which is the central issue in machine learning both from a theoretical or a practical perspective.

In much of the literature (*e.g.*, [3,24,149], machine learning problems are divided into these following categories : i) *supervised learning*, ii) *unsupervised learning*, iii) *semi-supervised learning*, and iv) *reinforcement learning*. Supervised learning is the case when the training data consists of the input vectors along with their corresponding target vectors. This type of data is typically known as *labeled data*. The tasks that can be included in supervised learning are *classification* and *regression*. Classification is a task of mapping each input vector into one of a finite number of discrete classes, whereas regression maps an input vector into a continuous value. The concept learning discussed in the previous paragraph can be considered as binary classification. Unsupervised learning is the case when we do not have the corresponding target values over the input vectors (*unlabeled data*). An example of unsupervised learning is the discovery of groups of similar examples within the data, which we refer to as *clustering*. Semi-supervised learning combines both labeled and unlabeled data to produce an appropriate model. Reinforcement learning is concerned with how the models or *agents*, can learn and act in a particular environment based on *reward* and *punishment*. Recently, a new type of learning has been defined, which is referred to as *inductive transfer*, *knowledge transfer*, or *transfer learning* [14,54,172,204]. It focuses on leveraging knowledge obtained while solving a particular task and then applying it to a different but related problem. Pan et al. [167] categorized transfer learning into several settings; one of them is called *domain adaptation*, which is the subject of interest in this thesis.

Another important task is to discover *representations* or *features* of data that are informative to solve machine learning tasks. In real-world situations, raw data such as images, video, and speech signals is complex, noisy, and highly variable, which makes the supervised learning process difficult. A possible solution is feature engineering, i.e, manually designing a processing pipeline to extract the desired features. However, feature engineering is often labor intensive and relies on knowledge of expert. It is highly desirable to have learning algorithms that can automate such a process without requiring much expert knowledge. This task is studied under the name of *representation learning* or *feature learning* [19].

## 2.2 PAC Learning Theory

To establish machine learning as science rather than art, a formal theory is needed. In more detail, one wants to seek the answers of some fundamental questions: How many examples are needed to achieve successful learning? How poorly does a learning algorithm perform relative to the true target? Are all functions learnable? What can be learned efficiently?

This section presents a theoretical framework proposed by Valiant [218]. The framework is now referred to as the Probably Approximately Correct (PAC) learning [5, 92], a tool that can formalize and address the forementioned issues under the supervised learning setting. The theoretical contributions in Chapter 6 draws on results from the PAC framework.

As is discussed in the previous section, machine learning deals with approximating an unknown concept or function with another function, that is, a hypothesis, taken from a known family. The PAC framework is concerned with the worst case analysis of the hypothesis in terms of *sample complexity*, *i.e.*, the number of samples needed to achieve an approximate solution with high probability in reasonable time. While it provides a certain level of guarantee, it may not be experimentally useful since we are more interested in the average or best case analysis in practice [169].

We first define prior notations and terminologies used to describe the PAC framework. Let $\mathcal{X}$ be the *input space*, the set of all possible *examples* or *data points* and $\mathcal{Y}$ be its corresponding *label space.* Without loss of generality, we focus on the *binary classification* problem, where $\mathcal{Y} = \{0, 1\}$. A *concept* $c : \mathcal{X} \to \mathcal{Y}$ is a *true*, unknown mapping from $\mathcal{X}$ to $\mathcal{Y}$, which is the target to learn. We denote by $\mathcal{A}$ an algorithm to learn $c$, and $\mathcal{C}$ the set of concepts called *concept class*, where $c \in \mathcal{C}$. Let $h : \mathcal{X} \to \mathcal{Y}$ be a *hypothesis*, which is a possible candidate solution taken from a set of hypotheses $\mathcal{H}$ that may not coincide with $\mathcal{C}$.

The core task of $\mathcal{A}$ is to learn the concept from finite data. Suppose that $S = \{x_1, \ldots, x_n\}$ is an independently and identically distributed (i.i.d.) sample drawn from a fixed unknown distribution $\mathcal{D}$, where $\{c(x_1), \ldots, c(x_n)\}$ is its corresponding labels. The learning task corresponds to selecting a hypothesis $h_S$ from $\mathcal{H}$ such that $h_S$ is the best approximate for $c$.

We naturally need a measure that quantifies how well $h$ approximates $c$. A possible way is to measure the error produced by $h(x)$ with respect to $c(x)$ known as *risk* [221].

**Definition 2** (**Risk**). *Given a hypothesis $h \in \mathcal{H}$, a concept $c \in \mathcal{C}$, and a probability distribution $\mathcal{D}$ from where some samples are drawn, the* ***risk*** *or* ***generalization error*** *of $h$ is define by*

$$R(h) = \mathop{\mathbb{E}}_{x \sim \mathcal{D}}[\mathbf{1}_{h(x) \neq c(x)}], \tag{2.1}$$

*where $\mathbf{1}_\omega$ is an indicator function of event $\omega$. Given an i.i.d. labeled sample of $n$ data $S = \{x_1, \ldots, x_n\} \sim \mathcal{D}$, the expected risk can be estimated by the* ***empirical risk*** *defined by*

$$\hat{R}(h) = \frac{1}{n} \sum_{i=1}^{n} [\mathbf{1}_{h(x_i) \neq c(x_i)}]. \tag{2.2}$$

Ideally, one should use the *true* risk $R(h)$ as a measure for choosing the hypothesis. In practice, however, we can only use the empirical risk $\hat{R}(h)$

calculated from the finite sample $S$, since the data-generating distribution $\mathcal{D}$ is unknown. Note that $\underset{S\sim\mathcal{D}^n}{\mathbb{E}}[\hat{R}(h)] = R(h)$ in the case of an i.i.d. sample $S$.

Recall that $h_S := \arg\min_h R(h)$. Our quantity of interest is $R(h_S)$: how bad $R(h_S)$ is with respect to the sample size $n$ for all possible samples $S \sim \mathcal{D}^n$. PAC-learning provides to answer the question in a probabilistic manner, *i.e.*, $n$ relates to the probability of $R(h_S)$ not exceeding a certain value.

The following presents the formal definition of the PAC-learning framework. We follow the definition introduced by Haussler [92], which incorporates both the approximation uncertainty and the computational complexity. Denote by $O(d)$ an upper bound on the cost of the computational representation of any element $x \in \mathcal{X}$.

**Definition 3** (**PAC-learning**). *A concept class $\mathcal{C}$ is said to be PAC-learnable if there exists an algorithm $\mathcal{A}$ such that for any $\varepsilon > 0$ and $\delta > 0$, for all distributions $\mathcal{D}$ on $\mathcal{X}$ and $\forall c \in \mathcal{C}$, the following inequality holds:*

$$\underset{S\sim\mathcal{D}^n}{\mathbb{P}}[R(h_S) \leq \varepsilon] \geq 1 - \delta \tag{2.3}$$

*for any sample size $n \geq \mathrm{poly}(\frac{1}{\varepsilon}, \frac{1}{\delta}, d)$, where $\mathrm{poly}(\cdot,\cdot,\cdot,\cdot)$ is a polynomial function. The algorithm $\mathcal{A}$ is referred to as a PAC-learning algorithm for $\mathcal{C}$ if it exists. Furthermore, if $\mathcal{A}$ runs in a polynomial time, the class $\mathcal{C}$ is said to be efficiently PAC-learnable.*

The notion $R(h_s) \leq \varepsilon$ is known as the *generalization bound* and $n \geq \mathrm{poly}(\frac{1}{\varepsilon}, \frac{1}{\delta}, d)$ is the *sample complexity bound.* We can see from the definition that there are two conditions for a concept class to be PAC-learnable. First, the hypothesis returned by the algorithm is *approximately correct* with a low error of, at most, $\varepsilon$ and a high probability of, at least, $1 - \delta$. Second, the sample complexity must be polynomial in $\frac{1}{\varepsilon}$, $\frac{1}{\delta}$, and $d$ if the full sample is received by the algorithm.

Note that the PAC-learning framework is a distribution-free framework: no particular assumption is specified for the distribution $\mathcal{D}$ from which samples are drawn. However, it relies on the assumption that the training and test samples are drawn from the same distribution, which is necessary for generalization to be possible in most cases.

Next we highlight several standard generalization bounds based on PAC-learning. Mohri et al. [151] divides the cases of the bounds into two with respect to the hypothesis set $\mathcal{H}$: i) finite $\mathcal{H}$ and ii) infinite $\mathcal{H}$. The following two subsections summarize some parts of chapter 2 and 3 in [151].

### 2.2.1 Bounds for Finite Hypothesis Set

A finite hypothesis set $\mathcal{H}$ is the one whose cardinality $|\mathcal{H}|$ is finite, *i.e.*, $|\mathcal{H}|$ is bounded by a certain positive integer. The following first presents the generalization bound for a finite hypothesis set $\mathcal{H}$, in the case where there exist consistent hypotheses in $\mathcal{H}$ [27,154,222]. A hypothesis $h \in \mathcal{H}$ is said to be consistent *iff* $\hat{R}(h) = 0$.

**Theorem 1** (**Learning bounds for finite, consistent $\mathcal{H}$**)**.** *Let $\mathcal{H} = \{h : \mathcal{X} \to \mathcal{Y}\}$ be a finite hypothesis set and $\mathcal{A}$ be an algorithm that returns a consistent hypothesis $h_S$,* i.e., *$\hat{R}(h_S) = 0$, for any target concept $c \in \mathcal{C}$ and i.i.d. sample $S$ of size $n$. For any $\varepsilon > 0$ and $\delta > 0$, the inequality $\mathbb{P}_{S\sim\mathcal{D}^n}[R(h_S) \leq \varepsilon] \geq 1 - \delta$ holds if the sample complexity is of the form*

$$n \geq \frac{1}{\varepsilon}\left(\log\frac{|\mathcal{H}|}{\delta}\right). \tag{2.4}$$

*Consequently, the following equivalent statement as a generalization bound also holds: for any $\varepsilon > 0$ and $\delta > 0$, with probability at least $1 - \delta$,*

$$R(h_S) \leq \frac{1}{n}\left(\log\frac{|\mathcal{H}|}{\delta}\right). \tag{2.5}$$

*Proof.* Recall that the algorithm $\mathcal{A}$ returns a consistent hypothesis $h_S \in \mathcal{H}$. There may be several consistent hypotheses in $\mathcal{A}$ and we do not know which one is selected by $\mathcal{A}$. One can provide a general bound that holds for all consistent hypotheses, which is referred to as a *uniform convergence bound.*

Denote by $\mathcal{H}_c = \{\exists h \in \mathcal{H} : \hat{R}(h) = 0 \cap R(h) > \varepsilon\}$ a set of consistent hypotheses with risk more than a fixed $\varepsilon > 0$. The following probability holds:

$$\begin{aligned} \mathbb{P}[\mathcal{H}_c] &\leq^{(a)} \sum_{h \in \mathcal{H}} \mathbb{P}[\hat{R}(h) = 0 \cap R(h) > \varepsilon] \\ &\leq^{(b)} \sum_{h \in \mathcal{H}} \mathbb{P}[\hat{R}(h) = 0 | R(h) > \varepsilon], \end{aligned} \tag{2.6}$$

where (a) follows the union bound and (b) is from the definition of conditional probability. Take a hypothesis $h$ from $\mathcal{H}$. The following inequality holds:

$$\mathbb{P}[\hat{R}(h) = 0 | R(h) > \varepsilon] \leq (1 - \varepsilon)^n, \tag{2.7}$$

which implies that

$$\sum_{h \in \mathcal{H}} \mathbb{P}[\hat{R}(h) = 0 | R(h) > \varepsilon] \leq |\mathcal{H}|(1 - \varepsilon)^n. \tag{2.8}$$

Setting $|\mathcal{H}|(1 - \varepsilon)^n \leq \delta$, we can derive the sample complexity bound as follows:

$$\begin{aligned} & |\mathcal{H}|(1 - \varepsilon)^n \leq^{(a)} |\mathcal{H}| \exp(-n\varepsilon) \leq \delta \\ \Rightarrow \quad & \log |\mathcal{H}| - n\varepsilon \leq \log \delta \\ \Rightarrow \quad & n \geq \tfrac{1}{\varepsilon}(\log \tfrac{|\mathcal{H}|}{\delta}), \end{aligned}$$

where $(a)$ follows the identity $(1 - x) \leq \exp(-x)$. Substituting $\varepsilon$ in the inequality (2.3) with the above last inequality, the generalization bound (2.5) holds, which concludes the proof. □

Theorem 1 suggests that we can obtain lower generalization error by increasing the sample size $n$ and/or decreasing the cardinality $|\mathcal{H}|$.

In more general cases, the hypothesis set $\mathcal{H}$ may not contain consistent hypotheses so that the hypothesis $h_S$, which is selected by the algorithm $\mathcal{A}$

is not consistent, *i.e.*, $\hat{R}(h_S) > 0$. The question of interest is now whether $\hat{R}(h_S)$ is sufficient close to $R(h_S)$ given a finite sample $S$. To derive the generalization bound for this situation, we need to quantify $|\hat{R}(h) - R(h)|$, which is given by the Hoeffding's inequality [99].

**Lemma 2** (**Hoeffding's Inequality of Risk**). *Let $S$ denote an i.i.d. sample of size $n$ and fix $\varepsilon > 0$. Then, for any hypothesis $h : \mathcal{X} \to 0, 1$, the following inequalities hold:*

$$\underset{S\sim\mathcal{D}^n}{\mathbb{P}}[\hat{R}(h) - R(h) \geq \varepsilon] \leq \exp(-2n\varepsilon^2),$$
$$\underset{S\sim\mathcal{D}^n}{\mathbb{P}}[\hat{R}(h) - R(h) \leq -\varepsilon] \leq \exp(-2n\varepsilon^2).$$

*By the union bound, the following two-sided inequality holds:*

$$\underset{S\sim\mathcal{D}^n}{\mathbb{P}}[|\hat{R}(h) - R(h) \geq \varepsilon|] \leq 2\exp(-2n\varepsilon^2). \tag{2.9}$$

*Proof.* This lemma is the special case of the original Hoeffding's inequality proved in [99] in which the evaluation of the loss function $\mathbf{1}_{h(x_i)\neq y_i}$ acts as the random variable. □

The left term of (2.9) tells us the probability of a "bad situation" happened. The right term represents the upper bound of the probability, which can become smaller if $n$ increases. It suggests that having more samples can minimize the occurrence of the bad situation.

Next we derive the generalization bound for inconsistent hypotheses, which is obtained by generalizing Hoeffding's inequality for multiple hypotheses as follows.

**Theorem 3** (**Learning bounds for finite, inconsistent $\mathcal{H}$**). *Let $\mathcal{H}$ be a finite hypothesis set. For any $\varepsilon > 0$ and $\delta > 0$, the inequality $\mathbb{P}_{S\sim\mathcal{D}^n}[R(h_S) \leq \varepsilon] \geq 1 - \delta$ holds with sample complexity bound of the form*

$$n \geq \frac{1}{2\varepsilon^2} \log \frac{2|\mathcal{H}|}{\delta}. \tag{2.10}$$

*Consequently, with probability at least* $1-\delta$*, the following generalization bound holds:*

$$\forall h \in \mathcal{H}, \qquad R(h) \le \hat{R}(h) + \sqrt{\frac{\log \frac{2|H|}{\delta}}{2n}} \tag{2.11}$$

*Proof.* Let $h_1, \dots, h_{|\mathcal{H}|} \in \mathcal{H}$. By the union bound and Lemma 2 to each hypothesis,

$$\begin{aligned}
&\mathbb{P}[\exists h \in \mathcal{H} : |\hat{R}(h) - R(h)| > \varepsilon] \\
&= \bigcup_{i=1}^{|\mathcal{H}|} \mathbb{P}[|\hat{R}(h_i) - R(h_i)| > \varepsilon|] \\
&\le \sum_{i=1}^{|\mathcal{H}|} \mathbb{P}[|\hat{R}(h_i) - R(h_i)| > \varepsilon|] \\
&\le 2|\mathcal{H}| \exp(-2n\varepsilon^2)
\end{aligned}$$

Setting the last term to be equal to $\delta$ and then solving for $n$ and $\varepsilon$ conclude the proof. □

It is instructive to compare Theorem 3 with Theorem 1. They are similar in the sense that the generalization error can be reduced by increasing the sample size $n$. In the inconsistent case, however, we need to quadratically increase the sample size to retain the same guarantee as in the consistent case. Another remark is that the bound suggests finding a trade-off between reducing $\hat{R}(h)$ versus controlling the size of the hypothesis set: a larger cardinality $|\mathcal{H}|$ is penalized by the second term but could help reduce $\hat{R}(h)$.

Although Theorem 3 is sufficiently general, it is uninformative when dealing with infinite hypothesis sets. The next subsection presents the learning bounds in the case when the hypothesis set is infinite, which is likely to happen in more general settings.

### 2.2.2 Bounds for Infinite Hypothesis

A general idea to derive sample complexity bounds or generalization bounds for infinite hypotheses is to reduce the infinite case to the analysis of finite sets of hypotheses. For doing so, we need a quantity, in the form of the complexity for the family of hypotheses, that can replace the role of the cardinality $\mathcal{H}$. In other words, the quantity should produce a generalization bound independent from the *infiniteness* of the hypothesis set.

An example of such complexity notions is the *Rademacher complexity* [12], named after Hans Rademacher. It captures the richness of a family of functions by measuring the degree to which a hypothesis set can fit random noise. The *Rademacher complexity* is defined as follows:

**Definition 4** (**Rademacher Complexity** [12])**.** *Let $G$ be a family of functions mapping from $\mathcal{X} \times \mathcal{Y}$ to $[a, b]$ and $S = \{z_1, ..., z_n\} \subseteq \mathcal{X} \times \mathcal{Y}$ be a fixed sample of size $n$. The empirical Rademacher complexity of $G$ with respect to the sample $S$ is*

$$\hat{\mathfrak{R}}_S(G) = \mathop{\mathbb{E}}_{\boldsymbol{\nu}}\left[\sup_{g \in G} \frac{1}{n} \sum_{i=1}^{n} \nu_i g(z_i)\right], \tag{2.12}$$

*where $\boldsymbol{\nu} = (\nu_1, \ldots, \nu_n)^\top$ are Rademacher variables, with $\nu_i$s independent uniform random variables taking values in $\{-1, +1\}$. The **Rademacher complexity** over all samples of size $n$ is*

$$\mathfrak{R}_n(G) = \mathop{\mathbb{E}}_{S \sim \mathcal{D}^n}\left[\hat{\mathfrak{R}}_S(G)\right]. \tag{2.13}$$

It is worth emphasizing that the function class $G$ does not directly represent the hypothesis set $\mathcal{H}$. That is, for any $h \in \mathcal{H}$, $G$ contains functions that evaluate a tuple $(x, h(x))$ into a real value. Examples of such functions are loss functions.

Here we present the standard generalization bound using the Rademacher complexity with the focus on the classification problem. Before doing so, it is desirable to establish the relationship between the empirical and true Rademacher complexities, since we can only calculate the empirical measure in practice. We first need a concentration measure bound referred to as McDiarmid's inequality [143].

**Lemma 4** (**McDiarmid's Inequality** [143])**.** *Let $X_1, ..., X_n$ be independent random variables taking values in the set $\mathcal{X}$ under a distribution $\mathbb{P}$. Further, let $f : \mathcal{X}^n \to \mathbb{R}$ be a function of $X_1, .., X_n$ that satisfies*

$$\sup_{x_1 \dots x_n, x_i' \in \mathcal{X}} \Big| f(x_1 \dots x_i \dots x_n) - f(x_1 \dots x_i' \dots x_n) \Big| \leq c_i,$$

*where $x_i \neq x_i'$ and $1 \leq i \leq n$. The following inequality holds for all $\epsilon > 0$*

$$\mathbb{P}\left\{ \Big| \mathbb{E}[f] - f \Big| \geq \epsilon \right\} \leq 2 \exp\left( \frac{-2\epsilon^2}{\sum_{i=1}^n c_i^2} \right).$$

Using PAC-learning and McDiarmid's inequality, we can bound $|\hat{\mathfrak{R}}_S(G) - \mathfrak{R}_n(G)|$ given by the following theorem.

**Theorem 5.** *Let $G$ be a family of functions mapping from $\mathcal{X} \times \mathcal{Y}$ to $[0, 1]$ and a fixed sample $S = \{z_1, \dots, z_n\} \subseteq \mathcal{X} \times \mathcal{Y}$. With probability $1 - \delta$, the following inequality holds:*

$$\mathfrak{R}_n(G) \leq \hat{\mathfrak{R}}_S(G) + \sqrt{\frac{\log \frac{2}{\delta}}{2n}}$$

*Proof.* Let $S$ and $S'$ be two samples differing by exactly one point, *e.g.*, $z_j \in S$ and $z_j' \in S'$. Using Definition 4,

$$\hat{\mathfrak{R}}_S(G) - \hat{\mathfrak{R}}_{S'}(G) \leq \mathbb{E}_{\boldsymbol{\nu}} \left[ \sup_{g \in G} \frac{v_j(g(z_j) - g(z_j'))}{n} \right] \leq \frac{1}{n}.$$

Similarly, we can obtain $\hat{\mathfrak{R}}_{S'}(G) - \hat{\mathfrak{R}}_S(G) \leq \frac{1}{n}$, which implies that $|\hat{\mathfrak{R}}_{S'}(G) - \hat{\mathfrak{R}}_S(G)| \leq \frac{1}{n}$. By Lemma 4, the following inequality holds:

$$\mathbb{P}[|\mathfrak{R}_n(G) - \hat{\mathfrak{R}}_S(G)| \geq \varepsilon] \leq 2 \exp\left(-2\varepsilon^2 n\right)$$

We conclude the proof by applying Definition 3. □

Note that, up to Theorem 5, the Rademacher complexity is defined on a family of functions $G$, which is not a hypothesis set. For deriving a bound in terms of generalization error/risk, we need to get the Rademacher complexity operating on the hypothesis set $\mathcal{H}$. The following lemma relates the empirical Rademacher complexity of the family of loss functions $G$ to that of the hypothesis set $\mathcal{H}$ in the case of zero-one loss.

**Lemma 6.** *Let $\mathcal{H}$ be a hypothesis set taking values in $\{-1, +1\}$ and $G$ be the family of loss functions associated to $\mathcal{H}$ for the zero-one loss: $G = \{(x, y) \mapsto \mathbf{1}_{h(x) \neq y}\}$. For any sample $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, denote by $S_{\mathcal{X}} = \{x_1, \ldots, x_n\}$ the projection of $S$ over $\mathcal{X}$. Then, the following equality holds:*

$$\hat{\mathfrak{R}}_S(G) = \frac{1}{2}\hat{\mathfrak{R}}_{S_{\mathcal{X}}}(\mathcal{H})$$

*Proof.* By definition of $\hat{\mathfrak{R}}_S(G)$,

$$\begin{aligned}
\hat{\mathfrak{R}}_S(G) &= \mathop{\mathbb{E}}_{\boldsymbol{\nu}}\left[\sup_{h \in \mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \nu_i \mathbf{1}_{h(x_i) \neq y_i}\right] \\
&=^{(a)} \mathop{\mathbb{E}}_{\boldsymbol{\nu}}\left[\sup_{h \in \mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \nu_i \frac{1 - y_i h(x_i)}{2}\right] \\
&= \frac{1}{2}\mathop{\mathbb{E}}_{\boldsymbol{\nu}}\left[\sup_{h \in \mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} -\nu_i y_i h(x_i)\right] \\
&=^{(b)} \frac{1}{2}\mathop{\mathbb{E}}_{\boldsymbol{\nu}}\left[\sup_{h \in \mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \nu_i h(x_i)\right] = \frac{1}{2}\hat{\mathfrak{R}}_{S_{\mathcal{X}}}(\mathcal{H}),
\end{aligned}$$

where (a) follows $\mathbf{1}_{h(x_i) \neq y_i} = \frac{1 - y_i h(x_i)}{2}$ and (b) follows the fact that for a fixed $y_i \in \{-1, +1\}$, $-\nu_i y_i$ and $\nu_i$ are distributed in the same way. □

The above lemma also implies that $\mathfrak{R}_n(G) = \frac{1}{2}\mathfrak{R}_n(\mathcal{H})$. We are now ready to derive generalization bounds for an infinite hypothesis set $\mathcal{H}$ in the case of binary classification using the Rademacher complexity.

**Theorem 7 (Learning bounds for infinite, inconsistent $\mathcal{H}$).** *Let $\mathcal{H}$ be a hypothesis set taking values in $\{-1, +1\}$ and $G$ be the family of loss functions* *associated to $\mathcal{H}$ for the zero-one loss: $G = \{(x, y) \mapsto \mathbf{1}_{h(x)\neq y}\}$. For any $\delta > 0$, with probability at least $1 - \delta$ over a sample $S = \{x_1, \ldots, x_n\}$ drawn from a distribution $\mathcal{D}$ over the input space $\mathcal{X}$, the following inequalities hold for any $h \in \mathcal{H}$:*

$$R(h) \leq \hat{R}(h) + \mathfrak{R}_n(\mathcal{H}) + \sqrt{\frac{\log \frac{1}{\delta}}{2n}} \tag{2.14}$$

*and*

$$R(h) \leq \hat{R}(h) + \hat{\mathfrak{R}}_S(\mathcal{H}) + 3\sqrt{\frac{\log \frac{2}{\delta}}{2n}} \tag{2.15}$$

*Proof.* Define a function $\Omega(S) = \sup\limits_{h\in\mathcal{H}} R(h) - \hat{R}_S(h)$. Let $S$ and $S'$ be two samples differing by exactly one point, *e.g.*, $x_j \in S$ and $x'_j \in S'$. Applying McDiarmid's inequality on $\Omega(S)$, we get

$$\Omega(S) \leq \mathop{\mathbb{E}}_{S}[\Omega(S)] + \sqrt{\frac{\log \frac{2}{\delta}}{2n}}. \tag{2.16}$$

We expand the term $\mathbb{E}_S[\Omega(S)]$ as follows:

$$\begin{aligned}
\mathop{\mathbb{E}}_{S}[\Omega(S)] &= \mathop{\mathbb{E}}_{S}[\sup_{h\in\mathcal{H}} R(h) - \hat{R}_S(h)] \\
&= \mathop{\mathbb{E}}_{S}\left[\sup_{h\in\mathcal{H}} \mathop{\mathbb{E}}_{S'}[\hat{R}_{S'}(h) - \hat{R}_S(h)]\right] \\
&\leq^{(a)} \mathop{\mathbb{E}}_{S,S'}\left[\sup_{h\in\mathcal{H}} \hat{R}_{S'}(h) - \hat{R}_S(h)\right] \\
&= \mathop{\mathbb{E}}_{S,S'}\left[\sup_{h\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{h(x'_i)\neq y'_i} - \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{h(x_i)\neq y_i}\right] \\
&= \mathop{\mathbb{E}}_{\boldsymbol{\nu},S,S'}\left[\sup_{h\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \nu_i \mathbf{1}_{h(x'_i)\neq y'_i} - \frac{1}{n}\sum_{i=1}^{n} \nu_i \mathbf{1}_{h(x_i)\neq y_i}\right] \\
&\leq^{(b)} \mathop{\mathbb{E}}_{\boldsymbol{\nu},S'}\left[\sup_{h\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} \nu_i \mathbf{1}_{h(x'_i)\neq y'_i}\right] + \mathop{\mathbb{E}}_{\boldsymbol{\nu},S}\left[\sup_{h\in\mathcal{H}} \frac{1}{n}\sum_{i=1}^{n} -\nu_i \mathbf{1}_{h(x_i)\neq y_i}\right] \\
&= 2\mathfrak{R}_n(G) =^{(c)} \mathfrak{R}_n(\mathcal{H})
\end{aligned} \tag{2.17}$$

where (a) holds by Jensen's inequality, (b) follows the identity $\sup(A+B) \leq \sup(A) + \sup(B)$, and (c) holds by Lemma 6. Combining (2.17) and (2.16) produces Equation (2.14). Finally, substituting $\mathfrak{R}_n(\mathcal{H})$ in (2.14) with the inequality in Theorem 5 yields (2.15), which concludes the proof. $\square$

## 2.3 Representation Learning

Representation learning or feature learning is the basic instrument used in this thesis to solve the dataset bias or domain shift problem. It is concerned with learning representations of data that provide useful information when building classifiers or other predictors [19].

Representation learning is motivated by the fact that supervised learning tasks require input representations or features that are mathematically and computationally convenient to process [55]. However, real world data such as images, video, audio signals are highly complex, inconvenient to process. The standard solution to this problem is to manually develop preprocessing pipelines and data transformations, which extract representations or features that are more useful than the original data [51, 66, 136]. Such feature engineering, however, usually requires intensive human labor, relies on expert knowledge, and often do not generalize well. Therefore, it is highly desirable to *learn* useful representations that can automate and generalize this process.

It is natural to ask and quantify the meaning of "useful representations". There are, at least, two key notions of the usefulness: i) *local invariance* and ii) *the disentanglement of explanatory factors* of data [19]. Local invariance means that the data representations should be insensitive to local changes that are uninformative to the task at hand. In image classification, for example, the noise appearing in a raw image should not alter its representation compared to the noiseless one. A more ambitious goal is to recover factors of variation of the data. For example, we want to disentangle the pose, shape, illumination, morphology, and expression, which are the factors to form face images, given only the raw images. This notion is more general than invariance in the sense that it does not try to discard information. Building invariant features reduces sensitivity in the direction of invariance. The risk of building invariance is of determining the set of features and variations relevant to the task at hand, which is often difficult to do.

In general, representation learning algorithms can be classified into two groups: i) *supervised representation learning* and ii) *unsupervised representation learning*. Examples of supervised representation learning are neural networks/deep learning and supervised dictionary learning [137]. Examples of unsupervised representation learning, which are more dominant, are K-means clustering, a family of linear models: principal component analysis (PCA) [103], independent component analysis (ICA) [108], Fisher's linear discriminant analysis (LDA) [69], and locally linear embedding (LLE) [181], and a class of non-linear models used in deep learning: restricted Boltzmann machines (RBMs) [1,200] and autoencoders [21,30]. Some of aforementioned linear models can be also generalized to non-linear settings by *kernel methods* [193].

The next two sections (2.4 and 2.5) discuss the representation learning approaches used in this thesis: deep learning and feature extraction via *kernels*. Deep learning is concerned with building hierarchical abstractions of representations (more abstract concepts are constructed from less abstract ones), while kernel feature extraction transforms data into *easier-to-classify* high dimensional representations without explicitly computing the transformation by virtue of a similarity function called *kernel*.

## 2.4 Deep Learning

Deep learning refers to a family of representation learning algorithms concerned with learning a hierarchy of representations that model high-level abstractions in data [17, 53]. The hierarchy allows the computer to learn complicated concepts or functions composed from simpler ones. It is theoretically justified that "deep" architectures, which is a hierarchical composition of several non-linearities, can learn the kind of complicated functions more efficiently than "shallow" architectures [22].

While the term "deep learning" itself is relative new, the field can be traced back to 1940s (see a comprehensive historical survey in [191]) and has been rebranded many times. Bengio et al. [20] mention three waves of development of deep learning: i) deep learning known as *cybernetics* in the 1940s-1960s, ii) deep learning known as *connectionism* in the 1980s-1990s, and iii) the current resurgence under the name of *deep learning* since the beginning of 2006.

The names of the fore-mentioned waves of development also reflect the goal and perspective of the field. In early development in this field, deep learning algorithms were intended to be computational models of biological learning – researchers were generally concerned about how learning could happen in the brain. At that time, some of the names that deep learning was recognized are the *perceptron* [180] and *artificial neural networks* (ANNs). In the 1980s, the field tended to go beyond the neuroscientific perspective in that people considered neural networks for solving engineering problems thanks to the backpropagation algorithm [183]. The modern wave of deep learning, started around the mid 2000 period, is identified by breakthroughs in some challenging real-world applications [43, 52, 97, 118, 150, 202, 207, 208]. This is also driven by the availability of massive labeled datasets [80, 185, 186] and the advancement of computational power, which could not be acquired during the first and second waves of development.

This section will describe all necessary aspects of deep learning linked to the main contributions of this thesis. It covers the description of McCulloch-Pitts neuron and the perceptron, feed forward neural networks and backpropagation, greedy-layer wise unsupervised training, and several useful ingredients related to the training and regularization such as optimization, weight initialization, data augmentation, and dropout regularization.

### 2.4.1 McCulloch-Pitts Neuron and The Perceptron

The McCulloch-Pitts neuron or artificial neuron can be considered as the earliest model of brain function [142]. This model is basically a mathematical function that receives one or more inputs (representing *dendrites*) and returns an output (representing an *axon*) in the form of a linear combination of the inputs and their weight values. The output sum is then passed through a non-linear function known as an *activation function*.

The mathematical description of the McCulloch-Pitts neuron is as follows. Let us define $d + 1$ input signals $x_0, \ldots, x_d$ and *weights* $w_0, \ldots, w_d$, where $x_0$ is usually set to $+1$ and $w_0$ is the *bias*. An artificial neuron $h$ is modeled by a function $f : \mathbb{R}^{d+1} \to \mathbb{R}$ given by

$$h = f(x_0, \ldots, x_d) = \sigma\left(\sum_{k=0}^{d} w_k x_k\right), \tag{2.18}$$

where $\sigma : \mathbb{R} \to \mathbb{R}$ is the activation function. In the context of neural networks, several popular choices for the activation function to date are

1. *Linear/Identity*: $\sigma(z) = z$;
2. *Logistic Sigmoid*: $\sigma(z) = \frac{1}{1+\exp(-z)}$;
3. *Hyperbolic Tangent*: $\sigma(z) = \tanh(z)$;
4. *Rectified Linear Unit (ReLU)*: $\sigma(z) = \max(z, 0)$.

As of 2015, the last activation function, ReLU, is the most commonly used activation function [123], see Section 2.4.6 for a more detailed explanation.

It is sometimes more convenient to denote the input signals by a vector $\mathbf{x} \in \mathbb{R}^{d+1}$, and the weights and bias by a vector $\mathbf{w} \in \mathbb{R}^{d+1}$. The function 2.18 can thus be written as $f(\mathbf{x}) = \sigma(\langle \mathbf{w}, \mathbf{x} \rangle)$, where $\langle \cdot, \cdot \rangle$ denotes the dot product. Figure 2.1 depicts an illustration of the McCulloch-Pitts neuron.

The McCulloch-Pitts neuron originally was not associated with the notion of learning. To mimic a particular kind of function, the weights and bias of

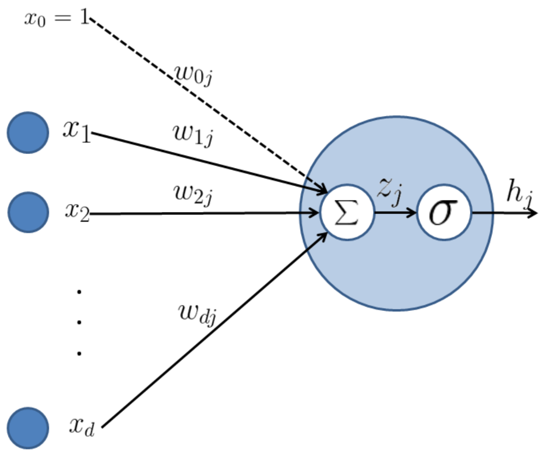


Figure 2.1: McCulloch-Pitts neuron.

the neuron must be predetermined. In 1958, Frank Rosenblatt came up with the *perceptron* [180], the first supervised learning algorithm to train artificial neurons. The perceptron is considered a binary classification algorithm, where the function $f(\cdot)$ returns a single binary value

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \langle \mathbf{w}, \mathbf{x} \rangle > 0 \\ 0 & \text{otherwise.} \end{cases}$$

The above function can be interpreted as an artificial neuron (2.18), where the activation function $\sigma(\cdot)$ is in the form of a *linear threshold function*.

Given a single labeled example $(\mathbf{x}, y) \in \mathbb{R}^{d+1} \times \{0, 1\}$, the perceptron learning corresponds to seeking the optimal weights and bias such that $f(\mathbf{x}) \approx y$. This can be achieved by the following update rule, which is allowed to run until convergence, for the weights and bias, respectively.

$$\begin{aligned} w_k^t &\leftarrow w_k^{t-1} + \alpha(y - f(\mathbf{x}))x_k, \forall k = 0, \ldots, d, \\ b^t &\leftarrow b^{t-1} + \alpha y, \end{aligned}$$

where $\alpha > 0$ is the *step size* or *learning rate*. Note that the above update rule is *online*, that is, it only looks at one example at a time.

A fundamental limitation of the perceptron is that it only works for *linearly separable* data – there exists some hyperplane that puts all the positive examples on one side and all the negative examples on the other side. A famous example of non-linear problems in which the perceptron cannot work is the XOR problem [148]. To deal with more complex problems, we need alternatives that can generate solutions with non-linear decision boundaries. One approach is to combine multiple perceptrons in a single model:*multilayer perceptrons* or *feedforward neural metworks.*

### 2.4.2 Feedforward Neural Networks and Backpropagation

A feedforward neural network is a generalized form of McCulloch-Pitts neurons with multiple intermediate layers, *i.e.*, *hidden layers*, between the input layer and the output layer. Neurons between two adjacent layers are connected, while neurons within a layer are not connected. Figure 2.2 shows the architecture of feedforward neural nets.

**Formal description.** Mathematically, a feedforward neural net model is a composition of McCulloch-Pitts neurons (2.18). We consider a feedforward neural net with one input layer, $L$ hidden layers, and one output layer. Denote the $j$-th neuron in the $l$-th layer by $h_j^{(l)}$. The $l$-th layer can thus be written as a vector $\mathbf{h}^{(l)} \in \mathbb{R}^{k_l+1}$, with $\mathbf{h}^{(0)} := \mathbf{x}$ represents the input layer and $\mathbf{h}^{(l)}, l = 1 \dots L$ represent the hidden layers. Finally, we denote the output layer by $\mathbf{o} \in \mathbb{R}^m$.[2]

Since the model has $k_l$ neurons in each hidden layer, it implies that there exists $k_l$ sets of weights; each corresponds to the connections between $h_j^{(l)}$ and $\mathbf{h}^{(l-1)}$. We denote the connection weights and bias between $h_j^{(l)}$ and $\mathbf{h}^{(l-1)}$ by $\mathbf{w}_j^{(l)} = [w_{0j}^{(l)}, w_{1j}^{(l)}, \dots, w_{k_{l-1}j}^{(l)}]^\top, \forall j = 1, \dots, k_l$. To simplify the notation of

[2]Here we focus on the classification problem, where there are multiple output neurons represented by a single vector.

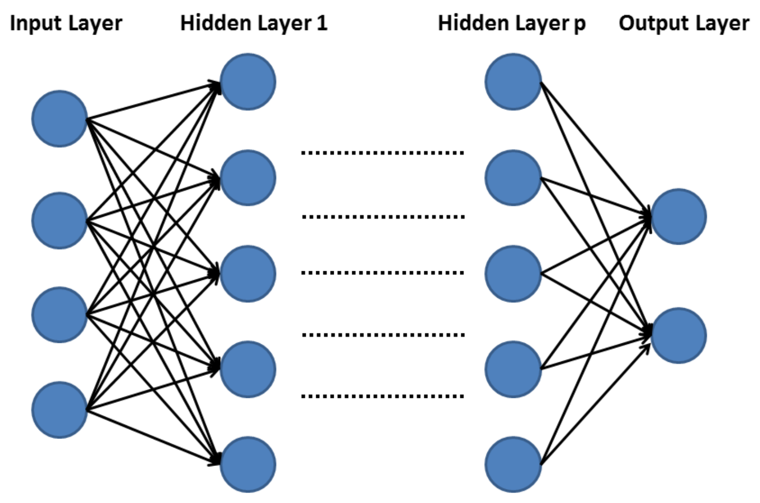


Figure 2.2: Illustration of feedforward neural networks.

the weights and biases in a particular hidden layer, one can express them as a matrix

$$\mathbf{W}^{(l)} = \begin{bmatrix} | & | & \cdots & | \\ \mathbf{w}_1^{(l)} & \mathbf{w}_2^{(l)} & \cdots & \mathbf{w}_{k_l}^{(l)} \\ | & | & \cdots & | \end{bmatrix} \in \mathbb{R}^{(k_{l-1}+1)\times k_l}, \forall l = 1, \ldots, L. \tag{2.19}$$

An analogous notation can be applied for the output layer, denoted by $\mathbf{W}^{\text{out}} \in \mathbb{R}^{k_L \times m}$.

Define a *feedforward mapping* between two adjacent layers $g_{\mathbf{W}^{(l)}} : \mathbb{R}^{k_{l-1}+1} \to \mathbb{R}^{k_l}$ such that

$$g_{\mathbf{W}^{(l)}}(\mathbf{h}^{(l-1)}) = s(\mathbf{W}^{(l)\top}\mathbf{h}^{(l-1)}) =: \mathbf{h}^{(l)}, \forall l = 1, \ldots, L, \tag{2.20}$$

where $s(\mathbf{z}) = [\sigma(z_1) \ldots \sigma(z_{k_l})]^\top$ is the element-wise operation of the activation function $\sigma(\cdot)$. For the output layer, the mapping can be defined as

$g_{\mathbf{W}^{\text{out}}}(\mathbf{h}^{(L)}) =: \mathbf{o}$.

Now we can conveniently define a feedforward neural net $f_{\mathbf{\Theta}} : \mathbb{R}^{d+1} \to \mathbb{R}^m$ parameterized by $\mathbf{\Theta} = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}, \mathbf{W}^{\text{out}}\}$, *i.e.*, the set of all connection weights and biases – recall that $\mathbb{R}^{d+1}$ represents the input space. Specifically, $f_{\mathbf{\Theta}}$ is a composition of feedforward mappings such that

$$f_{\mathbf{\Theta}}(\mathbf{x}) = (g_{\mathbf{W}^{\text{out}}} \circ g_{\mathbf{W}^{(L)}} \circ \cdots \circ g_{\mathbf{W}^{(1)}})(\mathbf{x}), \tag{2.21}$$

where $\circ$ denotes the function composition operation. Similarly to that of the perceptron, the learning problem in feedforward neural nets is to find an optimal set of parameters $\hat{\mathbf{\Theta}}$ such that $f_{\hat{\mathbf{\Theta}}}$ approximates a particular function.

**Backpropagation (BP).** Now we discuss a well known algorithm to train feedforwad neural nets referred to as *backpropagation* (BP). In short, this algorithm is the application of *gradient descent* [37] and the *chain rule* in a single framework. The early development of BP dates back to the 1960s, which was applied in non-neural network contexts [4,58]. BP was then explicitly used to minimize cost functions by adapting control parameters [59,231]. The first specific application of BP in the neural net context was found in 1981 [232]. Rumelhart et al. [183] experimentally demonstrated the emergence of useful internal representations in neural nets via BP, which brought a significant contribution to the popularization of BP. That version of BP, as of this thesis, is still the dominant approach to training neural networks, since it is scalable and can be parallelized on GPUs.

We describe the detailed BP algorithm to train a feedforward neural net $f_{\mathbf{\Theta}}(\cdot)$. in the context of *multi-class classification* problems, where the outputs are the binary vectors in $\{0, 1\}^m$, with all 0s except for a 1 at a particular index. The index of the element 1 corresponds to the class label.

Given a labeled instance $(\mathbf{x}, \mathbf{y}) \in \mathbb{R}^{d+1} \times \{0, 1\}$, we define a *cross-entropy* loss $\ell : \{0,1\}^m \times \{0,1\}^m$ as follows:

$$\ell(\mathbf{y}, f_{\mathbf{\Theta}}(\mathbf{x})) = -\sum_{k=1}^{m} y_k [\log f_{\mathbf{\Theta}}(\mathbf{x})]_k. \tag{2.22}$$

Recall that $f_{\mathbf{\Theta}}(\mathbf{x})$ is a composition of feedforward mappings (2.21). Here the output mapping $g_{\mathbf{W}^{\text{out}}}$ uses the *softmax* function as the activation function. Given a vector $\mathbf{z} \in \mathbb{R}^m$, the softmax function on the $j$-th element is defined by

$$\sigma(z_j) = \frac{\exp(z_j)}{\sum_{k=1}^{m} \exp(z_k)}. \tag{2.23}$$

Suppose that the model can learn from $n$ labeled data $\{(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_n, \mathbf{y}_n)\}$. The BP algorithm solves the following optimization problem:

$$\hat{\mathbf{\Theta}} := \arg\min_{\mathbf{\Theta}} \sum_{i=1}^{n} \ell(\mathbf{y}_i, f_{\mathbf{\Theta}}(\mathbf{x}_i)). \tag{2.24}$$

Since BP is based on gradient descent, it naturally requires the gradient $\nabla_{\mathbf{\Theta}} \ell(\mathbf{y}, f_{\mathbf{\Theta}}(\mathbf{x}))$ that provides a direction towards the optimal solution. In particular, we are interested in computing the error gradient with respect to the individual weight or bias $\frac{\partial \ell}{\partial w_{ij}}$, which can be divided into two cases:

1. **The error gradient for the output layer $\frac{\partial \ell(\cdot,\cdot)}{\partial w_{ij}^{\text{out}}}$.**
   Let $\mathbf{o} := f_{\mathbf{\Theta}}(\mathbf{x})$. By the chain rule,

$$\Delta w_{ij}^{\text{out}} := \frac{\partial \ell(\mathbf{y}, \mathbf{o})}{\partial w_{ij}^{\text{out}}} = \underbrace{\sum_{t=1}^{m} \frac{\partial \ell(\mathbf{y}, \mathbf{o})}{\partial o_t} \frac{\partial o_t}{\partial z_j^{\text{out}}}}_{\frac{\partial \ell(\mathbf{y}, \mathbf{o})}{\partial z_j^{\text{out}}}} \frac{\partial z_j^{\text{out}}}{\partial w_{ij}^{\text{out}}}, \tag{2.25}$$

   where $z_j^{\text{out}} = \sum_i w_{ij}^{\text{out}} h_i^{(L)}$ is the net output of the $j$-th neuron such that $o_j = \sigma(z_j^{\text{out}})$ – recall that the output neuron uses the softmax function for $\sigma(\cdot)$. The detailed derivation of (2.25) is as follows:

$$\begin{aligned}\frac{\partial \ell(\mathbf{y},\mathbf{o})}{\partial o_j} &= -\frac{y_j}{o_j},\\ \frac{\partial o_j}{\partial z_k^{\text{out}}} &= \begin{cases} o_j(1-o_j) & \text{if } j=k \\ -o_j o_k & \text{otherwise}\end{cases}.\end{aligned}$$

Using the above two equations, we can calculate that

$$\frac{\partial \ell(\cdot,\cdot)}{\partial z_j^{\text{out}}} = \sum_{t=1}^{m} \frac{\partial \ell(\mathbf{y},\mathbf{o})}{\partial o_t}\frac{\partial o_t}{\partial z_j^{\text{out}}} = o_j - y_j. \tag{2.26}$$

Given the fact that $\frac{\partial z_j}{\partial w_{ij}^{\text{out}}} = h_i^{(L)}$, it is now straightforward to see that

$$\Delta w_{ij}^{\text{out}} = (o_j - y_j)h_i^{(L)}.$$

2. **The error gradient for the hidden layers** $\frac{\partial \ell(\cdot,\cdot)}{\partial w_{ij}^{(l)}}$**.**
For $l = L, \dots, 1$, by the chain rule we get

$$\Delta w_{ij}^{(l)} := \frac{\partial \ell(\mathbf{y},\mathbf{o})}{\partial w_{ij}^{(l)}} = \underbrace{\sum_{t=1}^{k_{(l+1)}} \frac{\partial \ell(\mathbf{y},\mathbf{o})}{\partial z_t^{(l+1)}}\frac{\partial z_t^{(l+1)}}{\partial h_j^{(l)}}\frac{\partial h_j^{(l)}}{\partial z_j^{(l)}}}_{\frac{\partial \ell(\mathbf{y},\mathbf{o})}{\partial z_j^{(l)}}}\frac{\partial z_j^{(l)}}{\partial w_{ij}^{(l)}}. \tag{2.27}$$

Consider the case of $l = L$, which implies that the index $(l+1)$ corresponds to the output layer. The quantity $\frac{\partial \ell(\mathbf{y},\mathbf{o})}{\partial z_j^{(L)}}$ is thus

$$\frac{\partial \ell(\mathbf{y},\mathbf{o})}{\partial z_j^{(L)}} = \sum_{t=1}^{m}(o_t - y_t)w_{jt}^{\text{out}}\frac{\partial h_j^{(L)}}{\partial z_j^{(L)}} \tag{2.28}$$

Note that $(o_t - y_t)$ is taken from (2.26), the term of which is *backpropagated*, and that $\frac{\partial h_j^{(L)}}{\partial z_j^{(L)}}$ depends on the choice of the activation function

$\sigma(\cdot)$, *e.g.*, $\frac{\partial h_j^{(L)}}{\partial z_j^{(L)}} = h_j^{(L)}(1 - h_j^{(L)})$ if $\sigma(\cdot)$ is the sigmoid function. The error gradient for $l = L$ is therefore given by

$$\Delta w_{ij}^{(L)} = \sum_{t=1}^{m} (o_t - y_t) w_{jt}^{\text{out}} \frac{\partial h_j^{(L)}}{\partial z_j^{(L)}} h_i^{(L-1)}. \tag{2.29}$$

In conclusion, to calculate $\Delta w_{ij}^{(l)}$ for $l = L, \ldots, 1$, we simply calculate $\frac{\partial \ell(\mathbf{y},\mathbf{o})}{\partial z_j^{(l)}}$ recursively, and then multiply by $h_i^{(l-1)}$.

Algorithm 1 summarizes the BP algorithm to train feedforward neural nets with an *online* / *stochastic* setting. That is, the parameter update is calculated for each training example. One can also perform the *offline* / *batch* backpropagation that calculates the average of the error gradient by considering the entire training set, see *e.g.* [126] for a detailed comparison between the *online* and *offline* settings. Another variation of BP is the *mini-batch* setting, *i.e.*, taking the average of the error gradient over each subset of the training set, which is often more desirable in practice.

**Remark.** Feedforward neural nets with backpropagation have led the second wave of deep learning development in 1980s and 1990s. However, it fell out of favour due to several failed cases on training deep architectures and the rise of other learning algorithms such as kernel machines [29, 48, 192] and graphical models [112]. Those algorithms are theoretically well founded and, at that time, provided better performance and faster computation. In 2006, researches found a new strategy to help BP train deep architectures: *greedy layer-wise unsupervised training*. The next three sections describe this strategy in detail.

### 2.4.3 Greedy Layer-Wise Unsupervised Training

Greedy Layer-Wise Unsupervised Training [21,97] enables (deep) neural nets to model the data generating distribution. It trains deep neural nets from

**Algorithm 1** The *online* backpropagation (BP) algorithm.

**Input:**

• Labeled training examples: $S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n \in \mathbb{R}^d \times \{0,1\}^m$
• Set of weights and biases : $\mathbf{\Theta} = \{\mathbf{W}^{(1)}, \dots, \mathbf{W}^{(L)}, \mathbf{W}^{(L+1)}\}$ of a feedforward neural net $f_{\mathbf{\Theta}}(\cdot)$, where $\mathbf{W}^{(L+1)} := \mathbf{W}^{\text{out}}$
• Learning rate: $\alpha > 0$

```
1: Initialize the weights, e.g., $w_{ij}^{(l)} \sim \mathcal{U}[-p,p], \forall i \neq 0$ and the biases $w_{0j}^{(l)} = 0$;
2: while not at end of epoch do
3:     for all $(\mathbf{x}, \mathbf{y}) \in S$ do
4:         for $l = (L+1), L, \dots, 1$ do
5:             Calculate $\Delta w_{ij}^{(l)}$ based on (2.25) or (2.27);
6:             Update the weights and the biases, $\forall i = 0, \dots, k_{(l-1)}, j = 1, \dots, k_l$:
                   $w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \alpha \Delta w_{ij}^{(l)}$;
7:         end for
8:     end for
9: end while
```

**Output:**

• Learned weights and biases: $\hat{\mathbf{\Theta}}$

a set of *unlabeled* examples $\{\mathbf{x}_i\}_{i=1}^n$, one layer at a time starting from the bottom layer. This algorithm can be considered as a "clever" way to initialize the network weights and biases, namely, *pretraining*. The general procedure of pretraining is summarized in Algorithm 2.

The most important step in Algorithm 2 is Step 2: finding an optimal set of parameters at layer $l$. There exists two common approaches to doing so: i) restricted Boltzmann machines (RBMs) and ii) autoencoders (AEs), which will be discussed in Sections 2.4.4 and 2.4.5. The overall pretraining procedure can be viewed as stacking either RBMs or AEs into a single model.

### 2.4.4 Restricted Boltzmann Machines

A possible model as a building block for greedy layer-wise unsupervised training, *i.e.*, Step 2 in Algorithm 2 is the restricted Boltzmann Machine (RBM).

**Algorithm 2** Greedy Layer-Wise Unsupervised Training (Pretraining).

**Input:**
- Unlabeled training examples: $S = \{(\mathbf{x}_i)\}_{i=1}^n \in \mathbb{R}^d$
- Set of weights and biases : $\mathbf{\Theta} = \{\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}\}$ of a neural net $f_{\mathbf{\Theta}}(\cdot)$

```
1: for $l = 1, \ldots L$ do
2:     Given the unlabeled sample $S$, find the optimal weights and biases at
       the layer $l$: $\hat{\mathbf{W}}^{(l)}$;
3:     Compute the layer activation: $\mathbf{h}_i^{(l)} := g_{\hat{\mathbf{W}}^{(l)}}(\mathbf{x}_i), \forall i = 1, \ldots, n$;
4:     Set $\mathbf{h}_i^{(l)}$ as the new inputs to train the next layer: $\mathbf{x}_i \leftarrow \mathbf{h}_i^{(l)}$
5: end for
```

**Output:**
- Learned weights and biases: $\hat{\mathbf{\Theta}}$

An RBM is an energy-based stochastic model or Markov Random Field with a bipartite graph. That is, the model consists of a visible layer and a hidden layer with undirected inter-layer connections; each element in the layers is a (binary) random variable. This model was originally introduced under the name of *Harmonium* in 1986 [200]. The fully connected flavor of RBMs, which also has intra-layer connections, is the one that we refer to as the Boltzmann Machine [1].

Now we describe RBMs more formally. We generally follow the notations used for describing feedforward neural nets in Section 2.4.2. Define a (binary) RBM with a visible layer with $d$ nodes, $\mathbf{x} \in \{0, 1\}^d$, and a hidden layer with $k$ nodes, $\mathbf{h} \in \{0, 1\}^k$.[3] Denote by $\mathbf{W} \in \mathbb{R}^{d \times k}$ the matrix that contains the connection weights, see (2.19) – we ignore the bias terms to simplify the notation.

An RBM has a scalar value associated with each state configuration of the model, which we refer to as the *energy*. This scalar value is represented as an output of the so-called *energy function*. This function has a property that when the nodes are randomly chosen to update, the energy value will either lower or stay the same. Furthermore, under iterative updating the

[3] In general, RBMs can have real valued random variables, see *e.g.*, Cho et al. [39].

machine will eventually converge to a state that is a local minimum in the energy function. The energy function of a binary RBM is given by

$$E(\mathbf{x}, \mathbf{h}; \mathbf{W}) = -\mathbf{x}^\top \mathbf{W}\mathbf{h}. \tag{2.30}$$

The RBM performs sampling: repeatedly choosing a unit and setting its state according to a joint probability distribution, which is referred to as *Boltzmann distribution* (also called the *Gibbs distribution*) [76, 121]. The distribution is given by

$$P_\mathbf{W}(\mathbf{x}, \mathbf{h}) = \frac{\exp(-E(\mathbf{x}, \mathbf{h}; \mathbf{W}))}{Z}, \tag{2.31}$$

where $Z = \sum_{\mathbf{x},\mathbf{h}} \exp(-E(\mathbf{x}, \mathbf{h}; \mathbf{W}))$ is the normalizing constant (also called the partition function). A configuration of the RBM has thus a high probability when the energy $E$ is low. In the sampling process, the global state of the RBM will eventually follow $P_\mathbf{W}$ at *thermal equilibrium* regardless of the distribution of the initial state.

Using (2.31), one can derive the data likelihood and the conditional distributions over the visible and/or hidden nodes as follows:

$$\begin{aligned} P_\mathbf{W}(\mathbf{x}) &= \frac{\sum_\mathbf{x} \exp(-E(\mathbf{x}, \mathbf{h}))}{Z} && (2.32)\\ P_\mathbf{W}(h_j = 1|\mathbf{x}) &= \sigma(\mathbf{W}^\top \mathbf{x}) && (2.33)\\ P_\mathbf{W}(x_i = 1|\mathbf{h}) &= \sigma(\mathbf{W}\mathbf{h}), && (2.34) \end{aligned}$$

where $\sigma(z) = \frac{1}{1+\exp(-z)}$ is the logistic function.

The training objective of the RBM is to learn the data generating distribution from an empirical observation of $x$. The following is the RBM learning in the sense of maximum likelihood estimation (MLE). Define a likelihood function $\mathcal{L}(\mathbf{W}|\mathbf{x}) = P_\mathbf{W}(\mathbf{x})$. The learning objective of the RBM is to maximize the data log-likelihood parameterized by $\mathbf{W}$ from a finite sample

$\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ such that

$$\hat{\mathbf{W}} = \arg\max_{\mathbf{W}} \sum_{i=1}^{n} \log \mathcal{L}(\mathbf{W}|\mathbf{x}_i). \tag{2.35}$$

Consider the *online* setting, *i.e.*, $n = 1$. Notice that

$$\log \mathcal{L}(\mathbf{W}|\mathbf{x}) \;=\; \underbrace{\log \sum_{\mathbf{h}} \exp(-E(\mathbf{x}, \mathbf{h}; \mathbf{W}))}_{\phi^+} - \underbrace{\log Z(\mathbf{W})}_{\phi^-} \tag{2.36}$$

From now on, we refer $\phi^+$ and $\phi^-$ as the *positive phase* and the *negative phase*, respectively.

To solve the optimization problem (2.35), one can compute the gradient of (2.36) with respect to the parameter $\mathbf{W}$ given by

$$\frac{\partial \log \mathcal{L}(\mathbf{W}|\mathbf{x})}{\partial \mathbf{W}} \;=\; \frac{\partial \phi^+}{\partial \mathbf{W}} - \frac{\partial \phi^-}{\partial \mathbf{W}}. \tag{2.37}$$

If the optimal $\hat{\mathbf{W}}$ is successfully found such that it maximizes the log-likelihood (2.36) and the model of RBM is assumed to be well-matched to the observed sample, the RBM should be expected to generate data that appear to be similar to the sample, *i.e.*, $\hat{\mathbf{x}} \sim P_{\hat{\mathbf{W}}}(\mathbf{x})$ .

We can further look at (2.37) in more detail, that is, the gradient of the *phases* with respect to an individual weight $w_{ij}$. Observe that

$$\begin{aligned} \frac{\partial \phi^+}{\partial w_{ij}} &= -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{x}) \frac{\partial E(\mathbf{x}, \mathbf{h}; \mathbf{W})}{\partial w_{ij}} = \sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{x}) x_i h_j \\ &= P(h_j = 1|\mathbf{x}) x_i \end{aligned} \tag{2.38}$$

and

$$\begin{aligned}\frac{\partial \phi^-}{\partial w_{ij}} &= -\sum_{\mathbf{x},\mathbf{h}} P(\mathbf{x},\mathbf{h}) \frac{\partial E(\mathbf{x},\mathbf{h};\mathbf{W})}{\partial w_{ij}} = \sum_{\mathbf{x},\mathbf{h}} P(\mathbf{x},\mathbf{h}) x_i h_j \\ &= \sum_{\mathbf{x}} P(\mathbf{x}) P(h_j = 1|\mathbf{x}) x_i. \end{aligned} \tag{2.39}$$

Computing $\frac{\partial \phi^+}{\partial w_{ij}}$ is straightforward as $P(h_j = 1|\mathbf{x})$, since can be calculated using (2.33). However, computing $\frac{\partial \phi^-}{\partial W_{ij}}$ is computationally intractable for a large number of nodes due to the need to sum the joint or marginal distribution over all possible configurations. For the case of using the marginal distribution $P(\mathbf{x})$, the complexity is exponential in the size of $\mathbf{x}$, *i.e.*, the sum runs over $2^d$ for all possible binary combinations of $\mathbf{x}$. This fact naturally leads to the use of approximation methods to compute (2.37).

**Gibbs sampling in RBM.** An example of the approximation that can be used is the Metropolis-Hasting algorithm [91], which is a Markov Chain Monte Carlo (MCMC) method for sampling from probability distributions based on constructing a Markov chain.[4] One can use a class of the Metropolis-Hasting algorithm referred to as *Gibbs sampling* [74]. This algorithm provides an approach to obtaining samples that are approximated from a specified joint probability distribution of two or more random variables when the direct sampling is difficult. It is applicable when computing the conditional distribution of each variable is easier than computing the joint distribution. Hence, *Gibbs sampling* is well-adapted to sampling from the joint distribution of an RBM using a sequence of samples from the conditional distribution. This can be achieved by running the Markov chain until convergence to the *stationary/equilibrium distribution.*

More specifically, given a set of training example $S = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$, the

[4]A *Markov chain* refers to a time discrete stochastic process for which the *Markov property* holds. See a detailed explanation about Markov chain and MCMC, for example, in [33] as it is beyond our scope here.

data log-likelihood gradient over $w_{ij}$ can be rewritten as follows

$$\begin{aligned}
\frac{1}{n}\sum_{\mathbf{x}\in S}\frac{\partial \log \mathcal{L}(\mathbf{W}|\mathbf{x})}{\partial w_{ij}} &= \frac{1}{n}\sum_{\mathbf{x}\in S}[\frac{\partial \phi^+}{\partial w_{ij}} - \frac{\partial \phi^-}{\partial w_{ij}}] \\
&= \frac{1}{n}\sum_{\mathbf{x}\in S}[\mathop{\mathbb{E}}_{P(\mathbf{h}|\mathbf{x})}[x_i h_j] - \mathop{\mathbb{E}}_{P(\mathbf{x},\mathbf{h})}[x_i h_j]], \\
\sum_{\mathbf{x}\in S}\frac{\partial \log \mathcal{L}(\mathbf{W}|\mathbf{x})}{\partial w_{ij}} &\propto \langle x_i h_j\rangle_{data} - \langle x_i h_j\rangle_{model} \qquad (2.40)
\end{aligned}$$

where $\langle x_i h_j\rangle_{data}$ and $\langle x_i h_j\rangle_{model}$ are referred to as *data statistics* and *model statistics*, respectively.

Using the sampling-based approach, the state $h_j$ in *data statistics* can be drawn by samples from $P(h_j = 1|\mathbf{x})$, which is easy to compute, and $x_i$ is taken from the observed data. However, the states $x_i$ and $h_j$ in *model statistics* need to be drawn by samples from the model joint distribution that cannot computed exactly. To illustrate the *Gibbs sampling* running in an RBM, let $\{x_i^{(0)}, h_j^{(0)}\}$ be the initial states of the RBM assigned in the *data statistics*, and $\{x_i^{(\infty)}, h_j^{(\infty)}\}$ be the stationary states of the RBM assigned in the *model statistics*. Assigning the state of $i$-th observed data to $x_i^0$, the other variable states can be obtained by alternating sampling as follows

$$\begin{aligned}
h_j^{(0)} &\sim P(h_j = 1|\mathbf{x}^{(0)}) \\
x_i^{(1)} &\sim P(x_i = 1|\mathbf{h}^{(0)}) \\
h_j^{(1)} &\sim P(h_j = 1|\mathbf{x}^{(1)}) \\
&\cdots \\
x_i^{(\infty)} &\sim P(x_i = 1|\mathbf{h}^{(\infty)}) \\
h_j^{(\infty)} &\sim P(h_j = 1|\mathbf{x}^{(\infty)}). \qquad (2.41)
\end{aligned}$$

Note that the conditional probability over $\mathbf{x}$ or $\mathbf{h}$ can be considered as the *transition probability* in terms of the Markov chain. Since the transition probability is never zero as the result of the logistic function, it is a sufficient condition that guarantees that the Markov chain will converge to the stationary distribution. Therefore, $x_i^\infty$ and $h_j^\infty$ obtained by (2.41) are equal to samples from $P(\mathbf{x}, \mathbf{h})$.

However, the complexity of letting the chain converge to the stationary distribution is high. Hinton [96] proposed *contrastive divergence* learning, which is an efficient approach to learning RBM based on Gibbs sampling, as described in the following subsection.

**Contrastive divergence.** *Contrastive divergence* (CD) learning [96] refers to a Gibbs sampling-based stochastic approximation for RBM learning. It provides a biased estimate for the data log-likelihood gradient described in (2.40). The estimates are obtained by running the Markov chain to obtain the model statistics for only a few full steps instead of until convergence. Each full step consists of updating $\mathbf{h}$ given $\mathbf{x}$ then updating $\mathbf{x}$ given $\mathbf{h}$. Let $k$ be the number of full steps specified for running the chain, the CD with $k$ steps is denoted by $k$-CD. Using $k = 1$ is sometimes sufficient in practice to get correct gradient estimate [96].

The following is a reason why $k$-CD is considered as a biased estimate of the data log-likelihood gradient. Considering a single datapoint $\mathbf{x}$ and setting $\mathbf{x}^{(0)} := \mathbf{x}$, the $k$-CD algorithm computes

$$\begin{aligned} CD(\mathbf{W}, \mathbf{x}^{(0)}, k) = &-\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{x}^{(0)}) \frac{\partial E(\mathbf{x}^{(0)}, \mathbf{h})}{\partial \mathbf{W}} \\ &+ \sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{x}^{(k)}) \frac{\partial E(\mathbf{x}^{(k)}, \mathbf{h})}{\partial \mathbf{W}}. \end{aligned} \tag{2.42}$$

It is instructive to compare the above equation with (2.37). Note that the Gibbs chain is not guaranteed to converge to a stationary distribution if $k$ is finite. This means that $k$-CD is the biased estimate of the data likelihood gradient, $CD(\mathbf{W}, \mathbf{x}, k) \approx \frac{\partial \log \mathcal{L}(\mathbf{W}|\mathbf{x})}{\partial \mathbf{W}}$; the bias will vanish if we set $k \to \infty$.

### 2.4.5 Autoencoders

Another alternative to RBMs that can be used as the pretraining building blocks for deep neural nets is the autoencoder (AE) [21]. It is an unsupervised model similar to RBM but with a fundamentally different learning procedure. That is, AE is a single layer feedforward neural net, where its output signals are the data itself instead of the labels. The objective of the AE is to learn useful *codes* or vectorial representations, usually with lower dimensionality than the input dimensionality, that provide good data reconstructions. AE was initially considered as a compression algorithm, where the codes act as the compressed representations of the data [30]. According to learning theory [131, 221], the compressed representation is also sufficient to obtain good generalization.

Let $f^{\text{ae}}_{\mathbf{\Theta}}(\cdot)$ be an autoencoder parameterized by $\mathbf{\Theta} = \{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}\}$, which is composed by two functions: 1) the *encoder* function $g_{\mathbf{W}^{(1)}} : \mathbb{R}^d \to \mathbb{R}^k$ and 2) the *decoder* $g_{\mathbf{W}^{(2)}} : \mathbb{R}^k \to \mathbb{R}^d$ – all the notations are analogous those for feedforward neural nets as in (2.20) and (2.21). The matrices $\mathbf{W}^{(1)} \in \mathbb{R}^{d\times k}$ and $\mathbf{W}^{(2)} \in \mathbb{R}^{k\times d}$ denote the connection weights for the hidden layer and the output layer, respectively – again the biases are ignored in the notation for the sake of simplicity. Given a single datapoint $\mathbf{x} \in \mathbb{R}^d$, the autoencoder $f^{\text{ae}}_{\mathbf{\Theta}}(\cdot)$ can be formulated as

$$\begin{aligned}\mathbf{h} &= g_{\mathbf{W}^{(1)}}(\mathbf{x}) = s(\mathbf{W}^{(1)\top}\mathbf{x}) \\ \hat{\mathbf{x}} &= g_{\mathbf{W}^{(2)}}(\mathbf{h}) = s(\mathbf{W}^{(2)\top}\mathbf{h}) = f^{\text{ae}}_{\mathbf{\Theta}}(\mathbf{x}),\end{aligned} \tag{2.43}$$

where $s(\mathbf{z}) = [\sigma(z_1), ..., s_{\text{enc}}(z_{d_h})]^\top$ is the function operating element-wise activation. Note that we can set the AE weights to be tied, *i.e.*, $\mathbf{W}^{(2)} = \mathbf{W}^{(1)\top}$, which reduces the number of free parameters.

AE seeks an optimal solution for $\mathbf{\Theta}$ given a finite set of unlabeled data $\{\mathbf{x}_i\}_{i=1}^n$. The objective of the AE learning is as follows:

$$\hat{\mathbf{\Theta}} := \arg\min_{\mathbf{\Theta}} \sum_{i=1}^{n} \ell\left(f^{\text{ae}}_{\mathbf{\Theta}}(\mathbf{x}_i), \mathbf{x}_i\right). \tag{2.44}$$

where $\ell(\cdot,\cdot)$ is the loss function. Typical choices for the loss function are *least-square loss* $\ell(\mathbf{y},\mathbf{z}) = \|\mathbf{y}-\mathbf{z}\|_2^2$ or *cross-entropy loss* (2.22). Since an AE is basically a feedforward neural net, it can naturally be trained using backpropagation, see Algorithm 1. If we apply autoencoders to raw pixels of visual object images, the weights $\mathbf{W}^{(1)}$ usually form visually meaningful "filters" that can be interpreted qualitatively.

For discriminative tasks, we are interested in obtaining useful data representations from the learned autoencoder. To do so, let us consider a single datapoint $\mathbf{x}'$, which might not be from the training set. The representation of interest is obtained by $g_{\hat{\mathbf{W}}^{(1)}}(\mathbf{x}') =: \mathbf{h}'$ and can be then fed to any classifier.

Several flavors of AEs exist that aim to provide "better" features in the sense of discriminative tasks such as, sparse autoencoders (SAEs) [175], denoising autoencoders (DAEs) [225], contractive autoencoder (CAEs) [178], and saturating autoencoders (SATAEs) [85]. Next we discuss some of them in more detail.

**Sparse Autoencoders (SAEs).** As is discussed, the low-dimensional representation $\mathbf{h}$ induced by the standard AE learning 5.2 is expected to retain sufficient information about the input $\mathbf{x}$ and also to obtain good generalization. However, the choice of the appropriate dimensionality for $\mathbf{h}$ can differ depending on the domain of interest [17]. This is why, for instance, an image compression algorithm normally uses a different number of bits for different images, even if all inputs have the same dimensions. It is therefore more desirable to map each input to a variable-length representation.

One way to allow a flexible dimensionality of $\mathbf{h}$ is to first set a maximum possible value for the dimensionality high enough (normally higher than the input dimensionality). We then let a learning algorithm figure out the appropriate number of *non-zero codes* on its own. Using the standard AE learning to do so, it may suffer from learning an identity function that is unlikely to provide a more useful representation than the input. A possible approach to dealing with this issue is to encourage sparse representations, *i.e.*, the representation $\mathbf{h}$ contains elements with a few non-zero values.

A sparse autoencoder (SAE) [157, 175] aims to obtain sparse representations via a regularization. It is inspired from *sparse coding* [31, 61, 138, 238] that mimics certain properties of biological brain in visual area V1 [162]. Compared to traditional sparse coding approaches, SAEs enjoy the advantage of using the backpropagation algorithm that is highly scalable to the increasing size of the data. Furthermore, sparse coding typically requires a longer procedure to extract sparse representations at test time, which is not the case in SAEs.

The objective of SAE is similar to that of AE, with an additional term $\mathcal{S}(\mathbf{h})$ to (5.2) such that

$$\hat{\boldsymbol{\Theta}} := \arg\min_{\boldsymbol{\Theta}} \sum_{i=1}^{n} \ell\left(f_{\boldsymbol{\Theta}}^{\text{ae}}(\mathbf{x}_i), \mathbf{x}_i\right) + \eta \mathcal{S}\left(\mathbf{h}\right), \tag{2.45}$$

where $\mathcal{S}(\cdot)$ is a sparsity-inducing function. An ideal choice for $\mathcal{S}(\cdot)$ is the $\ell_0$-norm $\|\cdot\|_0$, which induces the sparsest solution. However, computing the $\ell_0$-norm is an NP-hard problem [155]. One often relaxes the sparsity term into $\|\cdot\|_1$, which can also induces the sparsest solution for most large underdetermined systems [57] – in our context, the dimensionality of $\mathbf{h}$ is much larger than that of $\mathbf{x}$. Sparsity-inducing functions for $\mathcal{S}(\cdot)$ other than the two norms above can also be used [122, 174, 175, 243].

**Denoising autoencoders (DAEs).** Good features should be invariant to local changes of the inputs. Denoising autoencoders (DAEs) attempt to improve feature invariance by learning to *denoise* the corrupted inputs [225]. Specifically, the objective is to reconstruct a *clean* input $\mathbf{x}$ given its *corrupted* counterpart $\tilde{\mathbf{x}} \sim \mathcal{Q}(\tilde{X}|X)$, where $\mathcal{Q}(\cdot|\cdot)$ is any noise distribution. This leads to a slightly different optimization problem. That is, given a finite sample of $n$ data $\{\mathbf{x}_i\}_{i=1}^n$ and the corrupted pairs $\{\tilde{\mathbf{x}}_i\}_{i=1}^n$ generated from $\mathcal{Q}(\cdot|\cdot)$, AE minimizes the following objective:

$$\hat{\mathbf{\Theta}} := \arg\min_{\mathbf{\Theta}} \sum_{i=1}^{n} \ell\left(f_{\mathbf{\Theta}}^{\mathrm{ae}}(\tilde{\mathbf{x}}_i), \mathbf{x}_i\right). \tag{2.46}$$

Commonly used types of corruption are zero-masking, Gaussian, and salt-and-pepper noise. Features extracted by DAE have been proven to be more discriminative than those extracted by AE [225].

### 2.4.6 Other Aspects in Deep Learning: ReLU, Dropout, Data Augmentation

In this section, we discuss some other aspects in deep learning that are essential to improve the generalization of neural nets: the Rectified Linear Unit (ReLU) activation function [153], dropout regularization [98], and data augmentation. A (full or partial) combination of these aspects has played an important role to provide state-of-the-art deep learning models to date. Such aspects also slightly change the direction of the deep learning development (around 2011-2012) in the sense of the use of the unsupervised *pre-training* [123]. That is, researchers have begun to omit the unsupervised *pretraining*, see Section 2.4.3, when there exists a massively *labeled* training set in the discriminative task perspective. The aspects explained below have been capable of providing good performing models comparable to or even better than those with pretraining.

**Rectified Linear Unit (ReLU) activation.** As was described in Section 2.4.1, ReLU is the ramp function, *i.e.*, $\sigma(z) = \max(z, 0)$ and the most popular choice for the activation function to date. It was claimed to be more biological plausible than the logistic sigmoid function and provides much faster training for very deep neural networks [78]. The rise of ReLU is perhaps the first sign that the use of unsupervised pretraining in deep neural nets is not necessary anymore for discriminative tasks [109].

The fast computation of ReLU is due to the following reasons. Firstly, it does not involve any exponential computation such as those required in logistic sigmoid and tanh. Secondly, the computation of the ReLU's first derivative is cheap, that is,

$$\frac{\partial \max(0, z)}{\partial z} = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2.47}$$

This also implies that only the weights associated with active neurons are updated during training. Krizhevsky et al. reported that ReLU produces $\sim 6\times$ faster BP training over existing activation functions [118].

**Dropout regularization.** Dropout was first introduced by Hinton et al. [98] as a new regularization to reduce overfitting in deep neural nets, especially when learning from a small sized dataset. It aims to prevent complex co-adaptations on the training data. Co-adaptations can be described as follows. We think of a neuron as a particular feature detector. Since a neuron receives the error signals from other neurons during training, neurons may adapt in a way that they fix up the "mistakes" of the other neurons. Hence, a collection of feature detectors may just explain the training data only, which makes it somewhat useless for the test data.

The dropout procedure is simple, that is, randomly dropping some number of neurons (along with their connections) according to a fixed probability $p$ during training. At test time, the neurons are always present and the weights are multiplied by $p$. Dropout requires an additional step in the feedforward mapping, see also (2.20), as follows:

$$\begin{aligned} \tilde{\mathbf{h}}^{(l)} &= \mathbf{m} * \mathbf{h}^{(l)}, \\ \mathbf{h}^{(l+1)} &= g_{\mathbf{W}^{(l+1)}}(\tilde{\mathbf{h}}^{(l)}), \end{aligned} \tag{2.48}$$

where $\mathbf{m}$ is a masking vector; each element is sampled from a Bernoulli distribution with probability $p$, *i.e.*, $m_j \sim \text{Bernoulli}(p)$

Dropout has significantly improved the performance of neural nets in a wide range of applications such as computer vision, speech recognition, document classification, and computational biology [202]. When training on large-scale labeled data, the use of dropout helps to provide state-of-the-art models without using pretraining. One of the most notable result is of the ImageNet classification task, where dropout plays a significant role to produce the best convolutional neural net [118].

**Data augmentation.** Data augmentation is a method of simply adding extra data into the training set. The augmented data are artificially created from the original data under certain types of transformations that preserve the class labels. In image recognition tasks, the transformations can be in the form of, *e.g.*, object translation, flipping, deformation, random perturbation, color alteration, and reflection, see *e.g.* [42, 197]. This very simple method has been proven to be useful in many situations. Note that, however, the choice of the transformations is usually data-dependent.

In practice, data implementation can be implemented in many ways. In the case of using the *mini-batched* version of backpropagation (BP), a common approach is to perform data augmentation during the batch processing. That is, several instances in a particular batch are transformed before computing the weight update. More specifically, the data augmentation step is done just after Step 3 in Algorithm 1. Such an approach does not require an additional disk storage for the augmented data [118].

### 2.4.7 Convolutional Neural Networks

Convolutional neural networks (ConvNets) are a special kind of feedforward neural net for processing data having a grid-like topology [20]. Examples of such data are audio data and image data, which can be considered as a 1D grid of signal samples and 2D grid of pixels, respectively. There are four main ideas of ConvNets that take advantage of the properties of natural data: local connections, shared weights, pooling, and the use of many hierarchical layers [123]. Like the standard feedforward neural nets, ConvNets can be trained using the back-propagation algorithm.

ConvNets were biologically-inspired by the classic notions of simple cells and complex cells in visual neuroscience [105]. The development of the ConvNet's computation model can be dated back to Neocognitron [71], the model of which has a similar architecture to modern ConvNets, but does not have an end-to-end supervised learning algorithm. The first successful ConvNet model was the so-called LeNet designed for handwritten and machine-printed character recognition [124,125]. ConvNets are the most successful deep learning models to date in the area of computer vision thanks to large-scale data and abundant computational resources [77, 118, 198, 208].

This section describes the architecture of ConvNets, focused on image data as inputs. It is then followed by the types of the ConvNet's layers including their elements and operations.

**Architectures** ConvNets typically consist of the following types of layers in an ordered fashion (from bottom to top): convolutional layer (CONV), pooling layer (POOL), and fully-connected layer (FC). The FC layers are those exactly found in the feedforward neural networks. Non-linear activation functions are applied on the CONV and FC layers, but not applied on the POOL layers.

*Shallow* ConvNets can have an architecture of CONV $\rightarrow$ POOL $\rightarrow$ FC. To construct *deep* ConvNets, the following arrangement is a typical: CONV $\rightarrow$ POOL $\rightarrow \ldots \rightarrow$ CONV $\rightarrow$ POOL $\rightarrow$ FC $\rightarrow \ldots \rightarrow$ FC. That is, any CONV layer is usually followed by a POOL layer, and all FCs compose the top layers, including the output layer. Figure 2.3 illustrates the architecture of a ConvNet.

**Convolutional layer.** Now we describe the elements of the CONV layers in details. In short, there are four important elements: i) feature map, ii) local connectivity, iii) weight sharing, and iv) convolution operation.

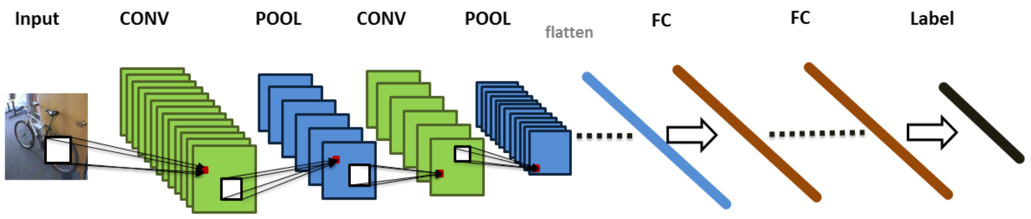


Figure 2.3: Illustration of the ConvNet architecture.

A *feature map* $\mathbf{H} \in \mathbb{R}^{F \times F}$ is a 2D grid that contains real-valued neurons – $\mathbf{H}$ is assumed to be a squared matrix for simplicity. A CONV layer can have $D$ features maps $\mathbf{H}^k, \forall k = 1, \dots, D$. Every neuron in a feature map is connected to only neurons in a local region of the previous layer, *i.e.*, *local connectivity*. These local connections has weights, referred to as the *receptive field* of the neuron, which are *shared* across neurons within a feature map. Such a weight sharing reduces the number of free parameters in the network. It is worth emphasizing that the biases are not shared.

We now describe the *convolution operation* applied in the CONV layers. Without loss of generality, suppose that $\mathbf{X} \in \mathbb{R}^{I \times I}$ is a 2D input grid and $\mathbf{W}^k \in \mathbb{R}^{R \times R}$ is the shared receptive field associated with the $k$-th feature map; both are assumed to be squared matrices. Let $\mathbf{H}^k$ be the $k$-th feature map in the first CONV layer. Every element in $\mathbf{H}^k$, $h^k_{ij}$, is equipped with the following operation:

$$h^k_{ij} = \sigma((\mathbf{W}^k * \mathbf{X})_{ij} + b^k), \tag{2.49}$$

where $*$ denotes the (discrete) 2D convolution operation, $\sigma(\cdot)$ is the activation function, and $b^k$ is the bias. In words, the above convolution is a *sliding window* operation, *i.e.*, we slide the receptive field $\mathbf{W}$ across the width and height of the input $\mathbf{X}$ and compute the matrix dot-product between $\mathbf{W}$ and the local region of $\mathbf{X}$.

In practice, there are two additional hyper-parameters: i) *stride* and ii) *zero-padding*, which are useful when working on 2D inputs. Stride ($S$) controls the pixel steps between two consecutive convolution operations, which usually uses to speed up the computation if the input size is large. Zero-padding ($P$) refers to additional *dark* pixels around the image boundary; $P = 1$ allows the *full convolution*. All above ingredients, including the convolution operation, are best explained in 1D, as shown in Figure 2.4.

Note that the size of feature map, $F \times F$, is a dependent parameter. Recall that $I \times I$ is the input size, $R \times R$ is the receptive field size, $S$ denotes the stride, and $P$ denotes the zero-padding. One of the axes of the feature map size, $F$, can be calculated as

$$F = \frac{I - R + 2P}{S} + 1. \tag{2.50}$$

The above equation is useful to decide a valid configuration for ConvNets in the implementation, that is, when (2.50) produces an integer. For example, if $I = 227$, $R = 11$, $P = 0$, and $S = 4$ as specified in [118], then $F = 55$, which is a valid configuration.

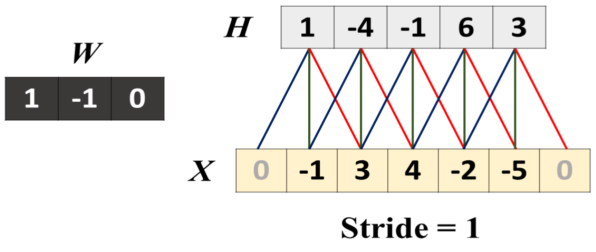
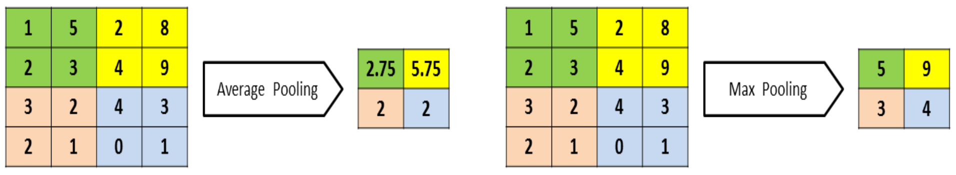


Figure 2.4: Example of the convolution operation in 1D with zero-padding of $P = 1$ and a linear activation function. Colors of the connection lines indicate the shared values. The left figure illustrates the CONV layer operation with stride of $S = 1$, while the right figure illustrates that with $S = 2$.

**Pooling layer.** The POOL layer has the *down-sampling* or *pooling* operation that yields a single value given neurons in a local region of the previous CONV layer. It basically reduces the spatial size of the CONV representation, which implies the reduction of the amount of parameters and overfitting. From the object recognition perspective, the pooling operation ensures to achieve a certain level of translation-invariance.

Figure 2.5 illustrates two common types of pooling operations: i) *average pooling*: taking the averaged value over a local region, and ii) *max pooling*: taking the maximum value over a local region. The latter is now commonly used, following a recent work that shows that the max poling operation is superior compared to the average pooling for capturing invariances in image data [190].



(a) Average Pooling (b) Max Pooling

Figure 2.5: Example of non-overlapping pooling operations with filter size of $2 \times 2$.

In general, the pooling operation can also be interpreted as a sliding window operation as in the CONV layers. For example, think of the max pooling as a "filter" of size $2 \times 2$. The filter is slid across the feature map of size $F \times F$ from the top-left to the bottom-right of the map with $S = 2$ (stride / step) and computes the maximum value of the map's local region. The resulting map is then of size $F/2 \times F/2$, see Figure 2.5(b). Note that $S < 2$ results in an overlapping pooling in this case.

### 2.4.8 Recap

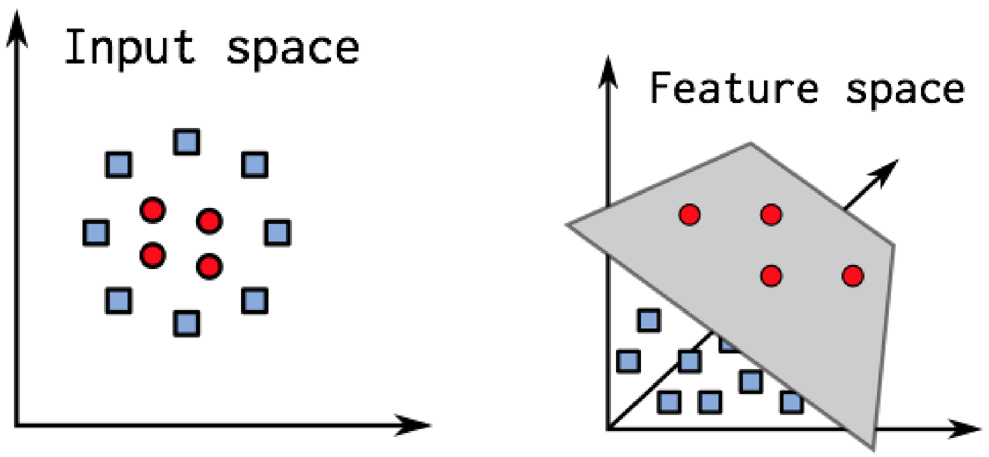
In a nutshell, deep learning is a powerful class of representation learning that has brought some significant advancements in solving real-world problems, especially the computer vision applications. It is arguably the best method to date in harnessing the benefit of large-scale data and computational resources. Deep learning has a long history of development (since the 1940s) under many different names such as cybernetics, connectionism, and artificial neural networks. In the recent wave of the development, the fundamental ideas surprisingly do not change much since 1980s – backpropagation and convolutional networks are still the core elements. Rather, recent successes are mainly driven by the scalability nature of backpropagation in processing a massive amount of labeled data using GPUs and some "small refinements", *e.g.*, the use of unsupervised pretraining, dropout regularization, ReLU activation, and data augmentation.

This thesis investigates the problem of *dataset bias*, which may not be solved by simply applying the off-the-shelf deep learning approaches. Some novel deep learning-based algorithms are presented in Chapters 3, 5, and 7 as solutions to some dataset bias problems in object recognition.

## 2.5 Kernel Methods for Representation Learning

This section presents kernel methods for representation learning, which is the basic framework for our contribution in Chapter 6. Kernel methods form a class of algorithms for pattern analysis that enable any inner product-based linear models to work in non-linear settings efficiently. The use of kernel for pattern analysis was described as early as the 1960s, along with the invention of the kernel perceptron [2]. The best known kernel-based algorithm is the support vector machine (SVM) [29]. Other popular linear algorithms that can be *kernelized* include principal component analysis [192], Fisher's linear discriminant [146], independent component analysis [9], and canonical correlation analysis [120].

From the perspective of representation learning, it is desirable to have a non-linear representation mapping, which converts representations of data into vectorized representations that are almost always *linearly separable*. Figure 2.6 illustrates the effect of such a mapping, where the representations in the higher-dimensional space are linearly separable. It is naturally easier for any supervised algorithm to classify datapoints in Figure 2.6(a) than those in Figure 2.6(b). However, this mapping to high-dimensional space may result in a serious computational drawback. An intriguing property of kernel methods is that the non-linear representation mapping can be computed almost for free. The strategy is known as the *kernel trick* by utilizing a function called *kernel* as a replacement for the dot product between two datapoints.



(a) Nonlinearly separable (2D) (b) Linearly separable (3D)

Figure 2.6: Comparison between a set of two-grouped datapoints in 2D and their transformed representations in 3D. No straight line that can separate the datapoints in (a), but there exist hyperplanes that can separate the representations in (b). The figures are adopted from `http://www.cg.cs.tu-bs.de/static/teaching/seminars/ss13/CG/webpages/SoerenPetersen/`.

In this section, we first introduce *kernel functions* (or just *kernels*) including their properties. We then provide an example of extending principal component analysis into a non-linear setting via kernels.

### 2.5.1 Kernels

To guarantee a certain level of linear separability, one can use some non-linear functions $\phi : \mathcal{X} \to \mathcal{H}$ that map the original data to a high-dimensional Hilbert space $\mathcal{H}$ [5] called a *feature space* [29, 48]. An example for the case where $\mathcal{X}$ is a two-dimensional vector space is $\phi([x_1, x_2]^\top) = [x_1^2, x_2^2, 2x_1x_2]^\top$, the second order polynomial. We can obtain even "better" linear separability if the feature space has higher (possibly infinite) dimensionality than that of the second order polynomial. However, this results in an expensive computation of $\phi(\cdot)$.

Kernel methods allow us to compute $\phi(\cdot)$ in a more efficient way, which is referred to as the *kernel trick.* The fundamental tool for kernel methods is a kind of function referred to as the *kernel* or *kernel function.* This function can be thought as a similarity function between two data points in $\mathcal{X}$.

**Definition 5** (Kernel)**.** *Let $\mathcal{X}$ be a compact set. A function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a kernel over $\mathcal{X}$.*

The basic idea is to define a kernel $\kappa$ such that for any two points $x, x' \in \mathcal{X}$, $\kappa(x, x')$ be equal to an inner product of vectors $\phi(x)$ and $\phi(x')$:

$$\forall x, x' \in \mathcal{X}, \quad \kappa(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}} \tag{2.51}$$

for some non-linear mappings $\phi : \mathcal{X} \to \mathcal{H}$. This means that we can compute the inner product in $\mathcal{H}$ using $\kappa$ only, without going directly to $\mathcal{H}$ via $\phi(\cdot)$. This can bring a significant impact if one can find such kernels: any linear algorithm that only relies on the inner product can be generalized into a non-linear setting efficiently.

Fortunately, the type of kernel expressed in (2.51) exists *iff* it has the so-called *reproducing property.* The kernel equipped with such a property is referred to as the *reproducing kernel*, which was first introduced by Stanislaw

[5] The mathematical definition of Hilbert space can be found in any standard textbook on function analysis, such as [116] and [176].

Zaremba in the 1907 [240]. The associated Hilbert space $\mathcal{H}$ is then called the *reproducing kernel Hilbert space* (RKHS).

**Definition 6** (Reproducing Kernel Hilbert Space)**.** *Let $\mathcal{X}$ be a compact set and $\mathcal{H}$ be a Hilbert space of real-valued functions on $\mathcal{X}$. $\mathcal{H}$ is said to be a reproducing kernel Hilbert space (RKHS) if the evaluation functionals, $L_x : f \mapsto f(x), \forall f(x)$, are bounded,* i.e., *for all $x \in \mathcal{X}$, there exists some $\mathbf{\Lambda}_x > 0$ such that*

$$L_x[f] := f(x) \le \mathbf{\Lambda}_x \|f\|_{\mathcal{H}} \tag{2.52}$$

The above definition of the RKHS is not trivial. For example, the square integrable functions $L_2[a, b]$ can have arbitrarily large values on finite point sets. In this case, no choice of $\mathbf{\Lambda}_x$ will provide the appropriate bound on $L_2[a, b]$ on those point sets [179]. However, it is sufficiently general in the sense that it is the weakest condition that ensures the existence of an inner product. Furthermore, it also ensures the ability to evaluate each function in $\mathcal{H}$ at every point in the domain.

As is mentioned before, the kernel endowed in RKHS has a reproducing property, which can be proved by using the definition 6 and Riesz representation theorem [70, 177].

**Lemma 8** (Reproducing Property)**.** *If $\mathcal{H}$ is an RKHS, then for each $x \in \mathcal{X}$ there exists a function $\kappa(x, \cdot) \in \mathcal{H}$, referred to as the representer of $x$, with the reproducing property*

$$L_x[f] = \langle \kappa(x, \cdot), f \rangle_{\mathcal{H}} = f(x). \tag{2.53}$$

This lemma tells us that the evaluation functional can be represented by taking the inner product with an element of $\mathcal{H}$.

Write $\phi(x) := \kappa(x, \cdot)$. For any $x' \in \mathcal{X}$, one can write

$$\kappa(x, x') = \langle \kappa(x, \cdot), \kappa(x', \cdot) \rangle_{\mathcal{H}} = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}, \tag{2.54}$$

which we define as the *reproducing kernel*, see also (2.51). Furthermore, it is straightforward to see that the *reproducing kernel* satisfies both symmetry and positive definiteness (SPD), since for any $c_i, c_j \in \mathbb{R}$,

$$\begin{aligned}\sum_{i=i,j=1}^{n} c_i c_j \kappa(x_i, x_j) &= \sum_{i=i,j=1}^{n} c_i c_j \langle \kappa(x_i, \cdot), \kappa(x_j, \cdot)\rangle_{\mathcal{H}} \\ &= \|\sum_{j=1}^{n} c_j \kappa(x_j, \cdot)\|_{\mathcal{H}}^2 \geq 0. \end{aligned} \tag{2.55}$$

The following definition provides another view of the positive definiteness of the reproducing kernel, which is more practically useful in the computational viewpoint.

**Definition 7** (Symmetric positive definite kernels). *A kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is said to be symmetric positive definite (SPD) if for any $S = \{x_1, \ldots, x_n\} \in \mathcal{X}$, the matrix $\mathbf{K} = [\kappa(x_i, x_j)]_{ij} \in \mathbb{R}^{n \times n}$ is symmetric positive semi-definite (SPSD).*

$\mathbf{K}$ is referred to as the *Gram matrix* or *kernel matrix* associated to $\kappa$ and the sample $S$. Note that $\mathbf{K}$ is SPSD if it is symmetric and one of the following two equivalent conditions holds, which is the interpretation of the Mercer's theorem [144]:

- for any vector $\mathbf{a} \in \mathbb{R}^n$,

$$\mathbf{a}^\top \mathbf{K} \mathbf{a} = \sum_{i,j=1}^{n} a_i a_j \kappa(x_i, x_j) \geq 0;$$

- the eigenvalues of $\mathbf{K}$ are non-negative.

Below are some examples of SPD kernels. For any $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$,

1. **Linear kernel**:

$$\kappa(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle. \tag{2.56}$$

2. **Polynomial kernel**:

$$\kappa(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + c)^r, \tag{2.57}$$

for any $c > 0$ and $r$ is the degree of the polynomial;

3. **Gaussian kernel or radial basis function (RBF) [2]**:

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right), \tag{2.58}$$

for any $\sigma > 0$;

4. **Sigmoid kernel**:

$$\kappa(\mathbf{x}, \mathbf{x}') = \tanh(a\langle \mathbf{x}, \mathbf{x}' \rangle + b), \tag{2.59}$$

for any $a, b \geq 0$.

In short, our previous explanations state that there exists an SPD kernel in an RKHS. It is natural to ask whether the converse is true: does every SPD kernel define a unique RKHS? This is important in practice when choosing or designing a "valid" kernel – we only need to check that the kernel is symmetric and positive definite. N. Aronszajn provided a theorem, although he attributes it to E. H. Moore, that validates the statement [8]. The theorem is referred to as Moore-Aronszajn theorem,

**Theorem 9** (Moore-Aronszajn theorem)**.** *Let $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be an SPD kernel. Then, there exists a unique Hilbert space $\mathcal{H}$ and a canonical feature map $\phi : \mathcal{X} \to \mathcal{H}$ for which $\kappa$ is a reproducing kernel.*

*Proof.* Define $\phi(x) = \kappa(x, \cdot)$ for all $x \in \mathcal{X}$. Let $\mathcal{H}_0 = \text{span}\{\phi(x) : x \in \mathcal{X}\}$ be a Hilbert space. Let $\mathbf{v} = \sum_{i=1}^{n} \alpha_i \phi(x_i)$ and $\mathbf{w} = \sum_{j=1}^{m} \beta_j \phi(x_j)$, where

$\mathbf{v}, \mathbf{w} \in \mathcal{H}_0$. Define an inner product on $\mathcal{H}_0$, that is, for any $\alpha_i, \beta_j \in \mathbb{R}$,

$$\begin{aligned}\langle \mathbf{v}, \mathbf{w} \rangle_{\mathcal{H}_0} &= \left\langle \sum_{i=1}^{n} \alpha_i \phi(x_i), \sum_{j=1}^{m} \beta_j \phi(x_j) \right\rangle_{\mathcal{H}_0} = \left\langle \sum_{i=1}^{n} \alpha_i \kappa(x_i, \cdot), \sum_{j=1}^{m} \beta_j \kappa(x_j, \cdot) \right\rangle_{\mathcal{H}_0} \\ &=: \sum_{i,j=1}^{n,m} \alpha_i \beta_j \kappa(x_i, x_j).\end{aligned}$$

Note that the above inner product is valid, *i.e.*, symmetric and non-degenerative, following from the symmetry and positive definiteness of $\kappa$. Let $\mathcal{H}$ be the completion of $\mathcal{H}_0$, which implies that $\mathcal{H}$ consists of functions of the form

$$f(x) = \sum_{i=1}^{\infty} c_i \kappa(x, x_i).$$

The following expression shows the reproducing property of $\kappa$, which validates that $\mathcal{H}$ is an RKHS:

$$\langle f, \kappa(x, \cdot) \rangle_{\mathcal{H}} = \left\langle \sum_{i=1}^{\infty} c_i \kappa(\cdot, x_i), \kappa(x, \cdot) \right\rangle_{\mathcal{H}} = f(x).$$

To prove that $\mathcal{H}$ is unique, suppose that $\mathcal{M}$ is another RKHS for which $\kappa$ is a reproducing kernel. For any $x, x' \in \mathcal{X}$,

$$\langle \kappa(x, \cdot), \kappa(x', \cdot) \rangle_{\mathcal{M}} = \kappa(x, x') = \langle \kappa(x, \cdot), \kappa(x', \cdot) \rangle_{\mathcal{H}}, \tag{2.60}$$

which implies that $\mathcal{M} = \mathcal{H}$, *i.e.*, no other RKHS than $\mathcal{H}$ that is defined by the kernel $\kappa$. □

The next subsection provides an example of *kernelizing* a linear model of the form $\mathbf{z} = \mathbf{W}^\top \mathbf{x}$, where $\mathbf{x} \in \mathbb{R}^d$ is the original data, $\mathbf{z} \in \mathbb{R}^k$ is the low-dimensional representation of $\mathbf{x}$ ($k < d$), and $\mathbf{W} \in \mathbb{R}^{d \times k}$ is a collection of coefficients forming a linear transformation. The idea is to alter $\mathbf{z} = \mathbf{W}^\top \phi(\mathbf{x})$ into a kernelized formulation.

### 2.5.2 Kernel PCA

Principal Component Analysis (PCA) seeks a linear projection to a subspace where the variance of the projected data points onto that subspace is maximized. It is perhaps the oldest and best known dimensionality reduction techniques introduced by Pearson [170] and developed independently by Hotelling [103]. PCA is an unsupervised method, since it does not use the label information.

Suppose that $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ are the original data points, which can be written as a matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$; each data point is a row-vector of $\mathbf{X}$. In PCA, we are interested in finding a linear, orthogonal mapping $\mathbf{W} : \mathbb{R}^d \to \mathbb{R}^k$, *i.e.*, $\mathbf{Z} = \mathbf{XW}$, where $\mathbf{Z} \in \mathbb{R}^{n \times k}$ are the projected data with maximum variance and $k < d$. The column-vectors in matrix $\mathbf{W}$ can be interpreted as the bases that span the subspace $\mathbb{R}^k$.

Define the empirical covariance $\mathrm{Cov}(\cdot)$. If we assume that $\frac{1}{n}\sum_{i=1}^n \mathbf{x}_i = \mathbf{0}$, then $\mathrm{Cov}(\mathbf{X}) = \frac{1}{n}\mathbf{X}^\top\mathbf{X}$. The objective of PCA is thus

$$\hat{\mathbf{W}} := \arg\max_{\mathbf{W}} \mathrm{Tr}\, \underbrace{\mathrm{Cov}(\mathbf{XW})}_{\mathrm{Cov}(\mathbf{Z})} \;\text{ s.t. }\; \mathbf{W}^\top\mathbf{W} = \mathbf{I}. \tag{2.61}$$

The Lagrangian of the above objective is

$$J(\mathbf{W}) = \frac{1}{n}\mathrm{Tr}(\mathbf{W}^\top\mathbf{X}^\top\mathbf{XW}) - \mathrm{Tr}(\mathbf{\Lambda}(\mathbf{W}^\top\mathbf{W} - \mathbf{I})), \tag{2.62}$$

where $\mathbf{\Lambda}$ contains the Lagrange multipliers. By evaluating $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = 0$, the following eigenvalue problem is obtained:

$$\frac{1}{n}\mathbf{X}^\top\mathbf{XW} = \mathbf{\Lambda W}, \tag{2.63}$$

where $\mathbf{\Lambda} = \mathrm{diag}(\mathbf{\Lambda}_1, \ldots, \mathbf{\Lambda}_d)$ are the eigenvalues of $\mathrm{Cov}(\mathbf{X})$ and the columns of $\mathbf{W}$ are the corresponding eigenvectors. Therefore, the PCA objective can be solved by performing the eigendecomposition of the covariance matrix $\mathrm{Cov}(\mathbf{X})$. Given a data point $\mathbf{x}' \in \mathbb{R}^d$, the corresponding projected data point is calculated by $\mathbf{z}' = \hat{\mathbf{W}}^\top \mathbf{x}'$; each element of $\mathbf{z}' \in \mathbb{R}^k$ is the *principal component* of $\mathbf{x}'$.

The original data points may not be linearly separable, which is troublesome in the perspective of classification tasks. Note that the standard PCA does not change the non-linear separability of the data points due to its linear nature. One can consider a non-linear model so that the projected data points that are linearly separable can be achieved.

We can generalize PCA to a non-linear setting without changing the optimization procedure, *i.e.*, solving the eigenvalue problem. That is, we operate the PCA algorithm on a high-dimensional, possibly infinite space $\mathcal{H}$ in which the linear separability is almost always guaranteed. The nonlinearity is carried out by the feature map $\phi : \mathbb{R}^d \to \mathcal{H}$, which maps the finite original data points onto $\mathcal{H}$, *i.e.*, $\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n) \in \mathcal{H}$.

Denote by $\mathbf{\Phi}$ a feature matrix, whose rows are the high-dimensional features explained before. In this setting, PCA now aims to find a linear projection $\mathbf{W} : \mathcal{H} \to \mathbb{R}^k$ by solving the following eigenvalue problem:

$$\frac{1}{n}\mathbf{\Phi}^\top \mathbf{\Phi} \mathbf{W} = \mathbf{\Lambda} \mathbf{W}. \tag{2.64}$$

Note that the above formulation is obtained by simply substituting $\mathbf{X}$ with $\mathbf{\Phi}$ in (2.63). However, solving the eigenvalue problem (2.64) is computationally expensive or even undoable, since it involves a direct computation of $\phi(\cdot)$.

If $\mathcal{H}$ is an RKHS endowed with a kernel $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, the direct computation of $\phi(\cdot)$ can be avoided by virtue of the kernel trick. This implies that $\mathbf{\Phi}\mathbf{\Phi}^\top = \mathbf{K}$ forms a Gram matrix, where $K_{ij} = [\kappa(\mathbf{x}_i, \mathbf{x}_j)]$ – the construction of the Gram matrix does not involve an explicit computation of the feature matrix $\mathbf{\Phi}$. Therefore, we need to change the formulation of (2.64) such that it only involves the Gram matrix.

To do so, we first note that all eigenvectors in $\mathbf{W}$ lie in the span of $\phi(\mathbf{x}_1), \ldots, \phi(\mathbf{x}_n)$, that is, for all $j = 1, \ldots, k$ there exist coefficients $b_{1j}, \ldots, b_{nj}$

such that

$$\mathbf{w}_j = \sum_{i=1}^{n} b_{ij}\phi(\mathbf{x}_i).$$

In terms of a matrix notation, it can be written as $\mathbf{W} = \mathbf{\Phi}^\top \mathbf{B}$. Substituting $\mathbf{W}$ and $\mathbf{X}$ in the Lagrangian (2.62), we get

$$J(\mathbf{B}) = \frac{1}{n}\operatorname{Tr}(\mathbf{B}^\top \mathbf{K}^2 \mathbf{B}) - \operatorname{Tr}(\mathbf{\Lambda}(\mathbf{B}^\top \mathbf{K}\mathbf{B} - \mathbf{I})) \tag{2.65}$$

This leads to the following eigenvalue problem as a result of calculating $\frac{\partial J(\mathbf{B})}{\partial \mathbf{B}} = 0$:

$$\begin{aligned} \mathbf{K}^2\mathbf{B} &= n\mathbf{\Lambda}\mathbf{K}\mathbf{B} \\ \Rightarrow \mathbf{K}\mathbf{B} &= \mathbf{\Gamma}\mathbf{B}, \end{aligned} \tag{2.66}$$

where $\mathbf{\Gamma} = \operatorname{diag}(\gamma_1, \dots, \gamma_n)$ are the eigenvalues of $\mathbf{K}$ and the columns of $\hat{\mathbf{B}} = [\mathbf{b}_1, \dots, \mathbf{b}_k] \in \mathbb{R}^{n\times k}$ are the corresponding eigenvectors. Note that now the dimensionality of the linear subspace $k < n$ is not necessarily less than the dimensionality of the original space $d$. Since $\mathbf{K}$ is symmetric, $\mathbf{B}$ must be orthogonal, *i.e.*, $\mathbf{B}^\top\mathbf{B} = I$.

Recall that PCA requires $\mathbf{W}$ to be orthogonal rather than $\mathbf{B}$. Since the computation of (2.66) results in the orthogonality of $\mathbf{B}$, one should normalize $\mathbf{B}$ such that $\mathbf{B}_{new} = \mathbf{B}\mathbf{\Gamma}^{\frac{1}{2}}$. Such a normalization follows from the expression below:

$$\mathbf{W}^\top\mathbf{W} = \mathbf{B}^\top\mathbf{K}\mathbf{B} = \mathbf{B}\mathbf{\Gamma}\mathbf{B} = \mathbf{B}\mathbf{\Gamma}^{\frac{1}{2}}\mathbf{\Gamma}^{\frac{1}{2}}\mathbf{B}.$$

Finally, given an original data point $\mathbf{x}' \in \mathbb{R}^d$ the projection onto the space $\mathbb{R}^k$ can be done by calculating $\mathbf{z}' = \mathbf{W}^\top\phi(\mathbf{x}')$. Again, the kernel trick can be used to avoid the explicit computation of $\phi(\cdot)$ – observe that

$$\mathbf{z}' = \mathbf{B}^\top \mathbf{\Phi} \phi(\mathbf{x}')\mathbf{z}' = \mathbf{B}^\top \begin{bmatrix} \kappa(\mathbf{x}_1, \mathbf{x}') \\ \vdots \\ \kappa(\mathbf{x}_n, \mathbf{x}') \end{bmatrix}.$$

The overall procedure of this nonlinear PCA algorithm is known as kernel Principal Component Analysis (KPCA) [192].

To summarize, KPCA has two intriguing properties that makes it practicable. Firstly, KPCA extends the standard PCA into a nonlinear model that allows the extraction of linearly separable representations over non-linearly separable data. Secondly, such a nonlinearity can be achieved by the same optimization procedure as that of linear PCA, *i.e.*, solving the eigendecomposition of a matrix that results in an exact solution. However, a computational issue of KPCA appears when dealing with a large number of data points – the size of $\mathbf{K}$ depends on the number of data points. Solving the eigendecomposition of a large Gram matrix $\mathbf{K}$ is computationally prohibitive.

## 2.6 Domain Adaptation and Domain Generalization

This section presents the task of domain adaptation and domain generalization. It begins with a formal definition of domain, domain adaptation, and domain generalization. It is then followed by a review of existing work in domain adaptation and domain generalization with the main focus on computer vision applications.

### 2.6.1 Definitions

A *domain* is a probability distribution $\mathbb{P}_{XY}$ on $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are the input and label spaces respectively. For the sake of simplicity, we equate

$\mathbb{P}_{XY}$ with $\mathbb{P}$. The terms domain and distribution are used interchangeably throughout the paper. Let $S = \{x_i, y_i\}_{i=1}^n \sim \mathbb{P}$ be an i.i.d. sample from a domain. It is convenient to use the notation $\hat{\mathbb{P}}$ for the corresponding empirical distribution $\hat{\mathbb{P}}(x, y) = \frac{1}{n}\sum_{i=1}^n \delta_{(x_i,y_i)}(x, y)$, where $\delta$ is the Dirac delta.

We define *domain adaptation* and *domain generalization* as follows.

**Definition 8** (**Domain Adaptation**). *Let $\mathbb{P}^s$ and $\mathbb{P}^t$ be a source and target domain respectively, where $\mathbb{P}^s \neq \mathbb{P}^t$. Denote by $S^s = \{x_i^s, y_i^s\}_{i=1}^{n_s} \sim \mathbb{P}^s$ a labeled sample from the source domain and $S_u^t = \{x_i^t\}_{i=1}^{n_t} \sim \mathbb{P}_X^t$ an unlabeled sample from the target domain. The task of domain adaptation is to learn a good labeling function $f_{\mathbb{P}^t} : \mathcal{X} \to \mathcal{Y}$ given $S^s$ and $S_u^t$ as the training examples.*

**Definition 9** (**Domain Generalization**). *Let $\Delta = \{\mathbb{P}^1, \ldots, \mathbb{P}^m\}$ be a set of $m$ source domains and $\mathbb{P}^t \notin \Delta$ be a target domain. Denote by $S^d = \{x_i^d, y_i^d\}_{i=1}^{n_d} \sim \mathbb{P}^d$ samples drawn from $m$ source domains. The task of domain generalization is to learn a labeling function $f_{\mathbb{P}^t} : \mathcal{X} \to \mathcal{Y}$ given $S^d, \forall d = 1, ..., m$ as the training examples.*

It is instructive to compare these two related definitions. The main difference between domain adaptation and domain generalization is on *the availability of the unlabeled target samples*. Both have the same goal: learning a labeling function $f : \mathcal{X} \to \mathcal{Y}$ that performs well on the target domain. In practice, domain generalization requires $m > 1$ to work well although $m = 1$ might not violate Definition 9. Note that domain generalization can be exactly reduced to domain adaptation if $m = 2$ and $\mathbb{P}_X^t \in \Delta$.

### 2.6.2 Domain Adaptation

Earlier studies on domain adaptation focused on natural language processing, see, *e.g.*, [110] and references therein. A notable algorithm in this area is the structural correspondence learning (SCL) [26]. SCL uses unlabeled data from both source and target domains to model correspondences among features with *pivot* features, that is, those that occur frequently and behave similarly in both domains. The pivot features are then used to learn a mapping from the original feature space to a shared, transformed feature space in which the domain difference is reduced.

Recently, domain adaptation has received increasing attention in computer vision [67, 83, 100, 135, 187, 214]. Readers are encouraged to consult the recent survey in visual domain adaptation [168] for a more comprehensive review. We classify domain adaptation algorithms into three categories: i) the classifier adaptation approach, ii) the selection/reweighting approach, and iii) the feature transformation-based approach.

The **classifier adaptation approach** aims to learn a good, adaptive classifier on a target domain by leveraging knowledge from source or auxiliary domains. Adaptive Support Vector Machines (A-SVMs) [237] utilize *auxiliary classifiers* to adapt a *primary classifier* that performs well on a target domain, where the optimization criterion is similar to standard SVMs. The Domain Adaptation Machine (DAM) [60] employs both a Laplacian manifold regularization (to make use of unlabeled target data) and a sparsity regularization in Least-Squares SVMs [220]. A-SVMs and DAM are examples of successful applications in video concept detection.

The **reweighting/selection approach** reduces sample bias by reweighting or selecting source instances that are 'close' to target instances – Selection can be considered as the 'hard' version of reweighting. The basic idea has been studied under the name of *covariate shift* [196]. Gong et al. [81] applied a convex optimization strategy to select some source images that are maximally similar to the target images according to Maximum Mean Discrepancy [86] – referred to as *landmarks*. The landmarks are then used to construct multiple auxiliary tasks as a basis for composing domain-invariant features. Transfer Joint Matching (TJM) [134] uses a reweighting strategy as part of the algorithm in the form of $\ell_{2,1}$-norm structured sparsity regularization on the source subspace bases.

The **feature transformation-based approach** is perhaps the most popular approach in domain adaptation. Here the notion of transformation has a broad meaning: feature projection, alignment, augmentation, etc. Some metric learning-based methods have been proposed, which can be considered as early work on the Office dataset [119, 187]. A kernelized projection-based algorithm, Transfer Component Analysis (TCA) and its semi-supervised version SSTCA [166], utilizes Maximum Mean Discrepancy (MMD) [86] to extract domain-invariant features for WiFi localization and text classification.

The idea of extracting "intermediate features" to minimize dataset bias by projecting data onto multiple intermediate subspaces was also considered. Sampling Geodesic Flow (SGF) [84] and Geodesic Flow Kernel (GFK) [83] generate multiple subspaces via an interpolation between the source and the target subspace on a Grassmann manifold – a point on the manifold is a subspace. Sampling Spline Flow (SSF) [36] follows the similar idea by utilizing the spline curve computed via rolling maps [107] as the interpolation path.

Subspace Alignment (SA) [67] transforms a source PCA subspace into a new subspace that is well-aligned to a target PCA subspace without requiring intermediate subspaces. More recently, adaptive features can also be extracted by aligning the source and target covariance matrices [205]. Other subspace learning-based methods such as Transfer Sparse Coding (TSC) [133] and Domain Invariant Projection (DIP) [10] make use of MMD, following TCA, to match the source and target distributions in the feature space. One of the methods proposed in [11] follows a similar intuition by using Hellinger distance as an alternative to MMD.

Algorithms based on learning hierarchical non-linear feature or deep learning have recently played a major role in the advancement of domain adaptation [41, 56, 79, 101, 132, 158]. An early attempt addressed large-scale sentiment classification [79], where the concatenated features from fully connected layers of stacked denoising autoencoders (SDA)have been found to be domain-adaptive [225]. Tang and Eliasmith [209] proposed the *sparsely-connected Deep Belief Network* (sDBN) that addresses a similar problem to domain adaptation. That is, sDBN reduces the impact of noise in the target domain that is unseen during training. Since it was designed to be noise-specific, sDBN may still suffer from dataset bias when observing objects with different types of noise. Furthermore, sDBN involves a Gibbs sampling-based denoising procedure that adds the complexity at test time.

It is widely known that deep convolutional networks (ConvNets) [125] are a more natural choice for visual recognition tasks and have achieved significant successes [77, 118, 198]. More recently, ConvNets pretrained on a large-scale dataset, ImageNet, have been shown to be reasonably effective for domain adaptation [118]. They provide significantly better performances than the SURF-based features on the Office dataset [56, 102]. An earlier approach on using a convolutional architecture without pretraining on ImageNet, DLID, has also been explored [41] and performs better than the SURF-based features. However, DLID cannot match the performance of pretrained deep networks.

To further improve the domain adaptation performance, the pretrained ConvNets can be *fine-tuned* under a particular constraint related to minimizing a domain discrepancy measure [72,132,216,217]. Deep Domain Confusion (DDC) [217] utilizes the maximum mean discrepancy (MMD) measure [28] as an additional loss function for the fine-tuning to adapt the last fully connected layer. Deep Adaptation Network (DAN) [132] fine-tunes not only the last fully connected layer, but also some convolutional and fully connected layers underneath, and outperforms DDC. Recently, the deep model proposed in [216] extends the idea of DDC by adding a criterion to guarantee the class alignment between different domains. However, it is limited only to the *semi-supervised* adaptation setting, where a small number of target labels can be acquired. The algorithm proposed in [72], which we refer to as ReverseGrad, handles the domain invariance as a binary classification problem. It thus optimizes two contradictory objectives: i) minimizing label prediction loss and ii) maximizing domain classification loss via a simple *gradient reversal* strategy. ReverseGrad can be effectively applied both in the pretrained and randomly initialized deep networks. The randomly initialized model is also shown to perform well on cross-domain recognition tasks other than the Office benchmark, i.e., large-scale handwritten digit recognition tasks.

Some theoretical studies in domain adaptation have also been proposed. Ben-David et al. [16] presented an early theoretical analysis of domain adaptation, a VC-dimension based generalization bound in classification tasks based on the $d_{\mathcal{A}}$-distance [114]. Mansour et al. [140] extended this work in several ways built on Rademacher complexity [12] and the *discrepancy distance*, as an alternative to $d_{\mathcal{A}}$-distance. Jiang et al. [111] provides a formal analysis of feature learning algorithms in which the second moments of source domain and target domain distributions of the features should be similar.

### 2.6.3 Domain Generalization

The task of domain generalization attempts to mitigate the dataset bias problem similarly to that of domain adaptation. As is previously mentioned, a fundamental difference between domain generalization and domain adaptation problems is that the unlabeled samples are not available in domain generalization. It has recently attracted attention in classification problems, including automatic gating of flow cytometry data [25,152] and object recognition [64,113,236]. Therefore, domain adaptation algorithms, which include unlabeled samples, are generally not applicable to the domain generalization problem.

The problem of domain generalization was formally introduced by Blanchard et al. [25]. The authors proposed a theoretically guaranteed kernel-based classifier that operates on multiple related domains inspired from multi-task learning. The proposed algorithm is effective for solving automatic gating of flow cytometry. Muandet et al. [152] proposed a kernel-based feature learning algorithm, Domain-Invariant Component Analysis (DICA). DICA extends Kernel PCA that learns an invariant transformation across domains by minimizing the difference among multiple source domains and preserving a functional relationship between the features and their labels.

Now we review several works related to computer vision applications. Khosla et al. [113] proposed a multi-task max-margin classifier, which we refer to as Undo-Bias, that explicitly encodes dataset-specific biases in feature space. These biases are used to push the dataset-specific weights to be similar to the global weights. Fang et al. [64] developed Unbiased Metric Learning (UML) based on learning to rank framework. Validated on weakly-labeled web images, UML produces a less biased distance metric that provides good object recognition performance. Xu et al. [236] extended an exemplar-SVM [139] to domain generalization by adding a nuclear norm-based regularizer that captures the likelihoods of all positive samples. The proposed model is denoted by LRE-SVM. Finally, Niu et al. [159] presented a weekly supervised classifier based on multi-instance learning that addresses two issues: i) dealing with noisy labels of web images/videos in the source domain, and ii) reducing dataset bias of the learned classifier.

### 2.6.4 Connection with Transfer Learning

A problem that is closely related to domain adaptation and domain generalization is *transfer learning*. It is concerned with reusing knowledge learned previously to solve new different but related problems [167]. It has gained great interest under some different names: learning to learn, inductive transfer, life-long learning, knowledge consolidation, and context-sensitive learning [212]

Pan and Yang [167] classify transfer learning into several subsettings based on two notions: i) "domain" (see the definition in Section 2.6.1) and ii) "task". The task is defined as a tuple of an output space and a label predictor $\mathcal{T} = \{\mathcal{Y}, f(\cdot)\}$. Suppose that $\mathbb{P}_S$ and $\mathcal{T}_S$ be the source domain and task, and $\mathbb{P}_T$ and $\mathcal{T}_T$ be the target domain and task. The following are three subsettings of transfer learning:

1. *Inductive transfer learning*: $\mathcal{T}_S \neq \mathcal{T}_T$; both are predictive tasks;

2. *Transductive transfer learning*: $\mathcal{T}_S = \mathcal{T}_T$, but $\mathbb{P}_S \neq \mathbb{P}_T$;

3. *Unsupervised transfer learning*: $\mathcal{T}_S \neq \mathcal{T}_T$; but both are not predictive tasks, *i.e.*, no labeled data are available in both source and target domains.

Recalling the definitions in Section 2.6.1, domain adaptation and domain generalization can therefore be included in *transductive transfer learning*. In other words, transfer learning covers wider aspects than domain adaptation and domain generalization, where the task performed on the target domain is not necessarily the same as the source task. Table 2.1 briefly compares the differences or similarities between domain adaptation, domain generalization, and transfer learning.

Table 2.1: Summary of a comparison between domain adaptation, domain generalization, and transfer learning.

| Factors | Standard ML | Domain Adaptation | Domain Generalization | Transfer Learning |
|---|---|---|---|---|
| Domain Mismatch | × | ✓ | ✓ | ✓ |
| Multiple Sources | × | × | ✓ | ✓or × |
| Target Domain | × | ✓ | × | ✓ |
| Same source-target task? | ✓ | ✓ | ✓ | ✓or × |

## 2.7 Related Work

Sections 2.6.2 and 2.6.3 briefly discuss existing work in domain adaptation and domain generalization. This section presents the details of some domain adaptation or domain generalization algorithms that belong to either *feature transformation-based approach* or *classifier adaptation approach*. Specifically, we discuss *transfer component analysis* (TCA) [166], *transfer joint matching* (TJM) [135], *domain-invariant component analysis* (DICA) [152], and *low-rank exemplar svms* (LRE-SVMs) [236].

### 2.7.1 Transfer Component Analysis

Transfer component analysis (TCA) is a feature learning algorithm that learns a domain-invariant representation in a reproducing kernel Hilbert space (RKHS) in the setting of domain adaptation [166]. The basic idea is similar to kernel PCA, that is, finding a projection onto a subspace spanned by some *components* in RKHS, see Section 2.5.2. TCA, however, requires that the projection should be able to produce a representation in which the difference between the source and target distributions is minimized, i.e., *feature-distribution matching*.

Denote by $\mathcal{X}$ a compact input space, $\mathcal{H}$ a reproducing kernel Hilbert space equipped with kernel $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, and $\phi : \mathcal{X} \to \mathcal{H}$ a feature map. Let $S^s = \{x_1^s, \ldots, x_{n_s}^s\} \sim \mathbb{P}^s$ be the source dataset, $S^t = \{x_1^t, \ldots, x_{n_t}^t\} \sim \mathbb{P}^t$ be the target dataset, and $S = S^s \cup S^t$ be the concatenated dataset. To produce a domain-invariant representation, TCA utilizes a probability distribution difference measure known as the maximum mean discrepancy (MMD) [28]. We define the empirical MMD as

$$\mathrm{MMD}(\hat{\mathbb{P}}^s, \hat{\mathbb{P}}^t) = \left\| \frac{1}{n_s} \sum_{i=1}^{n_s} \phi(x_i^s) - \frac{1}{n_t} \sum_{i=1}^{n_t} \phi(x_i^t) \right\|_{\mathcal{H}} \tag{2.67}$$

In TCA, the above MMD is operated in a subspace spanned by several basis vectors or components $\mathbf{w}_1, \ldots, \mathbf{w}_k \in \mathcal{H}$. Define a linear transformation $\mathbf{W} : \mathcal{H} \to \mathbb{R}^k$ composed by the fore-mentioned components, with $k \ll n_s + n_t$. The (squared) MMD with respect to the linear transformation $\mathbf{W}$ can be expressed as

$$\mathrm{MMD}^2_{\mathbf{W}}(\hat{\mathbb{P}}^s, \hat{\mathbb{P}}^t) = \left\| \frac{1}{n_s} \sum_{i=1}^{n_s} \mathbf{W}^\top \phi(x_i^s) - \frac{1}{n_t} \sum_{i=1}^{n_t} \mathbf{W}^\top \phi(x_i^t) \right\|^2_{\mathcal{H}} \tag{2.68}$$

The expansion of equation (2.68) leads to the following expression:

$$\begin{aligned} \mathrm{MMD}^2_{\mathbf{W}}(\hat{\mathbb{P}}^s, \hat{\mathbb{P}}^t) = & \frac{1}{n_s^2} \sum_{i=1,j=1}^{n_s} \left\langle \mathbf{W}^\top \phi(\mathbf{x}_i^s), \mathbf{W}^\top \phi(\mathbf{x}_j^s) \right\rangle \\ & + \frac{1}{n_t^2} \sum_{i=1,j=1}^{n_t} \left\langle \mathbf{W}^\top \phi(\mathbf{x}_i^t), \mathbf{W}^\top \phi(\mathbf{x}_j^t) \right\rangle \\ & - \frac{2}{n_s n_t} \sum_{i=1}^{n_s} \sum_{j=1}^{n_t} \left\langle \mathbf{W}^\top \phi(\mathbf{x}_i^s), \mathbf{W}^\top \phi(\mathbf{x}_j^t) \right\rangle \end{aligned} \tag{2.69}$$

By virtue of the representer theorem, each vector in $\mathbf{W}$ can be expressed as a linear combination of $\{\phi(x_i)\}$, that is, $\mathbf{w}_j = \sum_i b_{ij}\phi(x_i)$. Denote by $\mathbf{\Phi}$ a feature matrix, where $\phi(x_1)^\top, \dots, \phi(x_n)^\top$ are the row-vectors ($n = n_s + n_t$), and $\mathbf{B} \in \mathbb{R}^{n\times k}$ a transformation matrix in which $b_{ij}$ are the elements. Substituting $\mathbf{W} = \mathbf{\Phi}^\top\mathbf{B}$ (the representer theorem in matrix notation) into equation (2.69) yields

$$\begin{aligned}
\mathrm{MMD}^2_{\mathbf{W}}(\hat{\mathbb{P}}^s, \hat{\mathbb{P}}^t) &= \frac{1}{n_s^2}\sum_{i=1,j=1}^{n_s} \left\langle \mathbf{B}^\top\mathbf{\Phi}\phi(x_i^s), \mathbf{B}^\top\mathbf{\Phi}\phi(\mathbf{x}_j^t)\right\rangle \\
&+\frac{1}{n_t^2}\sum_{i=1,j=1}^{n_t} \left\langle \mathbf{B}^\top\mathbf{\Phi}\phi(\mathbf{x}_i^t), \mathbf{B}^\top\mathbf{\Phi}\phi(\mathbf{x}_j^t)\right\rangle \\
&-\frac{2}{n_s n_t}\sum_{i=1}^{n_s}\sum_{j=1}^{n_t} \left\langle \mathbf{B}^\top\mathbf{\Phi}\phi(\mathbf{x}_i^s), \mathbf{B}^\top\mathbf{\Phi}\phi(\mathbf{x}_j^t)\right\rangle \\
= \frac{1}{n_s^2}\mathrm{tr}(\mathbf{K}_s^\top\mathbf{B}\mathbf{B}^\top\mathbf{K}_s) &+ \frac{1}{n_t^2}\mathrm{tr}(\mathbf{K}_t^\top\mathbf{B}\mathbf{B}^\top\mathbf{K}_t) - \frac{2}{n_s n_t}\mathrm{tr}(\mathbf{K}_s^\top\mathbf{B}\mathbf{B}^\top\mathbf{K}_t) \\
&= \mathrm{tr}(\mathbf{B}^\top\mathbf{KLKB}) =: \mathrm{MMD}^2_{\mathbf{B}}(\hat{\mathbb{P}}^s, \hat{\mathbb{P}}^t), \qquad (2.70)
\end{aligned}$$

where $\mathbf{K} = [\mathbf{K}_s|\mathbf{K}_t] \in \mathbb{R}^{n\times n}$ is the kernel matrix and $\mathbf{L} = [L_{ij}]$ is a coefficient matrix with $L_{ij} = \frac{1}{n_s^2}$ if $i, j \in \mathcal{I}(s)$; $L_{ij} = \frac{1}{n_t^2}$ if $i, j \in \mathcal{I}(t)$; and $L_{ij} = -\frac{1}{n_s n_t}$ if $i \in \mathcal{I}(s)$ and $j \in \mathcal{I}(t)$. Note that $\mathcal{I}(s)$ and $\mathcal{I}(t)$ are the set of indices of the source dataset and target dataset, respectively.

The goal of TCA is to find a transformation matrix $\mathbf{B}$ such that the squared MMD of the embedded feature is minimized. In doing so, a regularization term $(tr)(\mathbf{B}^\top\mathbf{B})$ is usually needed to control the complexity of $\mathbf{B}$. The optimization problem of TCA then reduces to

$$\begin{aligned}
\min_{\mathbf{B}\in\mathbb{R}^{n\times k}} \quad & \underbrace{\mathrm{tr}(\mathbf{B}^\top\mathbf{KLKB})}_{\mathrm{MMD}^2_{\mathbf{B}}(\hat{\mathbb{P}}^s,\hat{\mathbb{P}}^t)} + \beta\mathrm{tr}(\mathbf{B}^\top\mathbf{B}) \\
\text{subject to} \quad & \mathbf{B}^\top\mathbf{KHKB} = \mathbf{I}_k, \qquad (2.71)
\end{aligned}$$

where $\beta$ is a trade-off parameter, $\mathbf{I}_k \in \mathbb{R}^{k\times k}$ is an identify matrix, and $\mathbf{H}$ is the centering matrix. The above optimization problem can be solved by generalized eigendecomposition of the form

$$(\mathbf{KLK} + \beta\mathbf{I}_n)\mathbf{B} = \mathbf{KHKB\Lambda}, \qquad (2.72)$$

where $\mathbf{\Lambda} \in \mathbb{R}^{k\times k}$ is a diagonal matrix containing the eigenvalues $\lambda_1 \geq \ldots \geq \lambda_k$.

In [166], TCA is modified such that it can include labels from the source domain. The extended version is referred to as *semi-supervised TCA* (SSTCA).

### 2.7.2 Transfer Joint Matching

Transfer Joint Matching (TJM) is a domain adaptation algorithm that combines two ideas: i) feature-distribution matching and ii) instance reweighting [135]. The way to achieve the feature-distribution matching is similar to that in TCA, that is, minimizing the squared MMD of the representation (2.70). Since the MMD minimization is sometimes not sufficient to achieve a good domain adaptation performance for complex cases, performing instance reweighting in the feature space, i.e., reducing the importance of source instances that are not relevant to the target domain, might be helpful to further improve the performance.

The instance reweighting in TJM is achieved by imposing the $\ell_{2,1}$-norm structured sparsity regularization, that is, $\|\mathbf{A}\|_{2,1} = \sum_i \sqrt{\sum_j A_{ij}^2}$, on the transformation matrix. Specifically, the $\ell_{2,1}$-norm regularization can induce *row-sparsity*, which corresponds to eliminating features. To see this, suppose that $\mathbf{B}_s \in \mathbb{R}^{n_s\times k}$ and $\mathbf{B}_t \in \mathbb{R}^{n_t\times k}$ are the transformation matrices associated with the source data and the target data, respectively, i.e., $\mathbf{B} = \begin{bmatrix}\mathbf{B}_s \\ \mathbf{B}_t\end{bmatrix} \in \mathbb{R}^{n\times k}$. The $\ell_{2,1}$-norm is imposed only on $\mathbf{B}_s$, that is, $\|\mathbf{B}_s\|_{2,1}$ and considered as a regularization in the following optimization problem:

$$\begin{aligned} \min_{\mathbf{B}\in\mathbb{R}^{n\times k}} \quad & \mathrm{tr}(\mathbf{B}^\top\mathbf{KLKB}) + \beta\left(\|\mathbf{B}_s\|_{2,1} + \|\mathbf{B}_t\|_F^2\right) \\ \text{subject to} \quad & \mathbf{B}^\top\mathbf{KHKB} = \mathbf{I}_k. \end{aligned} \tag{2.73}$$

Note that $\|\mathbf{B}_s\|_{2,1}$ is non-differentiable at zero, which is problematic if a gradient-based solution is used to solve the problem 2.73. To resolve this issue, TJM utilizes subgradient methods. Suppose that $\{\mathbf{b}_i\}_{i=1}^n$ are the row elements of $\mathbf{B}$. The subgradient is computed by $\frac{\partial\left(\|\mathbf{B}_s\|_{2,1}+\|\mathbf{B}_t\|_F^2\right)}{\partial\mathbf{B}} = 2\mathbf{GB}$, where $\mathbf{G} = \mathrm{diag}(g_{11}, \ldots, g_{nn})$ is a diagonal sub-gradient of the form

$$g_{ii} = \begin{cases} \frac{1}{2\|\mathbf{b}_i\|}, & \mathbf{x}_i \in S^s, \mathbf{b}_i \neq \mathbf{0} \\ 0, & \mathbf{x}_i \in S^s, \mathbf{b}_i = \mathbf{0} \\ 1, & \mathbf{x}_i \in S^t. \end{cases}$$

Since both **B** and **G** are unknown beforehand, TJM establishes an alternating optimization strategy, where one variable is alternately updated with keeping the other one fixed, see Algorithm 1 in [135].

### 2.7.3 Domain Invariant Component Analysis (DICA)

Domain Invariant Component Analysis (DICA) is perhaps the first feature learning algorithm that deals with the problem of domain generalization. Similar to KPCA and TCA, DICA seeks a linear transformation to a subspace in a reproducing kernel Hilbert space (RKHS) – feature are represented by projected datapoints onto the subspace. DICA features should satisfy two criteria: i) the distance between empirical distributions of the features is minimized and ii) the functional relationship between inputs and outputs is preserved.

Denote by $\mathbb{P}_{XY}$ a joint distribution or domain on $\mathcal{X} \times \mathcal{Y}$, from which labeled data are drawn, with $\mathbb{P}_X$ and $\mathbb{P}_{Y|X}$ are the marginal and conditional distribution, respectively. Let $\mathcal{B}_{XY} = \{\mathbb{P}^{(1)}_{XY}, \ldots, \mathbb{P}^{(D)}_{XY}\}$ be the set of $D$ domains, where the domains are assume to be drawn frame from a single distribution $\mathcal{P}$, and $S^{(i)} = \{(x_j, y_j)\}_{j=1}^{n_i} \sim \mathbb{P}^{(i)}_{XY}, i = 1, \ldots, D$ are the corresponding samples. Since in general $\mathbb{P}^{(i)}_{XY} \neq \mathbb{P}^{(j)}_{XY}, \forall i \neq j = 1, \ldots, D$, getting good generalization using standard machine learning methods is problematic. That is, a model trained on $S^{(i)}$ does not guarantee to perform well on $S^{(j)}$.

DICA aims to extract features that are robust to changes in the marginal distribution $\mathbb{P}_X$, while assuming that $\mathbb{P}_{Y|X}$ is stable or varies smoothly. The robustness is reflected by the two criteria mentioned before. The strategy to achieve each criterion will be discussed in more details below.

**Distribution minimization.** To minimize the distance between distributions $\{\mathbb{P}^{(i)}_X\}_{i=1}^D$ (from now on $\mathbb{P} \triangleq \mathbb{P}_X$), DICA utilizes a measure operating in RKHS

$\mathcal{H}$ that is referred to as the *distributional variance* $\mathbb{V}_{\mathcal{H}}(\underbrace{\{\mathbb{P}^{(1)}, \dots, \mathbb{P}^{(D)}\}}_{\mathcal{B}_X})$. Before formulating $\mathbb{V}_{\mathcal{H}}$, denote by $\mu_{\mathbb{P}}$ a *mean map* defined as follows:

$$\mu : \mathcal{B}_{\mathcal{X}} \to \mathcal{H} : \mathbb{P} \to \mathop{\mathbb{E}}_{x \sim \mathbb{P}}[\phi(x)] =: \mu_{\mathbb{P}}, \tag{2.74}$$

where $\phi : \mathcal{X} \to \mathcal{H}$ is a canonical feature map. Let $\mu_{\bar{\mathbb{P}}} = \frac{1}{D}\sum_{i=1}^{D} \mathbb{P}^{(i)}$ be the average over the domains, the distributional variance is defined as follows:

$$\mathbb{V}_{\mathcal{H}}(\mathcal{B}_{\mathcal{X}}) = \frac{1}{D}\sum_{i=1}^{D} \|\mu_{\mathbb{P}^{(i)}} - \mu_{\bar{\mathbb{P}}}\|^2_{\mathcal{H}}\,. \tag{2.75}$$

The practical problem of the above equation is that $\mathbb{V}_{\mathcal{H}}(\mathcal{B}_{\mathcal{X}})$ cannot be computed directly. However, we can compute its empirical measure given $D$ finite samples $\{S^{(i)}\}_{i=1}^{D}$, that is,

$$\hat{\mathbb{V}}_{\mathcal{H}}(\{S^{(1)}, \dots, S^{(D)}\}) = \frac{1}{D}\sum_{i=1}^{D} \left\|\mu_{\hat{\mathbb{P}}^{(i)}} - \mu_{\hat{\bar{\mathbb{P}}}}\right\|^2_{\mathcal{H}}, \tag{2.76}$$

where $\mu_{\hat{\mathbb{P}}^{(i)}} = \frac{1}{n_i}\sum_{j=1}^{n_i} \phi(x_j)$. Let $n = n_1 + \ldots + n_D$ be the total number of samples over all domains. By performing a tedious computation and using the kernel trick, we can express equation (2.80) into a matrix form as follows:

$$\hat{\mathbb{V}}_{\mathcal{H}}(\{S^{(1)}, \dots, S^{(D)}\}) = \operatorname{tr}(\mathbf{KD}) \tag{2.77}$$

with a kernel matrix $\mathbf{K} \in \mathbb{R}^{ntimesn}$ and a coefficient matrix $\mathbf{D} \in \mathbb{R}^{n \times n}$ of the form

$$\mathbf{K} = \begin{pmatrix} \mathbf{K}_{1,1} & \dots & \mathbf{K}_{1,D} \\ \vdots & \ddots & \vdots \\ \mathbf{K}_{D,1} & \dots & \mathbf{K}_{D,D}, \end{pmatrix} \tag{2.78}$$

$$\mathbf{D} = \begin{pmatrix} \mathbf{D}_{1,1} & \dots & \mathbf{D}_{1,D} \\ \vdots & \ddots & \vdots \\ \mathbf{D}_{D,1} & \dots & \mathbf{D}_{D,D}, \end{pmatrix} \tag{2.79}$$

where $\mathbf{D}_{i,j} \in \mathbb{R}^{n_i \times n_j}$ equal to $\frac{(D-1)}{(D^2 n_i^2)}$ if $i = j$, and $\frac{-1}{D^2 n_i n_j}$ otherwise. Notice that the empirical distributional variance $\hat{\mathbb{V}}_{\mathcal{H}}$ can be viewed as a generalized form of the squared empirical MMD (2.69).

As in TCA, the distributional variance is utilized to measure the distribution difference in the feature space rather than in the input space. Given a transformation matrix $\mathbf{B} \in \mathbb{R}^{n\times k}$, by the representer theorem we can verify that

$$\hat{\mathbb{V}}_{\mathbf{B}\circ\mathcal{H}}(\{S^{(1)}, \dots, S^{(D)}\}) = \mathrm{tr}(\mathbf{B}^\top \mathbf{KDKB}). \tag{2.80}$$

DICA attempts to find $\mathbf{B}$ such that the latter measure is minimized.

**Label preservation.** In addition to minimizing the distribution difference, DICA also encourages the extracted features preserving the functional relationship between $X$ and $Y$ by means of a notion called the *central subspace*. Denote by $W$ a central subspace, which is the minimal subspace that captures the functional relationship between $X$ and $Y$, i.e., $Y \perp X | W(X)$. Computing the central subspace therefore requires (source) label information. DICA uses the inverse regression framework [130] operating in an RKHS $\mathcal{F}$ to find the central subspace, which is adopted from *covariance operator inverse regression* (COIR) [115].

The most important term of such framework is the so-called *covariance of inverse regressor* denoted by $\Sigma_{y|x}$. Let $\Sigma_{xx}, \Sigma_{yy}, \Sigma_{xy}, \Sigma_{yx}$ be the covariance operators in and between the RKHSes $\mathcal{H}$ and $\mathcal{F}$. It is proven that the bases $\mathbf{W} \in \mathbb{R}^{d\times k}$ of the central subspace coincide with $k$ largest eigenvectors of $\Sigma_{xx}^{-1}\Sigma_{y|x}\Sigma_{xx}$. In other words, $\mathbf{W}$ can be found by solving the eigendecomposition problem associated with the following optimization:

$$\max_{\mathbf{W}\in\mathbb{R}^{d\times k}} \frac{\mathrm{tr}(\mathbf{W}^\top \Sigma_{xx}^{-1}\Sigma_{y|x}\Sigma_{xx}\mathbf{W})}{\mathrm{tr}(\mathbf{W}^\top\mathbf{W})} \tag{2.81}$$

Kim and Pavlovic [115] states that, under a mild assumption, $\Sigma_{y|x}$ can be expressed in terms of the covariance operators

$$\Sigma_{y|x} = \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{yx}, \tag{2.82}$$

for all $f \in \mathcal{H}$ and there exists $g \in \mathcal{F}$ such that $\mathbb{E}[f(X)|y] = g(y)$ for almost every $y \in \mathcal{Y}$. $\Sigma_{y|x}$ can be approximated by its empirical term $\hat{\Sigma}_{y|x}$ given the finite sets of samples $\{S^{(i)}\}_{i=1}^{D}$. Denote by $\mathbf{\Phi}_x = [\phi(x_1), \ldots, \phi(x_n)]$ an input feature matrix and $\mathbf{\Phi}_y = [\varphi(y_1), \ldots, \varphi(y_n)]$ an output feature matrix, where $\phi : \mathcal{X} \to \mathcal{H}$ and $\varphi : \mathcal{Y} \to \mathcal{F}$ are the feature maps on $\mathcal{X}$ and $\mathcal{Y}$ respectively. The empirical covariance of inverse regressor is expressed as

$$\hat{\Sigma}_{y|x} = \hat{\Sigma}_{xy}\hat{\Sigma}_{yy}^{-1}\hat{\Sigma}_{yx}, \tag{2.83}$$

where $\hat{\Sigma}_{xx} = \frac{1}{n}\Phi_x^\top \Phi_x$, $\hat{\Sigma}_{xy} = \frac{1}{n}\Phi_x^\top \Phi_y$, $\hat{\Sigma}_{yx} = \frac{1}{n}\Phi_y^\top \Phi_x$, and $\hat{\Sigma}_{yy} = \frac{1}{n}\Phi_y^\top \Phi_y$, assuming that the mapped instances $\{\phi(x_i)\}_i$ and $\{\varphi(y_i)\}_i$ are already centered. Again, by virtue of the kernel trick, the optimization (2.81) can be rewritten as finding a linear transformation $\mathbf{B} \in \mathbb{R}^{n\times k}$ such that

$$\max_{\mathbf{B}\in\mathbb{R}^{n\times k}} \frac{\mathrm{tr}(\mathbf{B}^\top \mathbf{K}_y(\mathbf{K}_y + n\epsilon\mathbf{I}_n)^{-1}\mathbf{K}^2\mathbf{B})}{\mathrm{tr}(\mathbf{B}^\top\mathbf{B})}, \tag{2.84}$$

where $\mathbf{K} = \mathbf{\Phi}_x\mathbf{\Phi}_x^\top$ and $\mathbf{K}_y = \mathbf{\Phi}_y\mathbf{\Phi}_y^\top$.

**DICA optimization.** Now we arrive at the overall optimization problem of DICA. DICA seeks a linear transformation $\mathbf{B} \in \mathbb{R}^{n\times k}$ that minimizes the distribution difference and maximize the inverse regression:

$$\max_{\mathbf{B}\in\mathbb{R}^{n\times k}} \frac{\frac{1}{n}\mathrm{tr}(\mathbf{B}^\top \mathbf{K}_y(\mathbf{K}_y + n\epsilon\mathbf{I}_n)^{-1}\mathbf{K}^2\mathbf{B})}{\mathrm{tr}(\mathbf{B}^\top\mathbf{KDKB} + \mathbf{B}^\top\mathbf{KB})} \tag{2.85}$$

Rewriting (2.85) as a constrained optimization results in Lagrangian

$$L = \frac{1}{n}\mathrm{tr}(\mathbf{B}^\top \mathbf{K}_y(\mathbf{K}_y + n\epsilon\mathbf{I}_n)^{-1}\mathbf{K}^2\mathbf{B}) - \mathrm{tr}((\mathbf{B}^\top\mathbf{KDKB} + \mathbf{B}^\top\mathbf{KB} - \mathbf{I}_k)\mathbf{\Lambda}).$$

Setting $\frac{\partial L}{\partial \mathbf{B}} = 0$, the following generalized eigenvalue problem is derived:

$$\frac{1}{n}\mathbf{B}^\top \mathbf{K}_y(\mathbf{K}_y + n\epsilon\mathbf{I}_n)^{-1}\mathbf{K}^2\mathbf{B} = (\mathbf{KDK} + \mathbf{K})\mathbf{B\Lambda},$$

where $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_k]$ are the $k$ leading eigenvectors with eigenvalues $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_k)$. At prediction time, a new sample $\mathbf{t} \in \mathbb{R}^d$ is mapped to a feature space by computing $\mathbf{z} = \mathbf{B}^\top\kappa(\cdot, \mathbf{t})$.

### 2.7.4 Low-Rank Exemplar-SVMs (LRE-SVMs)

Low-Rank Exemplar-SVMs (LRE-SVMs) [236] is a classifier built from exemplar-SVMs [139] that can be applied to domain generalization. Exemplar-SVMs are based on training a separate linear SVM classifier for every exemplar in the training set – an exemplar is defined as a set that consists of one positive instance and many negative instances. At prediction time, an ensemble of the trained exemplar-SVMs is used and surprisingly provides good generalization on the PASCAL VOC detection task. This method can be easily parallelized in a way that each exemplar-SVM is trained on a separate machine.

Consider a two-class classification problem in which we have $n$ positive instances $(\mathbf{x}_1^+, y_1^+), \dots, (\mathbf{x}_n^+, y_n^+)$ and $m$ negative instances $(\mathbf{x}_1^-, y_1^-), \dots, (\mathbf{x}_m^-, y_m^-) \in \mathbb{R}^d \times \mathbb{R}$ in the training set, where $y^+ = 1$ and $y^- = -1$. If a standard SVM is used to build a detector based on the fore-mentioned training set, it uses all the instances at once. Exemplar-SVMs cut up the learning problem into $n$ easy-to-solve subproblems with each subproblem containing a single positive instance. Suppose that $f_{\mathbf{w}_i} : \mathbb{R}^d \to \mathbb{R}$ is the $i$-th exemplar SVM parameterized by a weight $\mathbf{w}_i \in \mathbb{R}^d$ (the bias is ignored for notational convenience). Denote by $l : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ an SVM loss. We define the cost function of the $i$-th exemplar SVM as

$$J(\mathbf{w}_i) = \|\mathbf{w}_i\|_2^2 + c_1 l\left(f_{\mathbf{w}_i}(\mathbf{x}_i^+), y_i^+\right) + c_2 \sum_{j=1}^{m} l\left(f_{\mathbf{w}_i}(\mathbf{x}_j), y_j^-\right). \tag{2.86}$$

Xu et al. [236] utilize the logistic loss for $l : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ define by

$$l\left(f_{\mathbf{w}}(\mathbf{x}), y\right) = \log\left(1 + \exp(-y f_{\mathbf{w}}(\mathbf{x}))\right) = \log\left(1 + \exp(-y\mathbf{w}^\top \mathbf{x})\right). \tag{2.87}$$

Next we describe the formulation of LRE-SVMs as an extension of (2.86). LRE-SVMs attempt to discover $\mathbf{w}_i$ that provides a robust prediction $f_{\mathbf{w}_i}(\mathbf{x})$ regardless of the change of domains from which $\mathbf{x}$ is drawn. Suppose that $\mathbf{x}^+$ and $\mathbf{z}^+$ are the positive samples coming from different domains captured under similar conditions (e.g., frontal-view poses). Intuitive, $f_{\mathbf{w}_i}(\mathbf{x}^+)$ and $f_{\mathbf{w}_i}(\mathbf{z}^-)$, $\forall i = 1, \dots, n$ are expected to be similar to each other.

Formally, denote by $G(\mathbf{W}) = [f_{\mathbf{w}_i}(\mathbf{x}_j)]_{ij} \in \mathbb{R}^{n \times n}$ a likelihood matrix, where $\mathbf{W} = [\mathbf{w}_1, \dots, \mathbf{w}_n] \in \mathbb{R}^{d \times n}$. LRE-SVMs assume that a good domain generalization can be achieved by imposing $G(\mathbf{W})$ to be low-rank. In doing so, LRE-SVMs adds a regularization term into the cost function (2.86) such that it ends up with the following optimization problem:

$$\min_{\mathbf{w}_1,\ldots,\mathbf{w}_n} \sum_{i=1}^{n} J(\mathbf{w}_i) + \lambda \|G(\mathbf{W})\|_*, \tag{2.88}$$

where $\|\cdots\|_*$ is the nuclear norm to approximate the rank of $G(\mathbf{W})$. Solving the optimization (2.88), however, is nontrivial due to the use of the nuclear norm on $G(\mathbf{W})$ and the fact that $G(\mathbf{W})$ is a non-linear term w.r.t. $\mathbf{W}$. To resolve this issue, an intermediate matrix $\mathbf{F} \in \mathbb{R}^{n\times n}$ is introduced to model the true $G(\mathbf{W})$. That is, the term $\|G(\mathbf{W})\|_*$ in (2.88) is decomposed into two parts: i) imposing $\mathbf{W}$ to be low rank and ii) encouraging $G(\mathbf{W})$ to be close to $\mathbf{F}$. Hence, we relax the optimization (2.88) into the following:

$$\min_{\mathbf{w}_1,\ldots,\mathbf{w}_n} \sum_{i=1}^{n} J(\mathbf{w}_i) + \lambda_1 \|\mathbf{F}\|_* + \lambda_2 \|G(\mathbf{W}) - \mathbf{F}\|, \tag{2.89}$$

which can be solved by alternately updating $\mathbf{W}$ and $\mathbf{F}$, see Algorithm 1 in [236].

For prediction purposes, LRE-SVMs use a function $f_{\hat{\mathbf{W}}}(\cdot)$, which is an average of learned exemplar-SVMs. Given a test datapoint $\mathbf{t}$, the ensemble function $f_{\hat{\mathbf{W}}}(\cdot)$ is defined by

$$f_{\hat{\mathbf{W}}}(\mathbf{t}) = \frac{1}{n} \sum_{i=1}^{n} f_{\hat{\mathbf{w}}_i}(\mathbf{t}). \tag{2.90}$$

In practice, it is beneficial to fuse only a subset of $\{f_{\mathbf{w}_i}\}_{i=1}^{n}$ that provides the top $K$ prediction scores [236]. Let $\mathcal{T}(\mathbf{t})$ be the index set of the top $K$ exemplar-SVMs w.r.t. $\mathbf{t}$. The prediction on the test sample $\mathbf{t}$ can be obtained as

$$f_{\hat{\mathbf{W}}}(\mathbf{t}) = \frac{1}{K} \sum_{i \in \mathcal{T}(\mathbf{t})} f_{\hat{\mathbf{w}}_i}(\mathbf{t}). \tag{2.91}$$

Note that the LRE-SVM algorithm described above is only applicable to two-class classification problems (e.g., object detection). One can easily extend the algorithm in dealing with multi-class classification problems. To do so, we train $C$ sets of exemplar-SVMs, where $C$ indicates the number of classes. Hence, there will be $C$ learned ensemble functions $f_{\hat{\mathbf{W}}^{(k)}}(\cdot), \forall k = 1, \ldots, C$; each corresponds to the class. Given $\mathbf{t}$ as the test sample, the prediction can be done by simply taking the maximum over the ensemble predictions as follows:

$$\hat{y} = \max\left(f_{\hat{\mathbf{W}}^{(1)}}(\mathbf{t}), \ldots, f_{\hat{\mathbf{W}}^{(C)}}(\mathbf{t})\right). \tag{2.92}$$

## 2.8 Summary

Sections 2.1 - 2.5 present some basic concepts or theories relevant to this thesis. Meanwhile, Section 2.6 reviews existing work in the area of domain adaptation and domain generalization for object recognition.

Finally, it is worth emphasizing some gaps of existing domain adaptation and domain generalization work with respect to the goal and objectives of this thesis:

- Tang and Eliasmith [209] addresses a domain adaptation problem of recognizing noisy handwritten digits by a deep network, namely *sparsely-connected Deep Belief Nets* (sDBN), trained on clean digit examples. This thesis investigates the same problem addressed by sDBN but considers aspects that are not studied in [210] such as the effect of sparse feature regularization and the problem of learning on a small sized dataset.

- Since the emergence of the Office dataset [187], many domain adaptation algorithms in object recognition have been proposed [83,84,119,133]. To our knowledge, however, there have yet been studies involving neural networks that exist in literature at that time. This thesis establishes an early study of a neural network-based domain adaptation algorithm for object recognition.

- Domain generalization is a relatively new research area and still awaiting more powerful algorithms to bridge the performance gap to practically useful applications [25, 215]. This thesis develops an autoencoder-based representation learning algorithm that provides state-of-the-art domain generation performance in object recognition. The proposed algorithm is also the first study of utilizing autoencoders for domain generalization problems.

- Many state-of-the-art domain generalization algorithms deal with an optimization problem that is complex to solve [64,113,236]. Moreover, although domain generalization and domain adaptation are closely related settings, a domain generalization algorithm is typically not applicable to domain adaptation and vice versa. This thesis establishes a simple, fast algorithm that can be applied to both domain adaptation and domain generalization problems and provides competitive or better performance compared to state-of-the-art methods.

- The problem of domain adaptation remains essentially unsolved since the current methods are still far from mature for practical applications. Furthermore, many domain adaptation algorithms are optimized, for examples, by quadratic programming [34, 81] or eigen-analysis [67, 135, 166]. This thesis introduces a deep convolutional network-based domain adaptation model that is highly scalable and provides state-of-the-art object recognition performance.

# 3 Domain Adaptation on Noisy Objects with Deep Hybrid Networks

*This chapter presents the first contribution of this thesis. We develop a domain adaptation model by taking advantage of human knowledge as the auxiliary information. The proposed model, which we refer to as Deep Hybrid Network (DHN), deals with dataset bias or domain shift in handwritten digit recognition, where the actual environment contains noisy digits that are unseen during training.*

## 3.1 Introduction

A common strategy to develop an object recognition model is by learning a classifier from labeled image samples. One of the most effective models to do so is the Deep Neural Network (DNN) [97], which has become a breakthrough in handwritten digit recognition. In practice, the model may perform a recognition task in a noisy environment, but without observing the noise information during training. This is a specific example of *dataset bias* where a learning model may not generalize well. Therefore, it is necessary to develop a domain adaptation algorithm that learns from "clean" images only (source domain) and then generalizes on "noisy" objects (target domain).

### 3.1.1 Chapter Goals

The overall goal of this chapter is to develop a deep learning algorithm that can overcome dataset bias caused by noisy target objects. We refer to the proposed model as Deep Hybrid Network (DHN), which uses a particular combination of auto-encoder training and RBM training. An additional regularization is performed during auto-encoder training that induces sparse hidden layer activations/representations. We also propose an extension of DHN by setting local/sparse weight connections in the first hidden layer of the network, which we denote by *Sparsely-connected Deep Hybrid Network* (sDHN).

### 3.1.2 Chapter Organization

The remainder of this chapter is organized as follows. The proposed algorithms, DHN and sDHN, are described in Section 3.2. Sections 3.3 and 3.4 present the experiment results and discussions. Finally, Section 3.5 provides a summary of this chapter.

## 3.2 The Algorithms

This section starts with the notation and terminology including the formal description of the problem. It then describes the proposed algorithms to solve the problem.

### 3.2.1 Notation and Terminology

We introduce our notation and terminology used throughout this chapter. We use the same notation as defined in Section 2.4.3 to describe a deep neural network in terms of a set of visible and hidden layer vectors $\{\mathbf{v}, \mathbf{h}^{(1)}, ..., \mathbf{h}^{(L)}\}$, where each element in all vectors has a real value in the range of $[0, 1]$. Let $\mathbf{v}$ be a noiseless visible vector and $\tilde{\mathbf{v}}$ be a noisy visible vector obatined by adding some noise to $\mathbf{v}$, *i.e.*, $\tilde{\mathbf{v}} = \mathbf{v} + \Psi$, where $\Psi$ is an arbitrary unit-length noise vector. We define a notion of a representation noise invariance in terms of *$\varepsilon$-invariance*. Let us denote the $j$-th node activation of the first hidden layer given $\mathbf{v}$ by $h_j^{(1)}$ and the same node activation given $\tilde{\mathbf{v}}$ by $\tilde{h}_j^{(1)}$. The representation $\mathbf{h}^{(1)} \in \mathbb{R}^{n_{h^{(1)}}}$ is said to be *$\varepsilon$-invariant* if it satisfies the following condition

$$\begin{aligned} \forall j = 1, ..., n_{h^{(1)}}, \quad & d(h_j^{(1)}, \tilde{h}_j^{(1)}) \leq \eta\varepsilon \\ \text{subject to} \qquad & \|\Psi\| \leq \eta, \end{aligned} \tag{3.1}$$

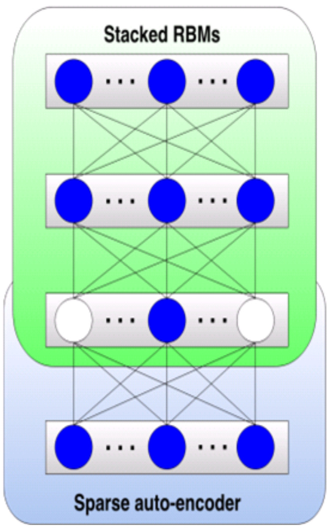
where $d(\cdot, \cdot)$ is any distance measure, $\eta$ is an upper bound for the noise magnitude, and $\varepsilon$ is a small error constant. We suggest that a deep neural network is likely to be robust to noise with a certain bound in the limit that $\eta \to 0$ if it has representations that satisfy (3.1) for a particular choice of $\varepsilon$.

In this context, the domain adaptation problem is simulated by specifying the source domain $\mathcal{D}_S$ as a distribution of clean images and the target domain $\mathcal{D}_T$ as a distribution of noisy images, *i.e.*, $\mathbf{v} \sim \mathcal{D}_S$ and $\tilde{\mathbf{v}} \sim \mathcal{D}_T$. Hence, we have a sufficient collection of labeled instances drawn from $\mathcal{D}_S$, but few or no labeled instances drawn from $\mathcal{D}_T$. We propose an algorithm that is robust to dealing with this problem.

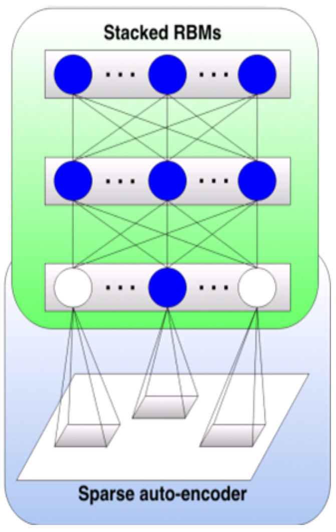
### 3.2.2 Deep Hybrid Networks

The main motivation of this chapter is to reduce the change between $h_j^{(1)}$ and $\tilde{h}_j^{(1)}$ as stated in (3.1). We introduce a deep network that we refer to as Deep Hybrid Network (DHN), which consists of a sparse autoencoder (SAE) on the bottom layer and stacked restricted Boltzmann machines (RBMs) on the top of the first hidden layer as the pretraining models, see Fig. 3.1(a).

The aim of combining SAE and DHNs is to achieve the following properties: 1) to preserve invariant representations over *global* noise achieved by sparsifying $\mathbf{h}^{(1)}$, and 2) to retain good generalization achieved by training stacked RBMs. It is known that stacked RBMs can provide good performance for *in-domain* recognition



(a) DHN



(b) sparsely-connected DHN (sDHN)

Figure 3.1: The architecture of Deep Hybrid Networks (DHN) and sparsely-connected DHN (sDHN). A sparse autoencoder is used as the pretraining model for the first layer, while restricted Boltzmann machines are used as the pretraining models for the higher layers. Each node in the first layer of sDHN is connected only with a local region in the input.

problem – in fact, they achieved a breakthrough in handwritten digit recognition [97]. Thus, we encourage the bottom RBM to observe the *denoised* feature $\mathbf{h}^{(1)}$ rather than to observe the raw data directly.

The sparse representation $\mathbf{h}^{(1)}$ is induced by the unsupervised autoencoder learning defined in Section 2.4.5 with an additional sparsity-inducing term. The autoencoder learning objective with a sparse regularization is given by

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{l=1}^{N} \frac{1}{2} \|\mathbf{v}^{(l)} - f(\mathbf{v}^{(l)}; \Theta)\| + \beta\phi(\mathbf{h}^{(1)(l)}) \tag{3.2}$$

where $\Theta$ is the set of all parameters and $N$ is the number of training examples.

We choose the sparsity-inducing function $\phi(\cdot)$ in the form of a generalized Kullback-Leibler (KL) divergence variant used in the *differentiable sparse coding*

[31] given by

$$KL(\mathbf{h}^{(1)} \| \boldsymbol{\gamma}) = \sum_{j=1}^{n_{h^{(1)}}} h_j^{(1)} \log \frac{h_j^{(1)}}{\gamma_j} - h_j^{(1)} + \gamma_j, \tag{3.3}$$

where $\boldsymbol{\gamma}$ is a uniform vector representing the sparsity target, *i.e.*, $\gamma_j = c, \forall j = [1, \ldots, n_{h^{(1)}}]$ with $c \in [0, 1]$ is a small real-valued constant. Here we always use the sigmoid function $h_j = \frac{1}{1+\exp(-z)}$ as the activation function so that the above KL divergence is a valid measure.

The generalized KL divergence in (3.3) has some beneficial properties. The first is that it is infinitely differentiable at any point, which leads to the possibility of using a simple gradient-based optimization algorithm to minimize (3.2). That is, it does not require any special step to impose the sparsity. Another property is that it approximates the $\ell_1$-norm of $\mathbf{h}^{(1)}$ if $c \to 0$, *i.e.*, $KL(\mathbf{h}^{(1)} \| \boldsymbol{\gamma}) \approx \|\mathbf{h}^{(1)}\|_1$. In contrast, it approximates the $\ell_2$-norm, $KL(\mathbf{h}^{(1)} \| \boldsymbol{\gamma}) \approx \|\mathbf{h}^{(1)}\|_2$ *i.e.*, if $c \to \infty$. Finally, the generalized KL divergence does not require us to assume that $\|\mathbf{h}^{(1)}\|_1 = 1$ and $\|\gamma\|_1 = 1$ so that it is convenient in the sense of eliminating the normalization process.

Note that encouraging sparse features induces a *filter* or *dictionary*, *i.e.*, the set of learned weights, that is visually interpretable associated with the visual area V1. In this case, the learned weights appear to be localized, oriented, and edge detector-like [163]. Coates and Ng [45] suggests that this property may retain good generalization even when the model is trained from a small set of labeled images. We confirm that DHN is also equipped with such a property in the experiment section.

As is typical in deep neural networks, the DHN training has two stages: 1) *pre-training* and 2) *fine-tuning*. In the *pre-training* stage, we train the bottom layer using autoencoder learning with respect to the objective (3.2). Backpropagation with L-BFGS optimization [160] is used to optimize the autoencoder. We then train the stacked RBMs above the bottom layer using the contrastive divergence, see Section 2.4.4, by taking the vector $\mathbf{h}^{(1)}$ as the first input. Here the input vector $h_j^{(1)}$ is a binary-valued vector sampled from $P(h_j^{(1)} = 1|\mathbf{v})$. The complete summary of DHN pre-training is described in Algorithm 3.

After the unsupervised pre-training is completed, DHN is fine-tuned using the labeled source images. Backpropagation with Conjugate Gradient [95] is employed for the fine-tuning.

**Algorithm 3** The Deep Hybrid Network Pre-training Algorithm

```
Input:;
• $L$ is the number of layers in the deep hybrid network.
• $\mathbf{W}^{(i)}$ is the weight matrix for level $i$, $\forall i = 1, ..., L$.
• $\mathbf{b}^{(i)}$ is the bias vector for level $i$, $\forall i = 0, ..., L$.
• $\mathbf{v}$ is the input data vector.
• $\mathbf{h}^{(i)}$ is the hidden nodes vector for layer $i$, $\forall i = 0, ..., L$, $\mathbf{h}^{(0)}$: observed data
1: Initialize $\mathbf{W}^{(1)}, ..., \mathbf{W}^{(L)}$ with random values;
2: $\mathbf{b}^{(0)}, ..., \mathbf{b}^{(L)} \longleftarrow \mathbf{0}$;
3: $\mathbf{h}^{(0)} \longleftarrow \mathbf{v}$;
4: Update $\mathbf{W}^{(1)}$, $\mathbf{b}^{(0)}$ and $\mathbf{b}^{(1)}$ using sparse autoencoder learning algorithm
   with respect to the objective (3.2);
5: Assign $P(\mathbf{h}^{(1)}|\mathbf{v}) \longleftarrow \sigma({\mathbf{W}^{(1)}}^\top \mathbf{v} + \mathbf{b}^{(1)})$;
6: for $l = 2$ to $L$ do
7:     Update $\mathbf{W}^{(l)}$, $\mathbf{b}^{(l-1)}$, and $\mathbf{b}^{(l)}$ using RBM learning algorithm;
8:     Sample $h_j^{(l)} \sim P(h_j = 1|\mathbf{h}^{(l-1)})$, $\forall j = 1, ..., n_{h^{(l)}}$;
9: end for
```

It is natural to ask the reason for using the autoencoder as the model in the first layer, since sparse features can be also encouraged by using the RBM-based method, *e.g.*, sparse RBM [128]. In this work, the only reason is for the sake of simplicity of obtaining the optimal hyper-parameters. Note that we employ back-propagation with L-BFGS optimization as the autoencoder learning, which does not require the learning rate. We found that getting good hyper-parameters associated with a visually interpretable dictionary is relatively simple and works for small sized datasets by doing so – the only hyper-parameter we care about is the sparsity parameter $c$. It is not the case when using sparse RBM – we need to tune many hyper-parameters such as learning rate, sparsity penalty, and other complementary strategies (*e.g.*, momentum and batch size) for the RBM training. In addition, we failed to obtain an interpretable dictionary of sparse RBM when learning from only 1,000 training images after an extensive hyper-parameter search.

### 3.2.3 Sparsely-connected Deep Hybrid Network

Although sparse features in DHNs are expected to be at best $\varepsilon$-invariant, the fully-connected weights specified in the autoencoder may still cause the features $\mathbf{h}^{(1)}$ to be sensitive to a locally occluded image. As suggested by Tang and Eliasmith [209], sparse weights between $\mathbf{v}$ and $\mathbf{h}^{(1)}$ can reduce the effect of occlusions on local regions. To incorporate both properties of sparse features and sparse weights, we propose a variant of the DHN that we refer to as the sparsely-connected Deep Hybrid Network (sDHN).

Let us denote a *binary mask* by $\mathbf{M} \in \{0,1\}^{n_v \times n_{h^{(1)}}}$. Each column of $\mathbf{M}$ represents a connection between each node in $\mathbf{h}^{(1)}$ to an $s \times s$ sub-image in $\mathbf{v}$, where $\mathbf{M}_{ij} = 1$ denotes that a connection between $v_i$ and $h_j^{(1)}$ exists. The choice of the $s \times s$ sub-image for each node $h_j^{(1)}$ is random, but constrained such that every node in $\mathbf{v}$ is guaranteed to have a particular connection with a node in $\mathbf{h}^{(1)}$. We then compute the sparse weights $\mathbf{W}_M^{(1)}$ by a *masking* operation

$$\mathbf{W}_M^{(1)} = \mathbf{W}^{(1)} \odot \mathbf{M}, \tag{3.4}$$

where $\odot$ is the element-wise multiplication. A possible configuration induced by (3.4) is best described in Fig. 3.1(b).

Tang and Eliasmith [209] also used the sparse weight connection in the bottom layer of a deep network referred to as sparse Deep Belief Network (sDBN). However, sDBN does not induce sparse representations of $\mathbf{h}^{(1)}$. If $\mathbf{W}_M^{(1)}$ is visualized, the difference between $\mathbf{W}_M^{(1)}$ learned by sDBN and sDHN can be seen in Fig. 3.3(c)-3.3(d). The results will be discussed in detail in Section 3.3.4.

Note that we do not employ a special technique referred to as *probabilistic denoising* used in sDBN [209] at test time. This technique may further increase the robustness of the model, but it will also increase the test time complexity, which is less desirable in practice. In this contribution, we only focus on the network architecture and sparsity with respect to the robustness to noise, while keeping the time complexity low.

## 3.3 Experiment 1: "Clean" vs "Noisy" Object Recognition

We evaluated the robustness of our proposed algorithms (DHN and sDHN) on recognizing handwritten digits under the following domain adaptation setting: the source domain contains *clean* images and the target domain contains *noisy* images. A range of noise types were investigated in this experiment, see the subsection below.

### 3.3.1 Data Preparation

We used the MNIST dataset [125] that consists of 10-class handwritten digit images of size $28 \times 28$, see Table 1.1 for a detailed configuration. We normalized the image pixels into $[0, 1]$ values. From this dataset, we created several new test sets by cluttering the original MNIST test set with noise. The types of noise used in this experiment are *2-pixel border*, block occlusion with random position and size (*block*) in each sample, impulse noise with ratio of 40% (*impulse-04*), and background image taken randomly from three natural scene patches (*back-im*). Figure 3.2 depicts the digit samples with respect to the fore-mentioned types of noise.

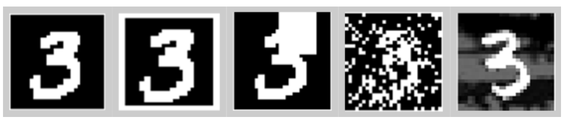


Figure 3.2: The MNIST digit images with various types of noise: *clean*, *2-pixel border*, *block*, *impulse-04*, *back-im*

### 3.3.2 Baselines

We compared the recognition performance of DHN and sDHN with three common deep networks: (i) Deep Belief Network (DBN) [97], (ii) sparsely-connected Deep Belief Network (sDBN) [209], and (iii) sparse auto-encoder (SAE). SAE uses the same sparse regularization strategy as our algorithms, i.e., that of the unnormalized

KL divergence (3.3). Note that SAE imposes the sparse feature regularization in all layers during pre-training, which is different from our algorithms. DHN and sDHN apply the sparse feature regularization only in the bottom layer.

### 3.3.3 Training Setup

Each network has the same architecture with three hidden layers of size 500, 500, 2000 nodes and a target layer with 10 nodes, following the network of Hinton and Osindero [97]. For the sparsely-connected networks, The binary mask $M$ were specified such that the connection between each node in $\mathbf{h}^{(1)}$ and the visible layer was of size $7 \times 7$.

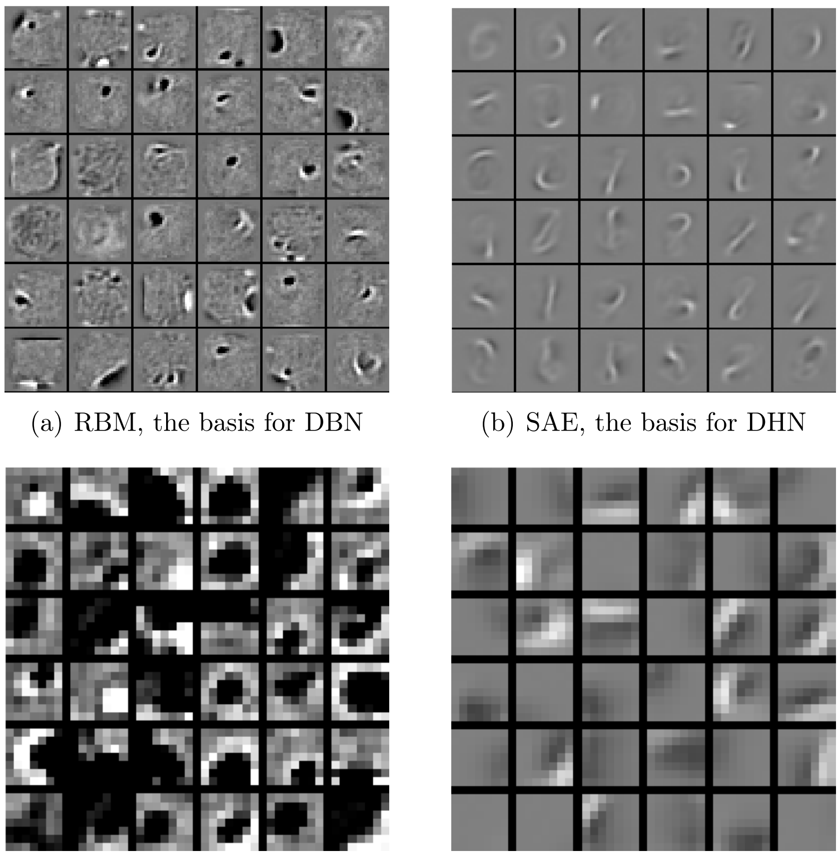
All networks were trained by the unsupervised greedy layer-wise pre-training and followed by the supervised back-propagation fine-tuning [97]. In every RBM training, we ran 50 epochs of mini-batched learning with learning rate $\alpha = 0.1$, weight decay $\lambda = 10^{-5}$, and 25-CD.[1] In the auto-encoder training, we used L-BFGS back-propagation as the optimizer with sparsity target $\gamma = 0.1$ and regularization constant $\beta = 3$ [122]. Each network was fine-tuned by Conjugate Gradient back-propagation [188]. We ran ten independent experiments for each model to obtain the statistical significance result.

### 3.3.4 Effect of Sparsity

It is useful to study the behavior of the first layer bases after pre-training using the MNIST training set. We visualize the basis vectors (the columns of $\mathbf{W}^{(1)}$) as shown in Fig. 3.3. We can see that the sparse regularized models (DHN and sDHN) have more interpretable weights, which look like digit brush-stroke patterns. This suggest that the feature sparsification somehow strengthens the salient information of digits and, in the same time, "cleans up" the less correlated information.

In sDBN and sDHN, the weights visually look like the subregions of those in their fully-connected counterparts (DBN and DHN). One can interpret this as that the weights $\mathbf{W}^{(1)}$ acts as a *local filter* to map an image patch to a real value – what happened to an image patch only affects a bit of information in the first hidden

[1]For a fair comparison, we used the same number of $k$ for the $k$-CD specified in [209].


(a) RBM, the basis for DBN (b) SAE, the basis for DHN

(c) RBM, the basis for sDBN (d) SAE, the basis for sDHN

Figure 3.3: The visualization of the first 36 basis vectors of $\mathbf{W}^{(1)}$ after training using 60,000 MNIST images.

layer. For instance, if a $7 \times 7$ patch is contaminated by noise, it will disturb only one node in $\mathbf{h}^{(1)}$. Furthermore, sDHN is also expected to obtain more invariant node $h_j^{(1)}$ associated with a particular patch than sDBN, since sDHN produces an interpretable dictionary.

On the other hand, sDBN and sDHN might lose capacity in terms of the number of distinct samples that can be captured. It is based on the claim that the information capacity is proportional to the number of connections in the network [68]. As the connections are more sparse, sDBN and sDHN may fail to store a large number of distinct samples as well as the fully connected networks. In general, it can be expected that the best robustness that a network can achieve decreases with the utilization of its capacity. Here we did not further investigate the model robustness in terms of the capacity, since this is beyond the scope of this contribution chapter.

### 3.3.5 Results on *In-domain* Recognition

We compared the in-domain recognition performance of each network on the MNIST data set. The *in-domain* recognition refers to the standard task in which the original MNIST test set is used. Table 3.1 summarizes the *in-domain* performance accuracy in terms of the accuracy rate (%). We can see that the DBN has the best performance in the case of learning by using all the training examples (60,000 images), while the DHN and sDHN perform slightly worse than the DBN. Furthermore, the SAE, which has sparse features in every layer, performs worst. This suggests that neither sparse weights nor sparse features do improve the performance of recognizing the *in-domain* test digits when sufficient training examples are available.

In the case of using a small training set, i.e., only 1,000 images (100 for each class) taken from the MNIST training set are used, the sparse features increases the recognition performance as the SAE, DHN, and sDHN perform better than the DBN. This is consistent with the suggestion that the model with an interpretable dictionary may obtain better generalization when only a few labeled training examples are used [45]. In addition, our proposed models (DHN and sDHN) perform slightly better than the SAE. This may be due to the property of stacked RBMs used in both DHN and sDHN in terms of retaining good model generalization, as we have expected in Section 3.2.

| Data set (#training) | DBN | sDBN | SAE | DHN | sDHN |
|---|---|---|---|---|---|
| MNIST (60,000) | **99.04 ± 0.01** | 98.89 ± 0.01 | 98.60 ± 0.07 | 98.87 ± 0.09 | 98.77 ± 0.01 |
| MNIST (1,000) | 89.06 ± 0.95 | 90.78 ± 0.34 | 91.35 ± 0.21 | **91.81 ± 0.37** | **92.04 ± 0.00** |

Table 3.1: Baseline accuracy rates (%); each deep network was trained and tested using the full set of MNIST. The best score (a few have overlapping confidence intervals) for each setting are written in bold.

### 3.3.6 Results on *Cross-domain* Recognition

We evaluated the cross-domain recognition performance of the algorithms, which is the main results of this chapter. The algorithms were trained from the original MNIST training set and tested on the artificial corrupted MNIST test sets. The types of noise used to contaminate the original test set have been described in Section 3.3.1. We investigate two settings with respect to the number of training examples used: (i) using the complete 60,000 images, and (ii) using only randomly selected 1,000 images. All accuracy rates are shown in Tables 3.2 and 3.3.

It is clear that DBN is unable to retain good recognition performance for any case with noisy input. The sDBN performs best in the case of *block* noise but is less robust when dealing with non-local noise (*2-pixel border*, *impulse-04*, and *back-im* noise). In contrast, the DHN and sDHN can cope with non-local noise better than the sDBN, as shown in the results of recognizing digits with impulse and background noise. Both also performs reasonably well when training from the small set with only 1,000 images.

In the recognition of digits with block occlusion, one unexpected result is that our sDHN does not perform, at least, as well as the sDBN in the recognition of digits with block occlusion although it still performs better than DHN. This result slightly contradicts our expectation that sDHN may incorporate both properties of sparse features and sparse weights. It means that our sDHN should have the best recognition performance on digits with both local and non-local noise. However, the fact is that it only slightly improves on DHN. Thus, sparse features may not be helpful for sparsely-connected networks when dealing with images with block occlusion. This performance is likely the result that sparsifying the bottom hidden layer of sparsely-connected networks may remove some salient features needed to recognize block-noisy images.

## 3.4 Experiment 2: Cross-domain digit recognition (MNIST vs USPS)

For completeness, it is useful to investigate the robustness of DHN and sDHN against a broader notion of noise, that is, the noise may be that of object shape,

| Noise | DBN | sDBN | SAE | DHN | sDHN |
|---|---|---|---|---|---|
| *2-pixel border* | 10.64 ± 0.02 | 97.45 ± 0.02 | 96.25 ± 0.30 | **97.83 ± 0.05** | **98.30 ± 0.02** |
| *block* | 50.17 ± 0.07 | **73.87 ± 0.20** | 58.17 ± 0.38 | 58.33 ± 0.02 | **62.20 ± 0.09** |
| *impulse-04* | 11.70 ± 0.02 | 38.08 ± 0.53 | **62.41 ± 1.12** | **67.93 ± 0.41** | **63.78 ± 0.19** |
| *back-im* | 51.23 ± 0.07 | 76.22 ± 0.20 | **92.14 ± 0.50** | **94.62 ± 0.20** | **92.54 ± 0.05** |

Table 3.2: Accuracy rates (%) on the MNIST test set with various types of noise. Each deep network was trained using 60,000 clean MNIST training examples. Bold-red and bold-black indicate the best and second best performance.

| Noise | DBN | sDBN | SAE | DHN | sDHN |
|---|---|---|---|---|---|
| *2-pixel border* | 10.49 ± 1.36 | 88.42 ± 0.08 | 90.58 ± 0.16 | **91.43 ± 0.42** | **91.87 ± 0.00** |
| *block* | 32.96 ± 4.34 | **71.60 ± 0.80** | 56.08 ± 0.50 | 60.64 ± 0.30 | **61.81 ± 0.00** |
| *impulse-04* | 12.74 ± 4.22 | 43.04 ± 4.72 | **64.79 ± 0.67** | **73.37 ± 1.02** | **71.82 ± 0.56** |
| *back-im* | 36.41 ± 8.91 | 68.62 ± 3.03 | **85.65 ± 0.22** | **88.94 ± 0.90** | **88.37 ± 0.00** |

Table 3.3: Accuracy rates (%) on the MNIST test set with various types of noise. Each deep network was trained using 1,000 clean MNIST training examples.

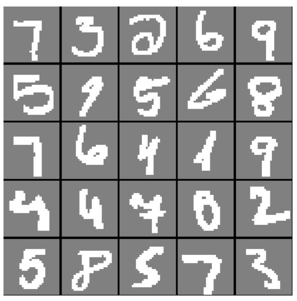
size, or style. This can be simulated by evaluating the algorithms on the target domain represented by a different dataset from one used as the source domain. In particular, we performed the handwritten digit recognition task across two different datasets: the MNIST and the USPS [106].

The USPS consists of 7,291 training images and 2,007 test images of size $16 \times 16$. Compared to the MNIST, the USPS can be considered as a small data set and has somewhat different variation in the shape, size, and stroke style of digit, see Fig. 3.4 and 3.5 for a comparison. We compared the performance of four deep neural networks including our proposed models: DBN, sDBN, sDHN, and DHN, see Section 3.3 for a detailed explanation of each network.

## 3.4.1 Evaluation Setting

The setting of this experiment is taking the MNIST images as the training examples and the USPS images as the test examples or vice versa. Let us denote MNIST by m and USPS by u. We can thus have two possible cross-domain cases: $\mathrm{m} \rightarrow \mathrm{u}$ and $\mathrm{u} \rightarrow \mathrm{m}$.

We evaluate the algorithms under two different domain adaptation protocols


(a) MNIST training

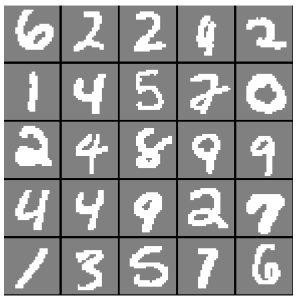
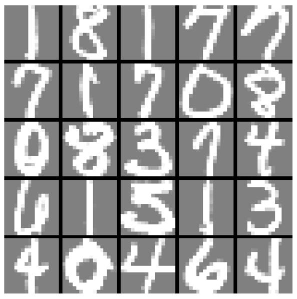

(b) MNIST test

Figure 3.4: Examples of MNIST digit images.


(a) USPS training

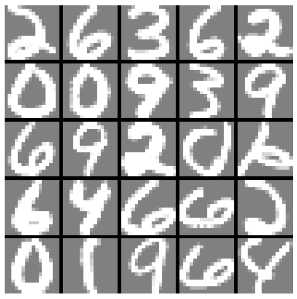

(b) USPS test

Figure 3.5: Examples of USPS digit images.

in terms of the label incorporation during training.

1. *Using training source labels (S).*
   This protocol only allows to use labels from the source domain as the training information. This means that our algorithms conduct the full training using only the labeled source data.

2. *Using both training source and target labels (ST).*
   This protocol allows to use labels from the target domain in addition to those from the source domain. In particular, our algorithms use labeled instances from both source and target training sets as the training examples.

### 3.4.2 Training Setup

We used the same training setup as described in Section 3.3.3 in terms of the network architecture, the unsupervised *pre-training*, and most of the learning parameters. The only difference is on the sparse regularization hyper-parameter for training the sDBN, DHN, and sDHN from the USPS data set, i.e., $\gamma = 0.15$, $\beta = 1.0$.

For the supervised discriminative learning, however, we used the *ordinary supervision* instead of the *fine-tuning*, *i.e.*, the node activation vectors in the last hidden layer are used as inputs to a classifier. The classifier used for the ordinary supervision was the linear SVM. LIBLINEAR [63], a C/C++ SVM library that can be executed through MATLAB, is used to run the linear SVM.

### 3.4.3 Results and Discussion

The complete evaluation results in terms of performance accuracy (%) are summarized in Table 3.4. We report the performance across a range of network depth, from one to three hidden layers. The *in-domain* recognition performances are also included as baselines. The bar plot representation of the results is also provided in Figure 3.6 for completeness and ease of observing the trend.

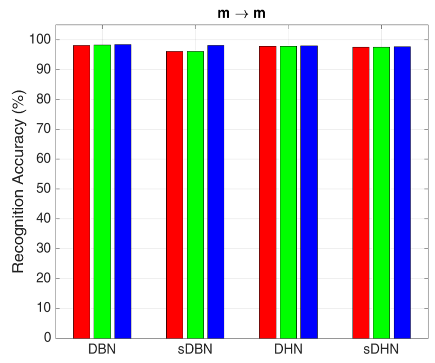
In general, the results tend to be consistent with the results from the previous experiment, *i.e.*, the DBN has the best *in-domain* recognition performance (m $\rightarrow$ m and u $\rightarrow$ u), while either DHN or sDHN has the best *cross-domain* recognition performance. This indicates that the DHN and sDHN are effective not only for recognizing images with noise of the form artificial corruption (as shown in the previous section), but also for recognizing images with different shape, size, and stroke style.

Note that there is an intriguing behavior regarding to the network depth: adding the depth of the models does not always end up with a superior cross-domain performance. Depth is only important when the task is the *in-domain* object recognition and the training set is large enough. Specifically, deeper networks always provide better performance only on the *in-domain* MNIST recognition tasks. This outcome could be attributed to the trade-off between the model complexity and the amount of training data.

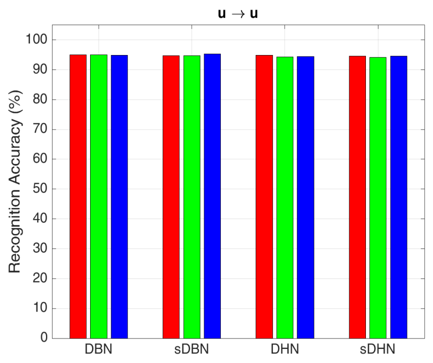
Finally, we observe that the mixture of sparsity, *i.e.*, the combination between sparse features and sparse weight connections, plays an important role to provide better *cross-domain* recognition performance, unlike the case in the first experiment. This is indicated by the best performance produced by sDHN in both m $\rightarrow$ u and u $\rightarrow$ m cases, while sDBN and SHN have comparable performance. This suggests that the combination of sparse features and sparse weight connections in the bottom layer is an effective strategy for *cross-domain* handwritten recognition tasks with respect to the shape, size, and stroke style difference.

Table 3.4: Performance accuracy (%) on MNIST ($m$) vs USPS ($u$) recognition tasks provided by DBN, sDBN, DHN, and sDHN according to a linear SVM classifier. $m \to m$ and $u \to u$ refer to the *in-domain* recognition cases. The remaining cases are the *cross-domain* recognition tasks. Bold-red and bold-black indicate the best and second best performance over the methods and HL stands for "Hidden Layer".

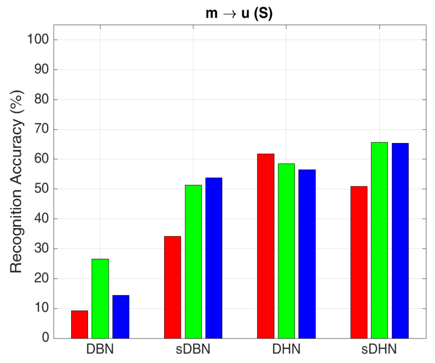
| Methods | HL | $m \to m$ | $u \to u$ | $m \to u$ (S) | $m \to u$ (ST) | $u \to m$ (S) | $u \to m$ (ST) |
|---|---|---|---|---|---|---|---|
| | 1 | 98.18 ± 0.16 | **95.02 ± 0.25** | 9.28 ± 0.23 | 42.35 ± 0.27 | 37.96 ± 0.16 | 55.59 ± 0.12 |
| DBN | 2 | 98.24 ± 0.13 | 94.97 ± 0.22 | 26.36 ± 0.28 | 51.61 ± 0.24 | 31.14 ± 0.13 | 69.23 ± 0.15 |
| | 3 | **98.50 ± 0.12** | 94.82 ± 0.23 | 14.35 ± 0.25 | 49.83 ± 0.22 | **48.89 ± 0.12** | 72.25 ± 0.10 |
| | 1 | 96.11 ± 0.14 | 94.77 ± 0.22 | 34.18 ± 0.23 | 70.85 ± 0.24 | 43.82 ± 0.14 | 67.91 ± 0.08 |
| sDBN | 2 | 96.11 ± 0.14 | 94.77 ± 0.22 | 51.37 ± 0.27 | 61.88 ± 0.18 | 39.00 ± 0.15 | 73.93 ± 0.09 |
| | 3 | **98.19 ± 0.15** | **95.31 ± 0.20** | 53.81 ± 0.22 | **75.14 ± 0.19** | 47.67 ± 0.13 | 79.70 ± 0.05 |
| | 1 | 97.81 ± 0.12 | 94.82 ± 0.23 | **61.84 ± 0.18** | 68.61 ± 0.21 | 42.42 ± 0.11 | 60.97 ± 0.07 |
| **DHN** | 2 | 97.93 ± 0.13 | 94.27 ± 0.21 | 58.54 ± 0.20 | 65.47 ± 0.23 | 39.63 ± 0.14 | 68.07 ± 0.05 |
| | 3 | 98.00 ± 0.11 | 94.47 ± 0.19 | 56.45 ± 0.24 | 74.08 ± 0.17 | 47.47 ± 0.14 | **84.12 ± 0.10** |
| | 1 | 97.57 ± 0.11 | 94.62 ± 0.20 | 50.92 ± 0.21 | 71.75 ± 0.25 | 49.42 ± 0.10 | 72.40 ± 0.09 |
| **sDHN** | 2 | 97.57 ± 0.11 | 94.12 ± 0.22 | **65.62 ± 0.24** | 72.05 ± 0.19 | 50.78 ± 0.11 | 74.49 ± 0.11 |
| | 3 | 97.79 ± 0.12 | 94.62 ± 0.20 | 65.37 ± 0.25 | **78.20 ± 0.17** | **54.64 ± 0.09** | **80.43 ± 0.08** |



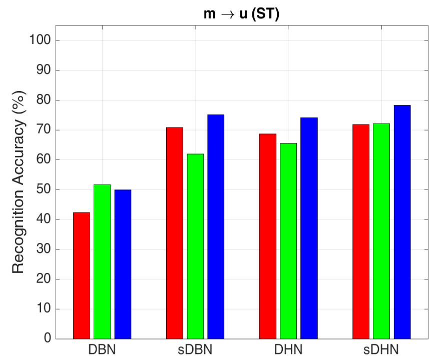
(a) $m \rightarrow m$



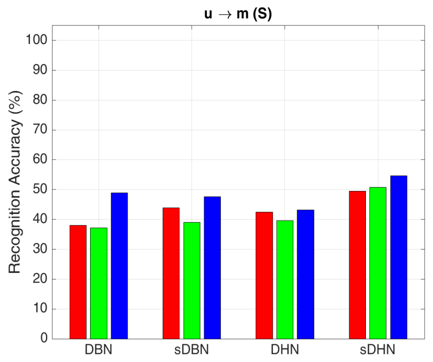
(b) $u \rightarrow u$



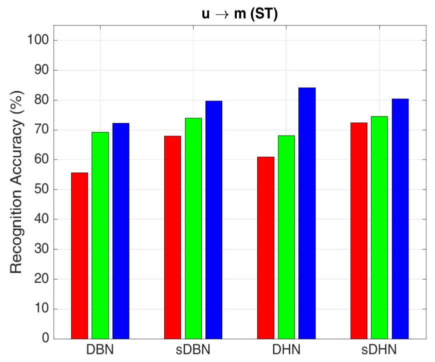
(c) $m \rightarrow u$ (S)



(d) $m \rightarrow u$ (ST)



(e) $u \rightarrow m$ (S)



(f) $u \rightarrow m$ (ST)

Figure 3.6: Performance accuracy (%) of MNIST vs USPS cross-recognition tasks in bar plots, see Table 3.4. Each bar color indicates the level of hidden layers in a deep network (1: red, 2: green, 3: blue).

## 3.5 Chapter Summary

In this chapter, the goal was to develop a domain adaptation algorithm based on deep neural networks that can recognize visual objects under dataset bias caused by noise. Specifically, the algorithm performs the following cross-domain recognition task: learning from *clean* source objects and recognizing *noisy* target objects over a range of noise types such as impulse noise, square-block occlusion, pixel border addition, and background image noise. Our algorithm, which we refer to as Deep Hybrid Network (DHN), is based on a particular combination of an autoencoder (AE) and restricted Boltzmann machines (RBMs). The autoencoder is equipped with a sparse feature regularization, which plays a role to undo the *global* noise occurring in the input images.

We also propose a variant of DHN, which imposes sparse, locally connected weights in the bottom layer in addition to the sparse feature regularization. The model is referred to as sparsely-connected Deep Hybrid Network (sDHN). This variant is expected to extend the ability of DHN in dealing with *local noise*, *i.e.*, block occlusion occurred in the local regions of the images. From experiments, the general outcome is that our algorithms provide better cross-domain handwritten digit recognition performance than the standard deep learning models, while retain good in-domain performance.

This chapter shows that DHN provides better cross-domain performance than the standard deep learning model, *i.e.*, Deep Belief Network (DBN). This means that the autoencoder learning with sparse regularization in the bottom layer can indeed reduce the impact of noise occurred in the target domain. Furthermore, DHN is still able to retain a good in-domain performance, comparable to DBN. In fact, the performance of DHN is even better than that of DBN in the case of learning from a small training set size.

More specifically, this chapter shows that DHN performs best in recognizing objects with impulse noise and background image noise, which are considered as *global noise*. This performance is consistent in both large ($60,000$ images) and small ($1,000$ images) MNIST training set cases. However, DHN still underperforms sDBN [209] in the case of *local noise*: square-block and 2-pixel border.

This chapter also shows that a variant of DHN, *i.e.*, sparsely-connected DHN (sDHN), can compromise between the global and local noise. In particular, sDHN provides lower error rate than DHN on square-block noise and 2-pixel border noise cases, while still retain a competitive performance on impulse noise and background image noise. sDHN does not outperform sDBN in the case of square-block noise, which suggests that the mixture of sparse features and sparse weight connection is not helpful for this case – the use of sparse weight connection only is sufficient.

sDHN provides the best performance on cross-domain MNIST vs USPS tasks, *i.e.*, taking MNIST images as the training examples and USPS images as the test examples or vice versa. These tasks simulate the *clean-noisy* object recognition with more general notion of noise: the noise is in the form of object shape, size, or style. In this case, the mixture of sparse features and sparse weight connections is indeed helpful to provide good generalization – DHN and sDBN do not provide competitive performance.

Another interesting outcome in this chapter is that the sparse autoencoder (SAE), the model that encourages sparse features in every layer during the greedy layer-wise autoencoder training, does not perform better than DHN on either in-domain or cross-domain recognition task. Recall that DHN involves only a single sparse feature regularization in the bottom layer. This indicates that the combination of (sparse) autoencoder and (non-sparse) RBM pretraining is beneficial to provide better generalization than using only a single learning algorithm to train deep networks.

The domain adaptation solution proposed in this chapter is an example of taking advantage of human knowledge as the auxiliary information to develop robust-to-noise algorithms. In this context, the knowledge is in the form of *noise* occurred in the target domain. The design strategy of the algorithm is thus established such that a *denoising* process should be performed during test time. However, this approach may not be suitable in a situation where human experts do not have sufficient knowledge about the target domain. This issue could be overcome by utilizing *unlabeled* target examples as the auxiliary information. The next chapter will develop a neural network-based approach that takes advantage of unlabeled target examples during training, which reduces the distribution discrepancy in the latent space and also minimizes the classification loss.

# 4 Domain Adaptive Neural Networks for Object Recognition

*This chapter presents the second contribution of this thesis. We propose a domain adaptation model based on neural networks that leverages unlabeled target samples as the auxiliary information. The model uses a distribution distance measure, Maximum Mean Discrepancy (MMD), to regularize the neural network training such that the representation distribution difference in the hidden layer is minimized. We refer to this model as Domain Adaptive Neural Network (DaNN).*

## 4.1 Introduction

In domain adaptation, domains are usually equated with probability distributions. A general domain adaptation algorithm is thus designed such that the training-test data distribution difference is minimized. There exist several strategies to do so; one of them is to extract data representations that satisfy this property.

In the context of representation learning, training neural networks is equivalent to finding discriminative hidden layer activations, that is, representations, of the observed data. However, a neural net trained on a particular domain may suffer from dataset bias if it is applied to another different but related domain. For example in image recognition, the hidden layer activations of a particular object from the target domain may be different from those of another object of the same category from the source domain. Therefore, it is necessary to establish an explicit setup such that the representation distribution difference between the source and target domain is minimized during training.

### 4.1.1 Chapter Goals

The goal of this chapter is to develop a simple variant of neural nets with good domain adaptation performance to object recognition, which we refer to as a Domain Adaptive Neural Network (DaNN). The model ensures that the hidden node distribution is similar for source data and target data, while learning to predict labels from the source data. Such a distribution matching is obtained by utilizing a distribution distance measure, *i.e.*, Maximum Mean Discrepancy (MMD) [28], as a regularization embedded in the supervised backpropagation training.

MMD has been used in many existing domain adaptation algorithms [60, 82, 134, 165]. Despite its effectiveness, to the best of our knowledge, the use of MMD in the context of neural networks has not been investigated. This work is therefore the first study of the use of MMD in neural networks. Specifically, this chapter will investigate:

- Whether the MMD regularization can indeed improve the object recognition performance of neural networks under two domain adaptation settings: *unsupervised domain adaptation* and *semi-supervised domain adaptation*;

- Whether DaNN can perform well on both raw image pixels and SURF-based extracted features as inputs;

- Whether the use of denoising autoencoder (DAE) pretraining helps to improve the domain adaptation performance of DaNN on raw pixels.

### 4.1.2 Chapter Organization

The remainder of this chapter is organized as follows. The proposed model, DaNN, is described in Section 4.2. The experiment results are discussed and analyzed in Section 4.3. Section 4.4 summarizes this chapter.

## 4.2 The Algorithm

This section describes the proposed neural net that incorporates Maximum Mean Discrepancy (MMD) as a regularization embedded in the supervised training. Before describing the main algorithm, we introduce the notation and terminology.

### 4.2.1 Notation and Terminology

We denote the data space by $\mathcal{X} \subseteq \mathbb{R}^d$ and the label space by $\mathcal{Y} \subseteq \{0,1\}^c$. The label space is constrained such that it consists of all 0s except for a 1 at a particular element. A domain is represented by a probability distribution on $\mathcal{X}$. Let $\mathbb{P}^s$ and $\mathbb{P}^t$ be the source distribution and target distribution respectively, where $\mathbb{P}^s \neq \mathbb{P}^t$. We denote by $\hat{\mathbb{P}}^s$ and $\hat{\mathbb{P}}^t$ the empirical source and target distributions.

### 4.2.2 Domain Adaptive Neural Networks

The proposed model, DaNN, is basically a standard feedforward neural network (NN) trained with a regularized backpropagation by means of Maximum Mean Discrepancy (MMD), see the detailed explanation of NN in Section 2.4.2. Specifically, MMD is used to control the distribution difference between the source and target data hidden layer activations during training. Such a regularization encourages the hidden layer activations to be invariant across different domains.

Let us denote the labeled source samples by $\{\mathbf{x}_i^s, \mathbf{y}_i^s\}_{i=1}^{n_s} \sim \mathbb{P}^s$ and the unlabeled target samples by $\{\mathbf{x}_i^t\}_{i=1}^{n_t} \sim \mathbb{P}_X^t$. Without losing generality, consider a single hidden layer neural net $f_{\mathbf{\Theta}} : \mathcal{X} \to \mathcal{Y}$, where $\mathbf{\Theta} = \{\mathbf{W}, \mathbf{W}^{\text{out}}\}$ is a set of weights and biases – the construction of matrices $\mathbf{W}, \mathbf{W}^{\text{out}}$ can be seen in (2.19). Since our concern is the multi-class classification, we can use the *cross-entropy* (2.22) as the loss function for the neural net, $\ell(\mathbf{y}, f_{\mathbf{\Theta}}(\mathbf{x}))$. The objective function of DaNNs is given by

$$J_{\text{DaNN}}(\mathbf{\Theta}) = \sum_{i=1}^{n_s} \ell(\mathbf{y}_i^s, f_{\mathbf{\Theta}}(\mathbf{x}_i^s)) + \gamma \text{MMD}_{\mathcal{F}}^2[\hat{\mathbb{P}}^s, \hat{\mathbb{P}}^t], \tag{4.1}$$

where $\text{MMD}_{\mathcal{F}}^2[\hat{\mathbb{P}}^s, \hat{\mathbb{P}}^t]$ is the squared empirical MMD and $\gamma$ is a constant controlling the contribution of MMD. Training a DaNN corresponds to finding a set of parameters $\hat{\mathbf{\Theta}}$ such that $J_{\text{DaNN}}(\mathbf{\Theta})$ is minimized in which the backpropagation algorithm can be used, see Section 2.4.2.

Now we explain the term $\text{MMD}_{\mathcal{F}}[\hat{\mathbb{P}}^s, \hat{\mathbb{P}}^t]$ in more detail. Note that $\hat{\mathbb{P}}^s$ and $\hat{\mathbb{P}}^t$ refer to the empirical source and target distributions, respectively. This means that MMD requires unlabeled data from both the source and target domains as inputs, $\{\{\mathbf{x}_i^s\}_{i=1}^{n_s} \cup \{\mathbf{x}_i^t\}_{i=1}^{n_t}\}$. Specifically, the inputs to the MMD are the *linear* output values of the hidden layer, $l(\mathbf{x}) = \mathbf{W}^\top \mathbf{x} \in \mathbb{R}^k$, as given by

$$\text{MMD}_{\mathcal{F}}[\hat{\mathbb{P}}^s, \hat{\mathbb{P}}^t] = \sup_{f \in \mathcal{F}} \left( \frac{1}{n_s} \sum_{i=1}^{n_s} f\left(l(\mathbf{x}_i^s)\right) - \frac{1}{n_t} \sum_{i=1}^{n_t} f\left(l(\mathbf{x}_i^t)\right) \right). \tag{4.2}$$

Let $\mathbf{q}^s := l(\mathbf{x}^s)$ and $\mathbf{q}^t := l(\mathbf{x}^t)$. If the function class $\mathcal{F}$ is chosen such that $\mathcal{F} \subseteq \mathcal{H}$, where $\mathcal{H}$ is a reproducing kernel Hilbert space endowed with a kernel $\kappa : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$, then the squared form of (4.2) can be represented in terms of a kernel as

$$\begin{aligned} \text{MMD}_{\mathcal{F}}^2[\hat{\mathbb{P}}^s, \hat{\mathbb{P}}^t] &= \frac{1}{n_s(n_s-1)} \sum_{i,j=1, i\neq j}^{n_s} \kappa(\mathbf{q}_i^s, \mathbf{q}_j^s) \\ + \frac{1}{n_t(n_t-1)} \sum_{i,j=1, i\neq j}^{n_t} \kappa(\mathbf{q}_i^t, \mathbf{q}_j^t) &- \frac{2}{n_s n_t} \sum_{i=1}^{n_s} \sum_{j=1}^{n_t} \kappa(\mathbf{q}_i^s, \mathbf{q}_j^t). \end{aligned} \tag{4.3}$$

See Lemma 6 in Gretton et al. [87] for a detailed derivation of the above equation.

To apply backpropagation on DaNN, we need to compute the gradient of $J_{\text{DaNN}}$. While computing the gradient $\frac{\partial J_{\text{NN}}}{\partial \mathbf{\Theta}}$ is trivial, computing the gradient of $\text{MMD}_{\mathcal{F}}^2[\hat{\mathbb{P}}^s, \hat{\mathbb{P}}^t]$ depends on the choice of the kernel function. We choose the Gaussian RBF kernel as the kernel function of the form $\kappa_G(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}\right)$, where $\sigma$ is the standard deviation. The main reason for choosing the Gaussian kernel $\kappa_G$ is that it has been well studied and proven to make MMD useful in practice [87]. In addition, the $\kappa_G$ is a universal kernel [145], *i.e.*, any feature map representation of $\kappa_G$ can approximate any continuous function arbitrarily close.

For completeness, we elaborate the calculation of $\frac{\partial \mathrm{MMD}^2_{\mathcal{F}}[\hat{\mathbb{P}}^s,\hat{\mathbb{P}}^t]}{\partial \mathbf{W}}$. Firstly, rewrite the $\mathrm{MMD}^2_{\mathcal{F}}[\hat{\mathbb{P}}^s,\hat{\mathbb{P}}^t]$ function in terms of the Gaussian kernel $\kappa_G$ by a matrix-vector form, which is given by the following equation:

$$\begin{aligned}
&\mathrm{MMD}^2_{\mathcal{F}}[\hat{\mathbb{P}}^s,\hat{\mathbb{P}}^t] = \\
&\frac{1}{n_s(n_s-1)}\sum_{i,j=1}^{n_s}\exp\left(-\frac{(\mathbf{x}_i^s-\mathbf{x}_j^s)^\top \mathbf{W}\mathbf{W}^\top(\mathbf{x}_i^s-\mathbf{x}_j^s)}{2\sigma^2}\right) \\
&+\frac{1}{n_t(n_t-1)}\sum_{i,j=1}^{n_t}\exp\left(-\frac{(\mathbf{x}_i^t-\mathbf{x}_j^t)^\top \mathbf{W}\mathbf{W}^\top(\mathbf{x}_i^t-\mathbf{x}_j^t)}{2\sigma^2}\right) \\
&-\frac{2}{n_s n_t}\sum_{i,j=1}^{n_s,n_t}\exp\left(-\frac{(\mathbf{x}_i^s-\mathbf{x}_j^t)^\top \mathbf{W}\mathbf{W}^\top(\mathbf{x}_i^s-\mathbf{x}_j^t)}{2\sigma^2}\right).
\end{aligned} \tag{4.4}$$

Let $G^{\bullet\bullet}(i,j)$ be the gradient of $\kappa_G(\mathbf{q}_i^\bullet,\mathbf{q}_j^\bullet)$ with respect to $\mathbf{W}$, where the symbol $\bullet$ denotes either the source symbol $s$ or the target symbol $t$ – recall that $\mathbf{q}^\bullet = \mathbf{W}^\top\mathbf{x}^\bullet$. Then, $G^{\bullet\bullet}(i,j)$ takes the form

$$G^{\bullet\bullet}(i,j) = -\frac{1}{\sigma^2}\kappa_G(\mathbf{x}_i^\bullet,\mathbf{x}_j^\bullet)(\mathbf{x}_i^\bullet-\mathbf{x}_j^\bullet)(\mathbf{x}_i^\bullet-\mathbf{x}_j^\bullet)^\top\mathbf{W}. \tag{4.5}$$

Using the above notation, the gradient of $\mathrm{MMD}^l_{\mathcal{F}}[\hat{\mathbb{P}}^s,\hat{\mathbb{P}}^t]$ is given by

$$\begin{aligned}
\frac{\partial \mathrm{MMD}^2_{\mathcal{F}}[\hat{\mathbb{P}}^s,\hat{\mathbb{P}}^t]}{\partial \mathbf{W}} = &\frac{1}{n_s(n_s-1)}\sum_{i,j=1,i\neq j}^{n_s} G^{ss}(i,j) + \\
&\frac{1}{n_t(n_t-1)}\sum_{i,j=1,i\neq j}^{n_t} G^{tt}(i,j) - \\
&\frac{2}{n_s n_t}\sum_{i=1}^{n_s}\sum_{j=1}^{n_t} G^{st}(i,j).
\end{aligned} \tag{4.6}$$

Recall that here MMD is applied to the (linear) net outputs, see (4.3), before the non-linear activation function. This means that MMD provides a biased estimate with respect to the actual distribution discrepancy of the hidden representations. In practice, we use Rectified Linear Unit (ReLU) [153], $\sigma(z) = \max(z, 0)$, as the activation function for the hidden layers. Since ReLU is close to linear, we expect that MMD would be able to provide a good approximation to the true distribution discrepancy.

We now describe the detailed implementation of the DaNN algorithm. The optimization process is divided into two steps: i) minimizing $J_{\text{NN}}$ and ii) minimizing $\text{MMD}^l_{\mathcal{F}}[\hat{\mathbb{P}}^s, \hat{\mathbb{P}}^t]$. $J_{\text{NN}}$ is minimized using a *mini-batched* stochastic gradient descent with respect to $\mathbf{W}$ update. The mini-batched setting has become a standard practice in neural network training to establish a compromise between speed and accuracy. $\text{MMD}^l_{\mathcal{F}}[\hat{\mathbb{P}}^s, \hat{\mathbb{P}}^t]$ is then minimized by re-updating $\mathbf{W}$ with respect to the gradient (4.6). The latter step is accomplished by a *full-batched* gradient descent. The detailed procedure is summarized in Algorithm 4.

**Algorithm 4** The DaNN learning algorithm.

```
Input:
• Labeled source data: ${\mathbf{x}_i^s, \mathbf{y}_i^s} \sim \mathbb{P}^s$
• Unlabeled target data: ${\mathbf{x}_i^t} \sim \mathbb{P}^t_{\mathcal{X}}$
• Hidden and output layer weight matrices: $\mathbf{W} \in \mathbb{R}^{d \times k}$ and $\mathbf{W}^{\text{out}} \in \mathbb{R}^{k \times c}$
• Learning rate: $\alpha > 0$
• MMD regularization constant : $\gamma > 0$
1: Initialize all elements in W and W^out with small random real values
2: while not at end-of-epoch do
3:     Update $\mathbf{\Theta} = \{\mathbf{W}, \mathbf{W}^{\text{out}}\}$ using a mini-batched gradient descent by the
       standard rule as follows:
           $\mathbf{\Theta} \leftarrow \mathbf{\Theta} - \alpha \frac{\partial \ell(\mathbf{y}, f_{\mathbf{\Theta}}(\mathbf{x}))}{\partial \mathbf{\Theta}}$
4:     Update W by a full-batched gradient descent as follows:
           $\mathbf{W} \leftarrow \mathbf{W} - \alpha\gamma \frac{\partial \text{MMD}^2_{\mathcal{F}}[\hat{\mathbb{P}}^s, \hat{\mathbb{P}}^t]}{\partial \mathbf{W}}$
5: end while
Output:
• Learned weights : $\mathbf{W}^*$, $\mathbf{W}^{*\text{out}}$
```

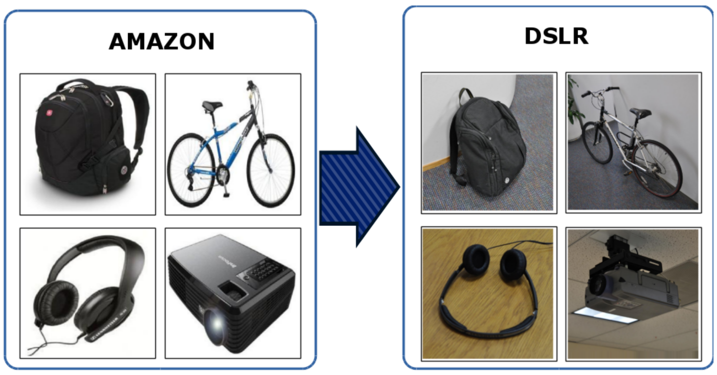


Figure 4.1: The Office data set [187] samples from *amazon* and *dslr* domains.

## 4.3 Experiments and Analysis

This section presents the performance evaluation of DaNN on a range of cross-domain object recognition tasks. The goal is to investigate the DaNN's performance accuracy on both SURF-based features and raw pixels of the Office images [187]. The section starts with the description of the dataset and the experimental setting. It is then followed by the detailed evaluation results and discussions.

### 4.3.1 Settings

**Data setup.** The Office dataset contains images of 31 object classes from three different domains: *amazon*, *webcam*, and *dslr*. In *amazon*, the images contain a single centered object, while for the others the images were acquired in unconstrained settings with some variations such as lighting and background changes. Here we only used 10 object classes following the protocol designed by [83], which leads to a total of 1410 instances. The number of images for *amazon (A)*, *webcam (W)*, and *dslr (D)*, respectively, are 958, 295, and 157. *Webcam* and *dslr* are known to be more similar to each other based on the Rank of Domain (ROD) measure [83]. Examples of the Office images can be seen in Figure 4.1.

**Model's hyper-parameter setting.** The DaNN model used in this experiment has only one hidden layer, i.e., a shallow network of 256 hidden nodes. The number of 256 was selected by grid-searching from a set of values

Table 4.1: Parameter setting of DaNN

| | |
|---|---|
| Learning rate ($\alpha$) | 0.02 |
| Iterations | 900 |
| Momentum | 0.05 |
| L2 weight regularization | 0.003 |
| Dropout fraction | 0.5 |

$\{64, 128, 256, 300, 512\}$. The input layer of the DaNN can be either raw pixels or SURF features. The output layer contains 10 nodes, the same number as the number of classes.

In all our experiments, we used the parameter setting for the DaNN's learning algorithm specified in Table 4.1. Note that we employed the *dropout* regularization [98], which randomly omits a hidden node for each training case under a certain probability. It has been proven to produce better performance in the sense of reducing the overfitting if a neural network is trained from a small training set.

For the MMD regularization, the standard deviation $\sigma$ of the Gaussian kernel was set analytically to the median distance among all datapoints [87],

$$\sigma = \text{median}\, \|\mathbf{a} - \mathbf{b}\|, \forall \mathbf{a}, \mathbf{b} \in \{\{\mathbf{x}_i^s\}_{i=1}^{n_s} \cup \{\mathbf{x}_i^t\}_{i=1}^{n_t}\}$$

Furthermore, the MMD regularization constant $\gamma$ was set to be sufficiently large to accommodate small values of (4.6) compared to $\frac{\partial J_{\text{NN}}}{\partial \mathbf{W}}$ for each iteration. Specifically, we grid-search $\gamma$ to choose the best $\gamma$ value from $\{10^2, 10^3, 10^4, 10^5\}$ according to the validation performance on source data.

We conducted six cross-domain tasks based on the three domains of the Office dataset ($A \to W$, $W \to A$, $A \to D$, $D \to A$, $W \to D$, and $D \to W$). The evaluation was divided into two settings: 1) *unsupervised domain adaptation*, and 2) *semi-supervised domain adaptation.* The *unsupervised domain adaptation* refers to a setting where we can only access the unlabeled target images as the auxiliary information to the labeled source images as the training samples. Meanwhile, the *semi-supervised domain adaptation* still allows a few labeled target images as the additional training samples – first three images per object category from the target domain were selected. Differently from the standard protocol [187], we used all labeled images from the source domain instead of randomly sampled from it.

**Baselines.** Our model performance was compared to SVM-based baselines, two existing domain adaptation methods, and a single layer feedforward neural net.

- **L-SVM**: an SVM [48] model with a linear kernel that was applied to the original features.[1]
- **L-SVM + PCA**: the same model as the L-SVM but preceded by PCA to reduce feature dimensionality.
- **GFK** [83]: the Geodesic Flow Kernel approach by considering an infinite number of intermediate subspaces between the source and target domains followed by k-NN classification. Here we used the subspaces constructed by PCA only.
- **TSC** [133]: the Transfer Sparse Coding technique based on the combination of the graph regularized sparse coding, the MMD regularization, and logistic regression.[2]
- **NN**: a single layer feedforward neural net with the same structure and parameter setting used for DaNN, see Table 4.1.

### 4.3.2 Results on SURF-BoW Features

We first investigated the performance of our model on the standard image features provided by Gong et al. [83]. The image features were acquired by first utilizing the SURF descriptor on resized and grayscaled images to detect local scale-invariant interest points. This was then followed by encoding the data points into 800-bin histograms using a codebook trained from a subset of *amazon* images [187]. The final features, which we refer to as SURF-BoW, were then normalized and z-scored to have zero mean and unit variance. We conducted the *unsupervised domain adaptation* setting evaluation with the results shown in Table 4.2.

[1]`http://www.csie.ntu.edu.tw/~cjlin/liblinear`

[2]`http://learn.tsinghua.edu.cn:8080/2011310560/long.html`

We found that DaNN and TSC have better performance than the other approaches on these standard features. More specifically, DaNN performs well when there is the *amazon* set in a particular domain pairs. In the case of *webcam-dslr* shifts, TSC, which has not been tested on the Office dataset in the previous work, is the best model. Despite its effectiveness, TSC has a longer feature extraction time than our method so that it is less efficient in a real world situation.

We also noted that the GFK, which incorporates multiple intermediate subspaces, fails to surpass the baselines in a few shift cases. This indicates that the projection onto the subspaces generated by GFK is insufficient to reduce the domain mismatch.

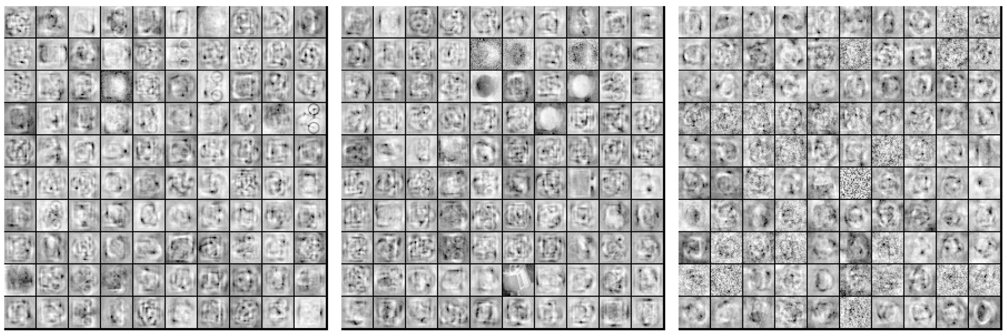
Table 4.2: *Unsupervised domain adaptation* performance accuracy (%) on the Office data set ($A$ : *amazon*, $W$ : *webcam*, $D$ : *dslr*) on **SURF-BoW** features provided by Gong et al. [83] as inputs. Bold-red and bold-black indicate the best and second best performance.

| Methods | $A \to W$ | $W \to A$ | $A \to D$ | $D \to A$ | $W \to D$ | $D \to W$ |
|---|---|---|---|---|---|---|
| L-SVM | $24.1 \pm 0.0$ | $35.8 \pm 0.0$ | $28.0 \pm 0.0$ | $32.7 \pm 0.0$ | $77.7 \pm 0.0$ | $78.0 \pm 0.0$ |
| PCA + L-SVM | $34.9 \pm 0.0$ | $34.8 \pm 0.0$ | $35.0 \pm 0.0$ | $32.2 \pm 0.0$ | $63.7 \pm 0.0$ | $65.4 \pm 0.0$ |
| GFK [83] | $39.0 \pm 0.0$ | $29.8 \pm 0.0$ | $36.3 \pm 0.0$ | $31.8 \pm 0.0$ | $80.3 \pm 0.0$ | $75.6 \pm 0.0$ |
| TSC [133] | **$47.4 \pm 1.7$** | **$39.1 \pm 0.4$** | $46.2 \pm 1.4$ | **$41.6 \pm 0.8$** | **$93.6 \pm 0.5$** | **$93.5 \pm 0.6$** |
| NN | **$44.4 \pm 0.6$** | **$37.3 \pm 0.1$** | **$46.8 \pm 0.9$** | $34.8 \pm 0.2$ | $81.5 \pm 0.0$ | $78.9 \pm 0.0$ |
| **DaNN** | **$45.4 \pm 0.8$** | **$38.7 \pm 0.2$** | **$49.0 \pm 0.7$** | **$38.1 \pm 0.3$** | **$83.4 \pm 0.0$** | **$81.0 \pm 0.0$** |

## 4.3.3 Results on Raw Pixels

Next we conducted the cross-domain recognition performance evaluation on raw pixels of the Office images. Previous works on the Office image set were mostly done using the SURF-based features. It is worth investigating the performance on the Office raw pixels directly since good models on raw pixels are preferable in the sense of reducing the need for manual feature extraction processes. We first converted the pixels of the Office images in 2D RGB values into grayscaled pixels and resized them into a dimension of $28 \times 28$. They were then z-scored to have zero mean and unit variance.

**Cross-domain recognition.** In this experiment, we ran both the *unsupervised domain adaptation* and *semi-supervised domain adaptation* setting for all domain pairs. In addition, we also investigated the effect of denoising autoencoder (DAE) pretraining that precedes the NN and DaNN supervised training with respect to the performance. The use of DAE pretraining will slightly change Step 1 of Algorithm 4. We denoted these models as **DAE** + NN and **DAE** + DaNN. Examples of the pretrained weights are depicted in Figure 4.2. The complete accuracy rates on the Office raw pixels for all domain pairs are presented in Table 4.3.



(a) *amazon-webcam* (b) *amazon-dslr* (c) *webcam-dslr*

Figure 4.2: The 2D visualization of 100 randomly chosen weights after the DAE pretraining for each domain pairs from the Office image set. The white, gray, and black pixels in each box indicate the high-positive, close-to-zero or zero, and high-negative values of a particular connection weight. The type of noise used here is the zero-masking noise with 30% destruction.

It is clear that our DaNN always provides accuracy improvements in all domain pairs compared to the SVM-based baselines and the NN model. In other words, the MMD regularization indeed improves the performance of neural networks. Compared to TSC that also employs the MMD regularization in the unsupervised training stage, our DaNN performs better in most cases. However, TSC can match the DaNN performance on *webcam-dslr* pairs, which has lower level mismatch than the other pairs. This indicates that the utilization of the MMD regularization in the supervised training might gain more adaptation ability than

Table 4.3: The performance on the Office dataset ($A$ : *amazon*, $W$ : *webcam*, $D$ : *dslr*) using the **raw pixels** as inputs.

| Methods | $A \rightarrow W$ | $W \rightarrow A$ | $A \rightarrow D$ | $D \rightarrow A$ | $W \rightarrow D$ | $D \rightarrow W$ |
|---|---|---|---|---|---|---|
| Unsupervised Setting | | | | | | |
| L-SVM | $14.9 \pm 0.0$ | $14.7 \pm 0.0$ | $19.1 \pm 0.0$ | $13.7 \pm 0.0$ | $36.0 \pm 0.0$ | $40.3 \pm 0.0$ |
| PCA + L-SVM | $20.3 \pm 0.0$ | $18.1 \pm 0.0$ | $16.9 \pm 0.0$ | $17.4 \pm 0.0$ | $40.4 \pm 0.0$ | $37.0 \pm 0.0$ |
| GFK [83] | $21.4 \pm 0.0$ | $15.0 \pm 0.0$ | $30.2 \pm 0.0$ | $13.8 \pm 0.0$ | $69.1 \pm 0.0$ | $65.0 \pm 0.0$ |
| TSC [133] | $22.3 \pm 1.0$ | $15.7 \pm 1.1$ | $25.6 \pm 1.6$ | $19.6 \pm 0.7$ | $\mathbf{74.1 \pm 1.9}$ | $\mathbf{67.5 \pm 1.5}$ |
| NN | $29.2 \pm 0.6$ | $17.0 \pm 0.3$ | $32.5 \pm 0.7$ | $15.0 \pm 0.3$ | $63.7 \pm 0.0$ | $57.3 \pm 0.0$ |
| DAE + NN | $32.5 \pm 0.2$ | $18.7 \pm 0.0$ | $\mathbf{37.8 \pm 0.2}$ | $17.4 \pm 0.0$ | $\mathbf{72.1 \pm 0.0}$ | $65.9 \pm 0.0$ |
| **DaNN** | $\mathbf{34.1 \pm 0.3}$ | $\mathbf{21.2 \pm 0.2}$ | $34.0 \pm 0.8$ | $\mathbf{20.1 \pm 0.5}$ | $64.4 \pm 0.0$ | $62.0 \pm 0.0$ |
| **DAE + DaNN** | $\mathbf{35.0 \pm 0.2}$ | $\mathbf{23.1 \pm 0.0}$ | $\mathbf{39.4 \pm 0.3}$ | $\mathbf{22.5 \pm 0.0}$ | $\mathbf{74.3 \pm 0.0}$ | $\mathbf{70.5 \pm 0.0}$ |
| Semi-supervised Setting | | | | | | |
| L-SVM | $18.9 \pm 0.0$ | $29.0 \pm 0.0$ | $25.2 \pm 0.0$ | $35.2 \pm 0.0$ | $45.7 \pm 0.0$ | $52.5 \pm 0.0$ |
| PCA + L-SVM | $20.8 \pm 0.0$ | $31.0 \pm 0.0$ | $25.6 \pm 0.0$ | $35.1 \pm 0.0$ | $50.4 \pm 0.0$ | $50.2 \pm 0.0$ |
| GFK [83] | $47.9 \pm 0.0$ | $33.1 \pm 0.0$ | $52.0 \pm 0.0$ | $31.8 \pm 0.0$ | $80.3 \pm 0.0$ | $\mathbf{74.7 \pm 0.0}$ |
| TSC [133] | $42.4 \pm 2.1$ | $34.1 \pm 0.8$ | $49.3 \pm 2.2$ | $36.4 \pm 0.9$ | $76.3 \pm 1.4$ | $\mathbf{71.1 \pm 1.1}$ |
| NN | $48.7 \pm 0.3$ | $34.5 \pm 0.3$ | $52.8 \pm 0.6$ | $36.2 \pm 0.4$ | $75.6 \pm 0.1$ | $67.2 \pm 0.0$ |
| DAE + NN | $\mathbf{52.8 \pm 0.1}$ | $\mathbf{36.8 \pm 0.0}$ | $\mathbf{57.5 \pm 0.1}$ | $36.5 \pm 0.0$ | $\mathbf{83.5 \pm 0.0}$ | $69.4 \pm 0.0$ |
| **DaNN** | $51.3 \pm 0.5$ | $\mathbf{36.6 \pm 0.4}$ | $55.9 \pm 0.3$ | $\mathbf{37.9 \pm 0.3}$ | $\mathbf{78.0 \pm 0.2}$ | $70.2 \pm 0.0$ |
| **DAE + DaNN** | $\mathbf{53.6 \pm 0.2}$ | $\mathbf{37.3 \pm 0.0}$ | $\mathbf{59.9 \pm 0.1}$ | $\mathbf{38.2 \pm 0.0}$ | $\mathbf{83.5 \pm 0.0}$ | $\mathbf{71.2 \pm 0.0}$ |

that in the unsupervised training for pairs with higher mismatch.

The DAE pretraining applied to NN and DaNN indeed improves the performance for all pairs of domains. The improvements are quite significant for several cases, especially for *webcam-dslr* pairs. In general, the DAE pretraining also produces more stable models in the sense of resulting in lower standard deviations over 30 independent runs. Furthermore, the combination of DAE pretraining and DaNN performs best among other methods in these experiments in almost all cases. In the sense of qualitative analysis, as can be seen in Figure 4.2, the DAE pretraining captures more distinctive "filters" from local blob detectors to object parts detectors, especially when the *amazon* images are included. This effect is somewhat similar to what was found in the initial DAE work [225], which might be the reason of the performance gain.

In the semi-supervised setting, the performance trend is somewhat similar to the unsupervised setting. However, the performance discrepancies between NN and DaNN here becomes smaller than those in the unsupervised setting. This outcome also holds for the case of the DAE pretraining. This suggests that both the MMD regularization and DAE pretraining might be less impactful when some labeled images from the target domain can be acquired.

**In-domain recognition.** One may ask whether the domain adaptation results shown in Table 4.3 are reasonable compared to the standard learning setting. We refer this standard setting to as the *in-domain* setting, where the training and test samples come from the same domain. The *in-domain* performance can be considered as the *upper bound* that indicates whether it is worthwhile to build better domain adaptation models for a particular application.

To demonstrate this, we investigated the *in-domain* performance of non-domain adaptive models described in Section 4.3.1, i.e., L-SVM, PCA+L-SVM, and NN on raw pixels of the Office images. For each domain, we conducted 10-fold cross validation where the data set is divided into ten subsets of size $n/10$, nine subsets as the training set and one as the test set. The evaluation is then repeated 10 times for each different configuration of training and test sets. The complete *in-domain* results in terms of the mean and standard deviation are shown in Table 4.4.

Table 4.4: *In-domain* performance on the Office data set using 10-fold cross validation on each domain.

| Methods | *amazon* | | *webcam* | | *dslr* | |
|---|---|---|---|---|---|---|
| | Training | Test | Training | Test | Training | Test |
| L-SVM | 99.0 ± 0.3 | 52.0 ± 4.6 | 100.0 ± 0.0 | 57.7 ± 13.9 | 100.0 ± 0.0 | 51.0 ± 14.1 |
| PCA+L-SVM | 64.4 ± 0.8 | 60.6 ± 6.4 | 72.0 ± 1.5 | 62.8 ± 8.7 | 75.6 ± 2.1 | 55.2 ± 13.1 |
| NN | 99.3 ± 0.1 | **74.2 ± 3.2** | 100.0 ± 0.0 | **87.2 ± 5.4** | 100.0 ± 0.0 | **77.9 ± 8.8** |

In general, we can see that the best *in-domain* model is the NN model on both training and test images. PCA+L-SVM performs better than L-SVM on test sets, but worse on training sets. We also note that the standard deviations of the test accuracies are relatively high for all methods, which might indicate that the image variation within a particular domain in the Office set is high.

The most interesting outcome here is that the best *in-domain* accuracies are, in general, still higher than the best performance with the domain mismatch when the *amazon* or *webcam* are used as the target sets (see the highest accuracy rates in column $D \to A$ and $D \to W$ on Table 4.3). However, this is not case for $W \to D$ pair. This means that we might not need a better domain adaptation model any more for $W \to D$ pair since the upper bound has been surpassed.

## 4.4 Chapter Summary

This chapter presents a neural network-based domain adaptation algorithm that minimizes the distribution difference between the source and target data representations in the hidden layer space. The distribution difference minimization is established by incorporating the Maximum Mean Discrepancy (MMD) measure that acts as a regularization in the label training. The resulting model is referred to as Domain Adaptive Neural Network (DaNN). The model performance was evaluated on six cross-domain recognition tasks taken from the Office dataset [187] and compared with that of several baselines.

Firstly, we show that DaNN performs well on SURF-BoW features, which are the handcrafted features taken from a particular combination of SURF and codebook training algorithms, comparable to Transfer Sparse Coding (TSC) [133], a prior state-of-the-art domain adaptation algorithm. Specifically, DaNN provides the best performance on three cross-domain recognition tasks: $A \rightarrow W$, $W \rightarrow A$, and $A \rightarrow D$, while TSC provides best on the other three tasks. Note that TSC requires a much longer training time than DaNN.

We then show that DaNN also performs well on raw image pixels under both the *unsupervised domain adaptation* and *semi-supervised domain adaptation* settings. In this case, the DaNN's performance is higher than the TSC's performance with a considerable gap in almost all cross-domain tasks. Furthermore, the DaNN preceded by the denoising autoencoder (DAE) training, i.e., DAE+DaNN, provides the best cross-domain recognition performance.

Despite the effectiveness of the MMD regularization in the supervised backpropagation training, there are still many aspects that can be further improved. We have seen that the performance on raw pixels, which is a main concern in representation learning approach, is still not as good as that on SURF-BoW features. We note that good models that perform well without any preceding handcrafted feature extractors are more desirable to reduce complexity. We might achieve a better model on raw pixels by using deeper neural network layers with a similar strategy since deep architectures have brought some successes in many applications in recent years [18]. Our initial work using deep architectures with the DAE pretraining, which is not shown here, suggested that deeper representations do not always improve the performance against the domain mismatch.

In addition, a study on the kernel choice for computing MMD regarding to the domain adaptation problem might be a potential direction. We assumed that the universal Gaussian kernel function can detect any underlying distribution mismatches in the Office data set, which might be not true. A better understanding about the relationship between a kernel function and a real-world image mismatch, *e.g.*, background, lighting, affine transformation changes, would have a great impact in this field of research.

The next contribution chapter will address the problem of domain generalization: learning a robust model from one or multiple source domains with no access to target domains. This is different from domain adaptation in which we can still utilize unlabeled target data; we do not have both labeled and unlabeled data in domain generalization. Our solution to the problem is a multi-task feature learning algorithm based on autoencoders that we refer to as Multi-task Autoencoders (MTAE).

# 5 Domain Generalization with Multi-task Autoencoders

*This chapter presents the third contribution of this thesis. We develop a state-of-the-art multi-task representation learning algorithm for domain generalization referred to as Multi-task Autoencoders (MTAE). MTAE learns to transform the original image into analogs in multiple related domains. It thereby learns features used as inputs to a classifier, which are robust to variations across domains.*

## 5.1 Introduction

In object recognition, the "visual worl" can be considered as decomposing into *views* (e.g. perspectives or lighting conditions) corresponding to domains. For example, frontal-views and 45° rotated-views correspond to two different domains. Alternatively, we can associate views or domains with standard image datasets such as PASCAL VOC2007 [62], and Office [187]. The dataset bias problem arises where a model learned from one view attempts to recognize objects from another view. The challenge is thus to build a system that recognizes objects in previously *unseen* datasets, given one or multiple training datasets, which is referred to as *domain generalization* [25, 152].

### 5.1.1 Chapter Goals

The goal of this chapter is to learn features using an autoencoder method that improve generalization performance across views/domains. Autoencoders were introduced to address the problem of "backpropagation without a teacher" by using *inputs as labels* – and learning to reconstruct them with minimal distortion [21,182]. Denoising autoencoders in particular are a powerful basic circuit for unsupervised representation learning [224]. Intuitively, corrupting inputs forces autoencoders to learn representations that are robust to noise.

This chapter proposes a broader view: autoencoders are *generic circuits for learning invariant features*. The main contribution is a new training strategy based on naturally occurring transformations such as: rotations in viewing angle, dilations in apparent object size, and shifts in lighting conditions. The resulting model, Multi-task Autoencoder (MTAE), learns features that are robust to real-world image variability, and therefore generalize well across domains.

To achieve the above goal, we establish the following specific objectives:

- Whether MTAE produces more discriminative features than commonly used autoencoder-based models in the context of domain generalization;
- Whether the MTAE's learned weights associated with the extracted features form a visually interpretable "filter" indicating that it captures the commonality, *e.g.*, object transformation, among views / domains;
- Whether MTAE provides better cross-domain recognition performance than prior state-of-the-art domain generalization models on modern benchmark datasets.

### 5.1.2 Chapter Organization

The remainder of this chapter is organized as follows. The proposed algorithm, Multi-task Autoencoder (MTAE), is described in Section 5.2. Section 5.3 presents the evaluation on MNIST and ETH-80 datasets. It investigates the behaviour of MTAE in comparison to standard single-task autoencoder models on raw pixels as proof-of-principle. Section 5.4 evaluates the performance of MTAE against several state-of-the-art algorithms on modern object datasets such as the Office [187], Caltech [88], PASCAL VOC2007 [62], LabelMe [186], and SUN09 [40]. Section 5.5 summarizes the chapter.

## 5.2 The Algorithm

Our goal is to learn features that provide good domain generalization. To do so, we extend the autoencoder [30] into a model that jointly learns multiple data-reconstruction tasks taken from related domains. Our strategy is motivated by prior work demonstrating that learning from multiple related tasks can improve performance on a novel, yet related, task – relative to methods trained on a single-task [7, 14, 35, 211].

### 5.2.1 Autoencoders

Autoencoders (AE) have become established as a pretraining model for deep learning [21]. The autoencoder training consists of two stages: 1) *encoding* and 2) *decoding*. Given an unlabeled input $\mathbf{x} \in \mathbb{R}^{d_x}$, a single hidden layer autoencoder $f_{\mathbf{\Theta}}(\mathbf{x}) : \mathbb{R}^{d_x} \to \mathbb{R}^{d_x}$ can be formulated as

$$\begin{aligned} \mathbf{h} &= \sigma_{\text{enc}}(\mathbf{W}^\top \mathbf{x}) \\ \hat{\mathbf{x}} &= \sigma_{\text{dec}}(\mathbf{V}^\top \mathbf{h}) = f_{\mathbf{\Theta}}(\mathbf{x}), \end{aligned} \tag{5.1}$$

where $\mathbf{W} \in \mathbb{R}^{d_x \times d_y}$, $\mathbf{V} \in \mathbb{R}^{d_y \times d_x}$ are *input-to-hidden* and *hidden-to-output* connection weights[1] respectively, $\mathbf{h} \in \mathbb{R}^{d_h}$ is the hidden node vector, and $\sigma_{\text{enc}}(\cdot) = [s_{\text{enc}}(z_1), ..., s_{\text{enc}}(z_{d_h})]^\top, \sigma_{\text{dec}}(\cdot) = [s_{\text{dec}}(z_1), ..., s_{\text{dec}}(z_{d_x})]^\top$ are element-wise non-linear activation functions, and $s_{\text{enc}}$ and $s_{\text{dec}}$ are not necessarily identical. Popular choices for the activation function $s(\cdot)$ are, e.g., the sigmoid $s(a) = (1+\exp(-a))^{-1}$

[1]While the bias terms are incorporated in our experiments, they are intentionally omitted from equations for the sake of simplicity.

and the rectified linear (ReLU) $s(a) = \max(0, a)$.

Let $\mathbf{\Theta} = \{\mathbf{W}, \mathbf{V}\}$ be the autoencoder parameters and $\{\mathbf{x}_i\}_{i=1}^N$ be a set of $N$ input data. Learning corresponds to minimizing the following objective

$$\hat{\mathbf{\Theta}} := \arg\min_{\mathbf{\Theta}} \sum_{i=1}^{N} \mathcal{L}\left(f_{\mathbf{\Theta}}(\mathbf{x}_i), \mathbf{x}_i\right) + \eta \mathcal{R}\left(\mathbf{\Theta}\right), \tag{5.2}$$

where $\mathcal{L}(\cdot, \cdot)$ is the loss function, usually in the form of *least square* or *cross-entropy* loss, and $\mathcal{R}(\cdot)$ is a regularization term used to avoid overfitting. The objective (5.2) can be optimized by the backpropagation algorithm [184]. If we apply autoencoders to raw pixels of visual object images, the weights $\mathbf{W}$ usually form visually meaningful “filters" that can be interpreted qualitatively.

To create a discriminative model using the learned autoencoder model, either of the following options can be considered: 1) the feature map $\phi(\mathbf{x}) := \sigma_{\text{enc}}(\hat{\mathbf{W}}^\top \mathbf{x})$ is extracted and used as an input to supervised learning algorithms while keeping the weight matrix $\hat{\mathbf{W}}$ fixed; 2) the learned weight matrix $\hat{\mathbf{W}}$ is used to initialize a neural network model and is updated during the supervised neural network training (*fine-tuning*).

Recently, several variants such as denoising autoencoders (DAE) [225] and contractive autoencoders (CAE) [178] have been proposed to extract features that are more robust to small changes of the input. In DAEs, the objective is to reconstruct a *clean* input $\mathbf{x}$ given its *corrupted* counterpart $\tilde{\mathbf{x}} \sim \mathcal{Q}(\tilde{\mathbf{x}}|\mathbf{x})$. Commonly used types of corruption are zero-masking, Gaussian, and salt-and-pepper noise. Features extracted by DAE have been proven to be more discriminative than ones extracted by AE [225].

### 5.2.2 Multi-task Autoencoders

We refer to our proposed domain generalization algorithm as *Multi-task Autoencoder* (MTAE). From an architectural viewpoint, MTAE is an autoencoder with multiple output layers, see Fig. 5.1. The input-hidden weights represent *shared* parameters and the hidden-output weights represent *domain-specific* parameters. The architecture is similar to the supervised multi-task neural networks proposed by Caruana [35]. The main difference is that the output layers of MTAE correspond to different domains instead of different class labels.

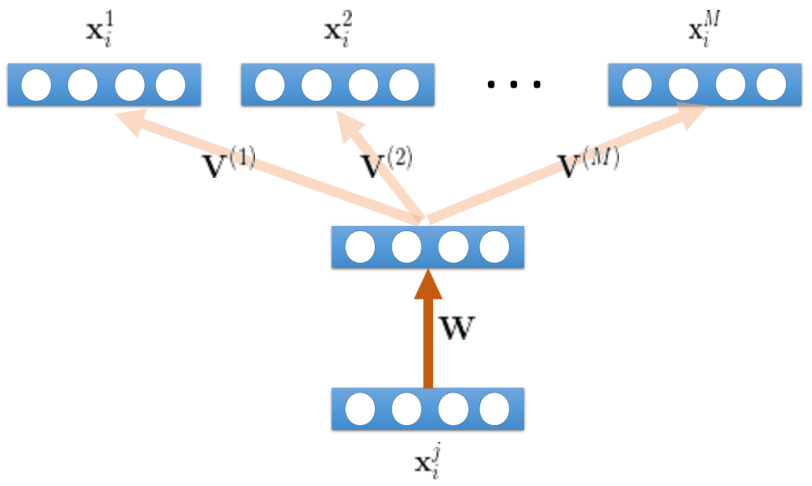


Figure 5.1: The single-layer Multi-task Autoencoder (MTAE) architecture, which consists of three layers with multiple separated outputs; each output corresponds to one task/domain.

The most important component of MTAE is the training strategy, which constructs a generalized denoising autoencoder that learns invariances to naturally occurring transformations. Denoising autoencoders focus on the special case where the transformation is simply noise. In contrast, MTAE training treats a specific perspective on an object as the "corrupted" counterpart of another perspective (e.g., a rotated digit 6 is the noisy pair of the original digit). The autoencoder objective is then reformulated along the lines of multi-task learning: the model aims to *jointly achieve good reconstruction of all source views given a particular view*. For example, applying the strategy to handwritten digit images with several views, MTAE learns representations that are invariant across the source views, see Section 5.3.

Two types of reconstruction tasks are performed during MTAE training: 1) ***self-domain*** reconstruction and 2) ***between-domain*** reconstruction. Given $M$ source domains, there are $M \times M$ reconstruction tasks, of which $M$ tasks are self-domain reconstructions and the remaining $M \times (M - 1)$ tasks are between-domain reconstructions. Note that the self-domain reconstruction is identical to the standard autoencoder reconstruction (5.1).

**Formal description.** Let $\{\mathbf{x}_i^l\}_{i=1}^{n_l}$, be a set of $d_x$-dimensional data points in the $l^{\text{th}}$ domain, where $l \in \{1, ..., M\}$. Each domain's data points are combined into a matrix $\mathbf{X}^l \in \mathbb{R}^{n_l \times d_x}$, where $\mathbf{x}_i^{l\top}$ is its $i^{\text{th}}$ row, such that $(\mathbf{x}_i^1, \mathbf{x}_i^2, \ldots \mathbf{x}_i^M)$ form a category-level correspondence. This configuration enforces the number of samples in a category to be the same in every domain. Note that such a configuration is necessary to ensure that the *between-domain* reconstruction works (we will discuss how to handle the case with unbalanced samples in Section 5.2.3). The input and output pairs used to train MTAE can then be written as concatenated matrices

$$\begin{aligned} \bar{\mathbf{X}} &= [\mathbf{X}^1; \mathbf{X}^2; ...; \mathbf{X}^M], \\ \bar{\mathbf{X}}^l &= [\mathbf{X}^l; \mathbf{X}^l; ...; \mathbf{X}^l] \end{aligned} \tag{5.3}$$

where $\bar{\mathbf{X}}, \bar{\mathbf{X}}^l \in \mathbb{R}^{N \times d_x}$ and $N = \sum_{l=1}^M n_l$. In words, $\bar{\mathbf{X}}$ is the matrix of data points taken from all domains and $\bar{\mathbf{X}}^l$ is the matrix of replicated data sets taken from the $l^{\text{th}}$ domain. The replication imposed in $\bar{\mathbf{X}}^l$ constructs input-output pairs for the autoencoder learning algorithm. In practice, the algorithm can be implemented efficiently – without replicating the matrix in memory.

We now describe MTAE more formally. Let $\bar{\mathbf{x}}_i^\top$ and $\bar{\mathbf{x}}_i^{l\top}$ be the $i^{\text{th}}$ row of matrices $\bar{\mathbf{X}}$ and $\bar{\mathbf{X}}^l$, respectively, the feedforward MTAE reconstruction is

$$\begin{aligned} \mathbf{h}_i &= \sigma_{\text{enc}}(\mathbf{W}^\top \bar{\mathbf{x}}_i), \\ f_{\mathbf{\Theta}^{(l)}}(\bar{\mathbf{x}}_i) &= \sigma_{\text{dec}}(\mathbf{V}^{(l)\top} \mathbf{h}_i), \end{aligned} \tag{5.4}$$

where $\mathbf{\Theta}^{(l)} = \{\mathbf{W}, \mathbf{V}^{(l)}\}$ contains the matrices of shared and individual weights, respectively.

The MTAE training is achieved as follows. Let us define the loss function summed over the datapoints

$$J(\mathbf{\Theta}^{(l)}) = \sum_{i=1}^N \mathcal{L}\left(f_{\mathbf{\Theta}^{(l)}}(\bar{\mathbf{x}}_i), \bar{\mathbf{x}}_i^l\right). \tag{5.5}$$

Given $M$ domains, training MTAE corresponds to minimizing the objective

$$\hat{\mathbf{\Theta}}^{(l)} := \arg\min_{\mathbf{\Theta}^{(l)}} \sum_{l=1}^M J(\mathbf{\Theta}^{(l)}) + \eta \mathcal{R}(\mathbf{\Theta}^{(l)}), \tag{5.6}$$

where $\mathcal{R}(\mathbf{\Theta}^{(l)})$ is a regularization term. In this work, we use the standard $l_2$-norm weight penalty $\mathcal{R}(\mathbf{\Theta}^{(l)}) = \|\mathbf{W}\|_2^2 + \|\mathbf{V}^{(l)}\|_2^2$. Stochastic gradient descent is applied on each reconstruction task to achieve the objective (5.6). Once training is completed, the optimal *shared* weights $\hat{\mathbf{W}}$ are obtained. The stopping criterion is empirically determined by monitoring the average loss over all reconstruction tasks during training – the process is stopped when the average loss stabilizes. The detailed steps of the MTAE training is summarized in Algorithm 5.

**Algorithm 5** The MTAE feature learning algorithm.

```
Input:
• Data matrices based on (5.3): $\bar{\mathbf{X}}$ and $\bar{\mathbf{X}}^l, \forall l \in \{1, ..., M\}$;
• Source labels: $\{y_i^l\}_{i=1}^{n_l}, \forall l \in \{1, ..., M\}$;
• The learning rate: $\alpha$;
 1: Initialize $\mathbf{W} \in \mathbb{R}^{d_x \times d_h}$ and $\mathbf{V}^{(l)} \in \mathbb{R}^{d_h \times d_x}$, $\forall l \in \{1, ..., M\}$ with small
    random real values;
 2: while not end of epoch do
 3:   Do RAND-SEL as described in Section 5.2.3 to balance the number of
      samples per categories in $\bar{\mathbf{X}}$ and $\bar{\mathbf{X}}^l$;
 4:   for $l = 1$ to $M$ do
 5:     for all row of $\tilde{\mathbf{X}}$ do
 6:       Do a forward pass based on (5.4);
 7:       Update $\mathbf{W}$ and $\mathbf{V}^{(l)}$ to achieve the objective (5.6) with respect to
          the following rules
          $$\begin{aligned} V_{ij}^{(l)} &\leftarrow V_{ij}^{(l)} - \alpha \frac{\partial J(\{\mathbf{W}, \mathbf{V}^{(l)}\})}{\partial V_{ij}^{(l)}}, \\ W_{ij} &\leftarrow W_{ij} - \alpha \frac{\partial J(\{\mathbf{W}, \mathbf{V}^{(l)}\})}{\partial W_{ij}}; \end{aligned}$$
 8:     end for
 9:   end for
10: end while
Output:
• MTAE learned weights: $\hat{\mathbf{W}}$ $\forall l \in \{1, ..., M\}$;
```

The training protocol can be supplemented with a denoising criterion as in [225] to induce more robust-to-noise features. To do so, we simply replace $\bar{\mathbf{x}}_i$ in (5.4) with its corrupted counterpart $\tilde{\bar{\mathbf{x}}}_i \sim Q(\tilde{\bar{\mathbf{x}}}_i|\bar{\mathbf{x}}_i)$. We name the MTAE model after applying the denoising criterion the *Denoising Multi-task Autoencoder* (D-MTAE).

### 5.2.3 Handling unbalanced samples per category

MTAE requires that every instance in a particular domain has a category-level corresponding pair in every other domain. MTAE's apparent applicability is therefore limited to situations where the number of source samples per category is the same in every domain. However, unbalanced samples per category occur frequently in applications. To overcome this issue, we propose a simple *random selection* procedure applied in the *between-domain* reconstructions, denoted by RAND-SEL, which is simply balancing the samples per category while keeping their category-level correspondence.

In detail, the RAND-SEL strategy is as follows. Let $m_c$ be the number of subsamples in the $c$-th category, where $m_c = \min(n_{1c}, n_{2c}, \ldots, n_{Mc})$ and $n_{lc}$ is the number of samples in the $c$-th category of domain $l \in \{1, \ldots, M\}$. For each category $c$ and each domain $l$, select $m_c$ samples randomly such that $n_{lc} = n_{2c} = \ldots n_{Mc} = m_c$. This procedure is executed in every iteration of the MTAE algorithm, see Line 3 of Algorithm 5.

Although one can argue that this procedure might reduce the optimality of the multi-task strategy because of the random selection, this is perhaps the simplest way to ensure the applicability of MTAE on the unbalanced sample situation. In the experiments, we will show that this strategy works well on some cross-dataset evaluations, see Section 5.4.

## 5.3 Experiment 1: Cross-domain Recognition on MNIST and ETH-80

We conducted experiments on several real world object datasets to evaluate the domain generalization ability of our proposed system. In Section 5.3, we investigate the behaviour of MTAE in comparison to standard single-task autoencoder models on raw pixels as proof-of-principle. In Section 5.4, we evaluate the performance of MTAE against several state-of-the-art algorithms on modern object datasets such as the Office [187], Caltech [88], PASCAL VOC2007 [62], LabelMe [186], and SUN09 [40].

In this part, we aim to understand MTAE's behavior when learning from multiple domains that form physically reasonable object transformations such as roll, pitch rotation, and dilation. The task is to categorize objects in views (domains) that were not presented during training. We evaluate MTAE against several autoencoder models. To perform the evaluation, a variety of object views were constructed from the MNIST handwritten digit [125] and the ETH-80 object [129] datasets.

## 5.3.1 Data Setup

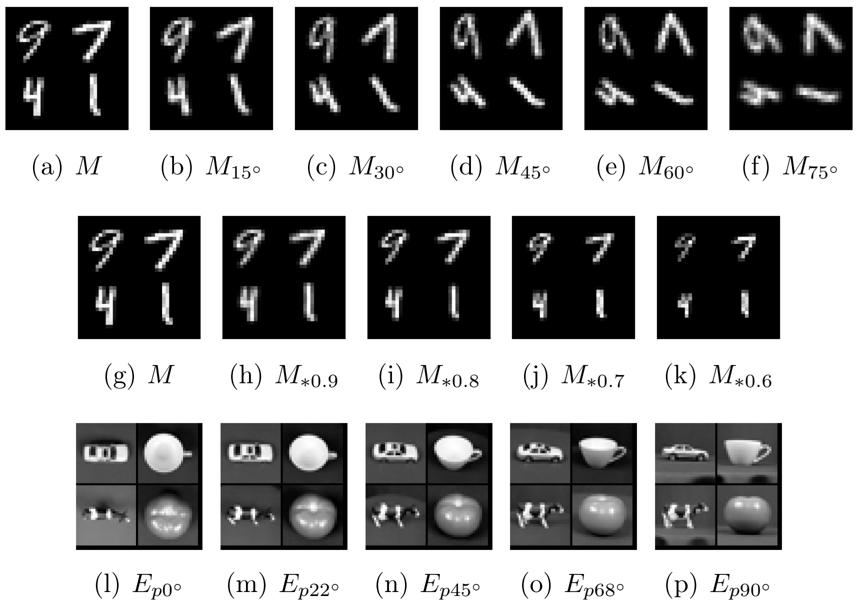
We created four new datasets from MNIST and ETH-80 images: 1) MNIST-r, 2) MNIST-s, 3) ETH80-p, and 4) ETH80-y. These new sets contain multiple domains so that every instance in one domain has a pair in another domain. The detailed setting for each dataset is as follows.

**MNIST-r** contains six domains, each corresponding to a degree of roll rotation. We randomly chose 1000 digit images of ten classes from the original MNIST training set to represent the *basic* view, i.e., 0 degree of rotation;[2] each class has 100 images. Each image was subsampled to a $16 \times 16$ representation to simplify the computation. This subset of 1000 images is denoted by $M$. We then created 5 rotated views from $M$ with $15°$ difference in counterclockwise direction, denoted by $M_{15°}$, $M_{30°}$. $M_{45°}$, $M_{60°}$, and $M_{75°}$. The **MNIST-s** is the counterpart of MNIST-r, where each domain corresponds to a dilation/scaling factor. The views are denoted by $M$, $M_{*0.9}$, $M_{*0.8}$, $M_{*0.7}$, and $M_{*0.6}$, where the subscripts represent the dilation factors with respect to the original view $M$.

The **ETH80-p** consists of eight object classes with 10 subcategories for each class. In each subcategory, there are 41 different views with respect to pose angles. We took five views from each class denoted by $E_{p0°}$, $E_{p22°}$, $E_{p45°}$, $E_{p68°}$, and $E_{p90°}$, which represent the horizontal poses, i.e., pitch-rotated views starting from the top view to the side view. This makes the number of instances only 80 for each view. We then greyscaled and subsampled the images to $28 \times 28$. The **ETH80-y** contains

[2]Note that the rotation angle of the basic view is not perfectly $0°$ since the original MNIST images have varying appearances.

five views of the ETH-80 representing the vertical poses, i.e., yaw-rotated views starting from the right-side view to the left-side view denoted by $E_{+y90^\circ}$, $E_{+y45^\circ}$, $E_{y0^\circ}$, $E_{-y45^\circ}$, and $E_{-y90^\circ}$. Other settings such as the image dimensionality and preprocessing stage are similar to ETH80-p. Examples of the resulting views are depicted in Figure 5.2.


(a) $M$ (b) $M_{15^\circ}$ (c) $M_{30^\circ}$ (d) $M_{45^\circ}$ (e) $M_{60^\circ}$ (f) $M_{75^\circ}$

(g) $M$ (h) $M_{*0.9}$ (i) $M_{*0.8}$ (j) $M_{*0.7}$ (k) $M_{*0.6}$

(l) $E_{p0^\circ}$ (m) $E_{p22^\circ}$ (n) $E_{p45^\circ}$ (o) $E_{p68^\circ}$ (p) $E_{p90^\circ}$

Figure 5.2: Some image examples from the MNIST-r, MNIST-s, and ETH80-p. A domain is represented by a particular view.

### 5.3.2 Algorithms for Comparison and Parameter Settings

We compared the classification performance of our models with several single-task autoencoder models: Descriptions of the methods and their hyperparameter settings are provided below.

- **AE** [21]: the standard *autoencoder* model trained by stochastic gradient descent, where all object views were concatenated as one set of inputs. The number of hidden nodes was fixed at 500 on the MNIST dataset and at 1000 on the ETH-80 dataset. The learning rate, weight decay penalty, and number of iterations were empirically determined at 0.1, $3 \times 10^{-4}$, and 100, respectively.

- **DAE** [225]: the *denoising autoencoder* with zero-masking noise, where all object views were concatenated as one set of input data. The corruption level was fixed at 30% for all cases. Other hyper-parameter values were identical to AE.

- **CAE** [178]: the autoencoder model with the Jacobian matrix norm regularization referred to as the *contractive autoencoder*. The corresponding regularization constant $\lambda$ was set at 0.1.

- **uDICA**: the unsupervised Domain-Invariant Component Analysis [152], a kernel feature projection algorithm . The tunable hyper-parameters are the kernel width $\sigma$ for the Gaussian RBF kernel and the number of subspace bases $s$. We grid-search $\sigma \in \{10^{-3}, 10^{-2}, \dots, 10^{2}, 10^{3}\}$ and $s \in \{50, 100, 150, \dots, 500\}$ according the validation performance on source domains.

- **MTAE**: our proposed multi-task autoencoder with identical learning settings as AE, except for the learning rate set at 0.03, which was also chosen empirically. This value provides a lower reconstruction error for each task and visually clearer first layer weights.

- **D-MTAE**: MTAE with a denoising criterion. The learning rate was set the same as MTAE; other hyper-parameters followed DAE.

We also did experiments using DICA, the supervised variant of uDICA, where the Dirac kernel is used as the label similarity function. Surprisingly, the peak performance of uDICA is consistently higher than DICA. A possible explanation is that the Dirac kernel function measuring the label similarity is less appropriate in this task. So we only include uDICA in our comparisons.

We normalized the raw pixels to a range of $[0, 1]$ for autoencoder-based models and $l_2$-unit ball for uDICA. We evaluated the classification accuracies of the learned features using multi-class SVM with linear kernel (L-SVM) [49]. Using a linear kernel keeps the classifier simple – since our main focus is on the feature extraction process. The LIBLINEAR package [63] was used to run the L-SVM.

### 5.3.3 Cross-domain Recognition Results

We evaluated the object classification accuracies of each algorithm by *leave-one-domain-out* test, i.e., taking one domain as the test set and the remaining domains as the training set. For all autoencoder-based algorithms, we repeated the experiments on each *leave-one-domain-out* case 30 times and reported the average accuracies. The standard deviations are not reported since they are small ($\pm 0.1$), and also for presentation convenience.

The detailed results on the MNIST-r and MNIST-s can be seen in Table 5.1. On average, MTAE has the second best classification accuracies, and in particular outperforms single-task autoencoder models. This indicates that the multi-task feature learning strategy can provide better discriminative features than the single-task feature learning with respect to unseen views. The algorithm with the best performance is on these datasets is D-MTAE. Specifically, D-MTAE performs best on average and also on 9 out of 11 individual cross-domain cases of the MNIST-r and MNIST-s. The closest single-task feature learning competitor to D-MTAE is CAE. This suggests that the denoising criterion strongly benefits domain generalization. The denoising criterion is also useful for single-task feature learning although it does not yield competitive accuracies, see AE and DAE performance.

Observe that there is an anomaly in the MNIST-r dataset: the performance on $M_{45^\circ}$ is worse than its neighbors ($M_{30^\circ}, M_{60^\circ}$). This anomaly appears to be related to the geometry of the MNIST-r digits. We found that the most frequently misclassified digits are 4, 6, and 9 on $M_{45^\circ}$, which rarely occurs on other MNIST-r's domains – typically 4 as 9, 6 as 4, and 9 as 8. The same phenomenon applies to L-SVM.

We also obtain a consistent trend on the ETH80-p and ETH80-y datasets, *i.e.*, D-MTAE and MTAE are the best and second best models on average. Table 5.2 summarizes the complete evaluation results on those datasets. The perfect accuracy is obtained on recognizing $E_{-y45}$, which indicates that dataset bias is non-existence in that case. Note that D-MTAE does not provide good performance on $E_{p90}$, which can be attributed to overfitting – the model may be too complex compared to the number of ETH80 training samples.

Table 5.1: The *leave-one-**domain**-out* classification accuracies % on the MNIST-r and MNIST-s. Bold-red and bold-black indicate the best and second best performance.

| Source | Target | Raw | AE | DAE | CAE | uDICA | **MTAE** | **D-MTAE** |
|---|---|---|---|---|---|---|---|---|
| | | | | MNIST-r leave-one-**roll-rotation**-out | | | | |
| $M_{15^\circ}$, $M_{30^\circ}$, $M_{45^\circ}$, $M_{60^\circ}$, $M_{75^\circ}$ | $M$ | $52.40 \pm 0.00$ | $74.20 \pm 0.02$ | $76.90 \pm 0.02$ | $72.10 \pm 0.04$ | $67.20 \pm 0.00$ | **$77.90 \pm 0.05$** | **$82.50 \pm 0.09$** |
| $M$, $M_{30^\circ}$, $M_{45^\circ}$, $M_{60^\circ}$, $M_{75^\circ}$ | $M_{15^\circ}$ | $74.10 \pm 0.00$ | $93.20 \pm 0.05$ | $93.20 \pm 0.03$ | $95.30 \pm 0.09$ | $87.80 \pm 0.00$ | **$95.70 \pm 0.06$** | **$96.30 \pm 0.01$** |
| $M$, $M_{15^\circ}$, $M_{45^\circ}$, $M_{60^\circ}$, $M_{75^\circ}$ | $M_{30^\circ}$ | $71.40 \pm 0.00$ | $89.90 \pm 0.10$ | $91.30 \pm 0.07$ | **$92.60 \pm 0.03$** | $88.80 \pm 0.00$ | $91.20 \pm 0.10$ | **$93.40 \pm 0.02$** |
| $M$, $M_{15^\circ}$, $M_{30^\circ}$, $M_{60^\circ}$, $M_{75^\circ}$ | $M_{45^\circ}$ | $61.40 \pm 0.00$ | **$82.20 \pm 0.05$** | $81.10 \pm 0.10$ | **$81.50 \pm 0.07$** | $77.80 \pm 0.00$ | $77.30 \pm 0.09$ | $78.60 \pm 0.03$ |
| $M$, $M_{15^\circ}$, $M_{30^\circ}$, $M_{45^\circ}$, $M_{75^\circ}$ | $M_{60^\circ}$ | $67.40 \pm 0.00$ | $90.00 \pm 0.09$ | **$92.80 \pm 0.04$** | $92.70 \pm 0.10$ | $84.20 \pm 0.00$ | $92.40 \pm 0.05$ | **$94.20 \pm 0.03$** |
| $M$, $M_{15^\circ}$, $M_{30^\circ}$, $M_{45^\circ}$, $M_{60^\circ}$ | $M_{75^\circ}$ | $55.40 \pm 0.00$ | $73.80 \pm 0.09$ | $76.50 \pm 0.06$ | $79.30 \pm 0.10$ | $69.50 \pm 0.00$ | **$79.90 \pm 0.06$** | **$80.50 \pm 0.10$** |
| Average | | $63.68 \pm 0.00$ | $83.88 \pm 0.07$ | $85.30 \pm 0.05$ | $85.58 \pm 0.07$ | $79.22 \pm 0.00$ | **$85.73 \pm 0.07$** | **$87.58 \pm 0.05$** |
| | | | | MNIST-s leave-one-**dilation**-out | | | | |
| $M_{*0.9}$, $M_{*0.8}$, $M_{*0.7}$, $M_{*0.6}$ | $M$ | $54.00 \pm 0.00$ | $67.50 \pm 0.05$ | $71.80 \pm 0.02$ | **$75.80 \pm 0.01$** | **$75.80 \pm 0.01$** | $74.50 \pm 0.02$ | **$76.00 \pm 0.01$** |
| $M$, $M_{*0.8}$, $M_{*0.7}$, $M_{*0.6}$ | $M_{*0.9}$ | $80.40 \pm 0.00$ | $95.10 \pm 0.04$ | $94.00 \pm 0.01$ | $94.90 \pm 0.03$ | $88.60 \pm 0.00$ | **$97.80 \pm 0.03$** | **$98.00 \pm 0.08$** |
| $M$, $M_{*0.9}$, $M_{*0.7}$, $M_{*0.6}$ | $M_{*0.8}$ | $82.60 \pm 0.00$ | $94.60 \pm 0.07$ | $92.90 \pm 0.10$ | $94.90 \pm 0.10$ | $86.60 \pm 0.00$ | **$96.30 \pm 0.05$** | **$96.40 \pm 0.02$** |
| $M$, $M_{*0.9}$, $M_{*0.8}$, $M_{*0.6}$ | $M_{*0.7}$ | $78.20 \pm 0.00$ | $93.70 \pm 0.05$ | $91.60 \pm 0.10$ | $92.50 \pm 0.04$ | $87.40 \pm 0.00$ | **$95.80 \pm 0.06$** | **$94.90 \pm 0.05$** |
| $M$, $M_{*0.9}$, $M_{*0.8}$, $M_{*0.7}$ | $M_{*0.6}$ | $64.70 \pm 0.00$ | $74.80 \pm 0.03$ | $76.10 \pm 0.08$ | $77.50 \pm 0.07$ | $75.30 \pm 0.00$ | **$78.00 \pm 0.04$** | **$78.30 \pm 0.06$** |
| Average | | $71.98 \pm 0.00$ | $85.14 \pm 0.05$ | $85.28 \pm 0.06$ | $87.12 \pm 0.05$ | $82.74 \pm 0.00$ | **$88.48 \pm 0.04$** | **$88.72 \pm 0.04$** |

Table 5.2: The *leave-one-**domain**-out* classification accuracies % on the ETH80-p and ETH80-y.

| Source | Target | AE | DAE | CAE | **MTAE** | **D-MTAE** |
|---|---|---|---|---|---|---|
| ETH80-p leave-one-**pitch**-out | | | | | | |
| $E_{p22}, E_{p45}, E_{p68}, E_{p90}$ | $E_{p0}$ | $70.00 \pm 0.07$ | $73.73 \pm 0.08$ | $\mathbf{74.50 \pm 0.00}$ | $73.76 \pm 0.05$ | $\mathbf{77.50 \pm 0.00}$ |
| $E_{p0}, E_{p45}, E_{p68}, E_{p90}$ | $E_{p22}$ | $86.25 \pm 0.00$ | $88.74 \pm 0.05$ | $88.50 \pm 0.00$ | $\mathbf{92.50 \pm 0.00}$ | $\mathbf{92.50 \pm 0.00}$ |
| $E_{p0}, E_{p22}, E_{p68}, E_{p90}$ | $E_{p45}$ | $92.51 \pm 0.05$ | $93.77 \pm 0.06$ | $93.49 \pm 0.05$ | $\mathbf{97.51 \pm 0.08}$ | $\mathbf{97.53 \pm 0.10}$ |
| $E_{p0}, E_{p22}, E_{p45}, E_{p90}$ | $E_{p68}$ | $95.01 \pm 0.05$ | $\mathbf{98.74 \pm 0.05}$ | $\mathbf{99.00 \pm 0.00}$ | $\mathbf{98.78 \pm 0.08}$ | $\mathbf{98.78 \pm 0.09}$ |
| $E_{p0}, E_{p22}, E_{p45}, E_{p68}$ | $E_{p90}$ | $75.00 \pm 0.00$ | $\mathbf{75.02 \pm 0.06}$ | $\mathbf{74.49 \pm 0.08}$ | $\mathbf{75.03 \pm 0.10}$ | $72.78 \pm 0.09$ |
| Average | | $83.75 \pm 0.03$ | $86.00 \pm 0.06$ | $86.00 \pm 0.03$ | $\mathbf{87.51 \pm 0.06}$ | $\mathbf{87.82 \pm 0.05}$ |
| ETH80-y leave-one-**yaw**-out | | | | | | |
| $E_{+y45}, E_{y0}, E_{-y45}, E_{-y90}$ | $E_{+y90}$ | $84.98 \pm 0.06$ | $\mathbf{94.97 \pm 0.09}$ | $91.20 \pm 0.01$ | $91.26 \pm 0.08$ | $\mathbf{92.50 \pm 0.00}$ |
| $E_{+y90}, E_{y0}, E_{-y45}, E_{-y90}$ | $E_{+y45}$ | $98.75 \pm 0.00$ | $98.75 \pm 0.00$ | $98.75 \pm 0.00$ | $98.75 \pm 0.00$ | $98.75 \pm 0.00$ |
| $E_{+y90}, E_{+y45}, E_{-y45}, E_{-y90}$ | $E_{y0}$ | $92.48 \pm 0.06$ | $93.73 \pm 0.08$ | $94.72 \pm 0.09$ | $\mathbf{96.25 \pm 0.00}$ | $\mathbf{97.50 \pm 0.00}$ |
| $E_{+y90}, E_{+y45}, E_{y0}, E_{-y90}$ | $E_{-y45}$ | $97.49 \pm 0.05$ | $98.75 \pm 0.00$ | $98.75 \pm 0.00$ | $\mathbf{100.00 \pm 0.00}$ | $\mathbf{100.00 \pm 0.00}$ |
| $E_{+y90}, E_{+y45}, E_{y0}, E_{-y45}$ | $E_{-y90}$ | $91.23 \pm 0.06$ | $94.96 \pm 0.09$ | $93.80 \pm 0.10$ | $\mathbf{96.25 \pm 0.00}$ | $\mathbf{96.25 \pm 0.00}$ |
| Average | | $92.99 \pm 0.05$ | $96.23 \pm 0.05$ | $95.44 \pm 0.06$ | $\mathbf{96.50 \pm 0.02}$ | $\mathbf{97.00 \pm 0.00}$ |

### 5.3.4 Weight Visualization

Useful insight is obtained from considering the qualitative outcome of the MTAE training by visualizing the first layer weights. Figures 5.3-5.5 depict the learned weights of the autoencoder-based models on the MNIST-r, MNIST-s, and ETH80-p datasets, respectively. From these figures, we can conclude that both MTAE and D-MTAE's weights form a filter that seem to capture the underlying transformation across the views. On the contrary, the AE's and DAE's weights only explain the *contents* of the objects such as, in the MNIST-r and MNIST-s cases, local blob detectors and stroke detectors [225]. This may be a reason that MTAE and D-MTAE features can provide better domain generalization than AE and DAE, since they implicitly capture the relationship among the source domains.

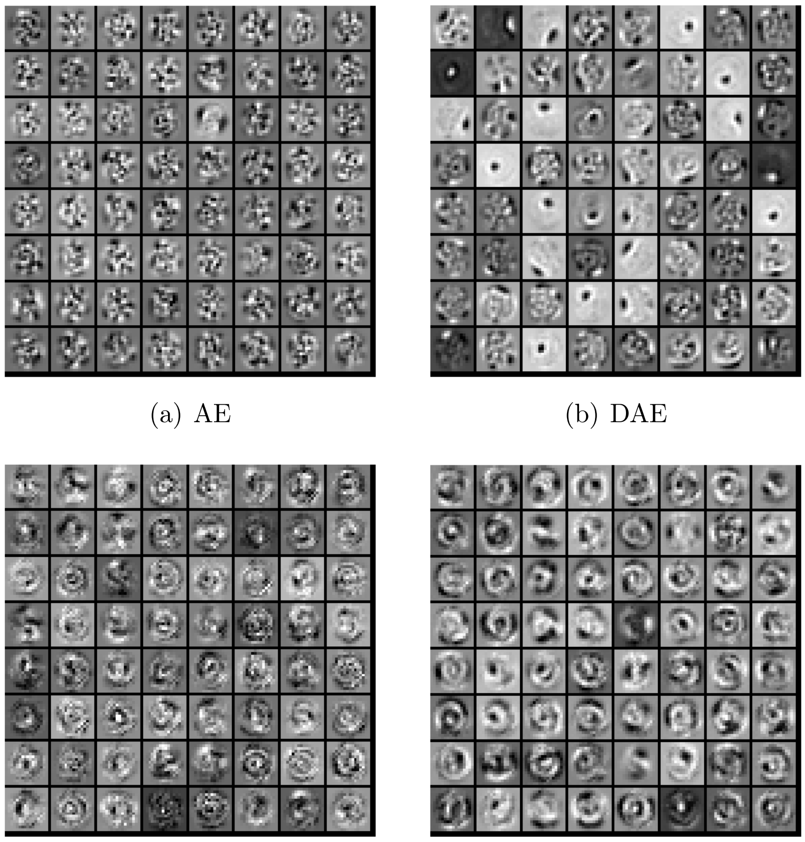
Next we discuss the difference between MTAE and D-MTAE filters. The D-MTAE filters not only capture the object transformation, but also produce features that describe the object contents more distinctively. These filters basically combine both properties of the DAE and MTAE filters that might further benefit the domain generalization.

### 5.3.5 Invariance Analysis

A possible explanation for the effectiveness of MTAE relates to the dimensionality of the manifold in feature space where samples concentrate. We hypothesize that if features concentrate near a low-dimensional submanifold, then the algorithm has found simple invariant features and will generalize well.

To test the hypothesis, we examine the singular value spectrum of the Jacobian matrix $\mathcal{J}_{\mathbf{x}}(\mathbf{z}) = \left[\frac{\partial z_i}{\partial x_j}\right]_{ij}$, where $\mathbf{x}$ and $\mathbf{z}$ are the input and feature vectors respectively [178]. The spectrum describes the local dimensionality of the manifold around which samples concentrate. If the spectrum decays rapidly, then the manifold is locally of low dimension.

Figure 5.6 depicts the average singular value spectrum on test samples from MNIST-r and MNIST-s. The spectrum of D-MTAE decays the most rapidly, followed by MTAE and then DAE (with similar rates), and AE decaying the slowest. The ranking of decay rates of the four algorithms matches their ranking in terms of empirical performance in Table 5.1. Figure 5.6 thus provides partial confirmation



(a) AE

(b) DAE

(c) MTAE

(d) D-MTAE

Figure 5.3: The 2D visualization of 100 randomly chosen weights after pre-training on the MNIST-r dataset. Each patch corresponds to a row of the learned weight matrix **W** that represents a “filter”.

for our hypothesis. However, a more detailed analysis is necessary before drawing strong conclusions.

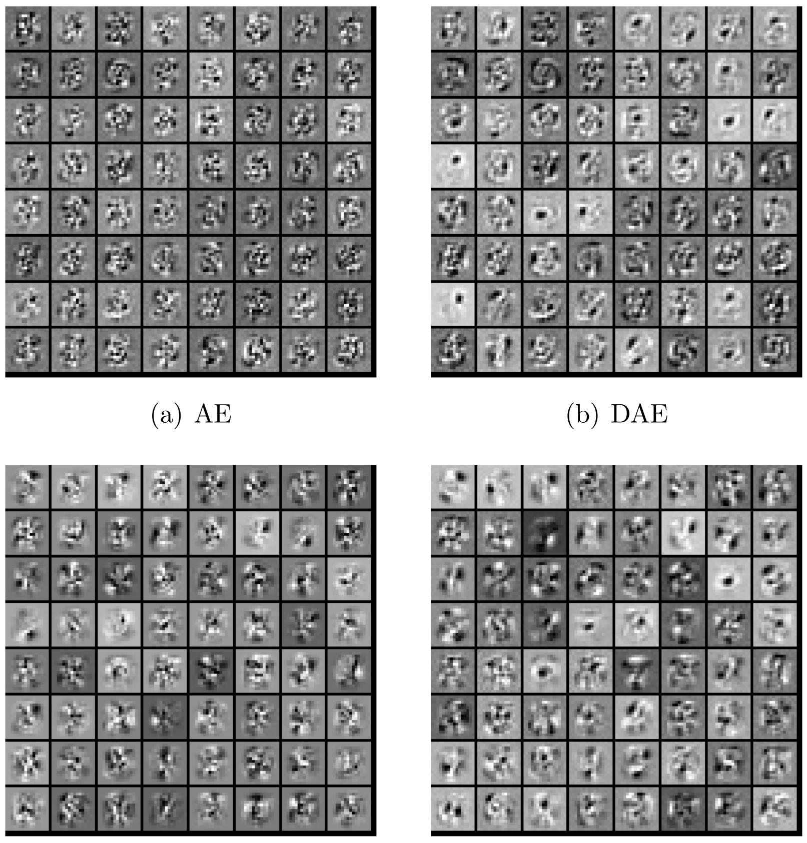


Figure 5.4: The 2D visualization of 100 randomly chosen weights after pre-training on the MNIST-s dataset.

## 5.4 Experiment 2: Cross-domain Recognition on Modern Benchmarks

In the second set of experiments, we evaluated the cross-recognition performance of the proposed algorithms on modern object datasets. The aim is to show that MTAE and D-MTAE are applicable and competitive in the more general setting. We used the Office, Caltech, PASCAL VOC2007, LabelMe, and SUN09 datasets from which we formed two cross-domain datasets. Our general strategy is to extend

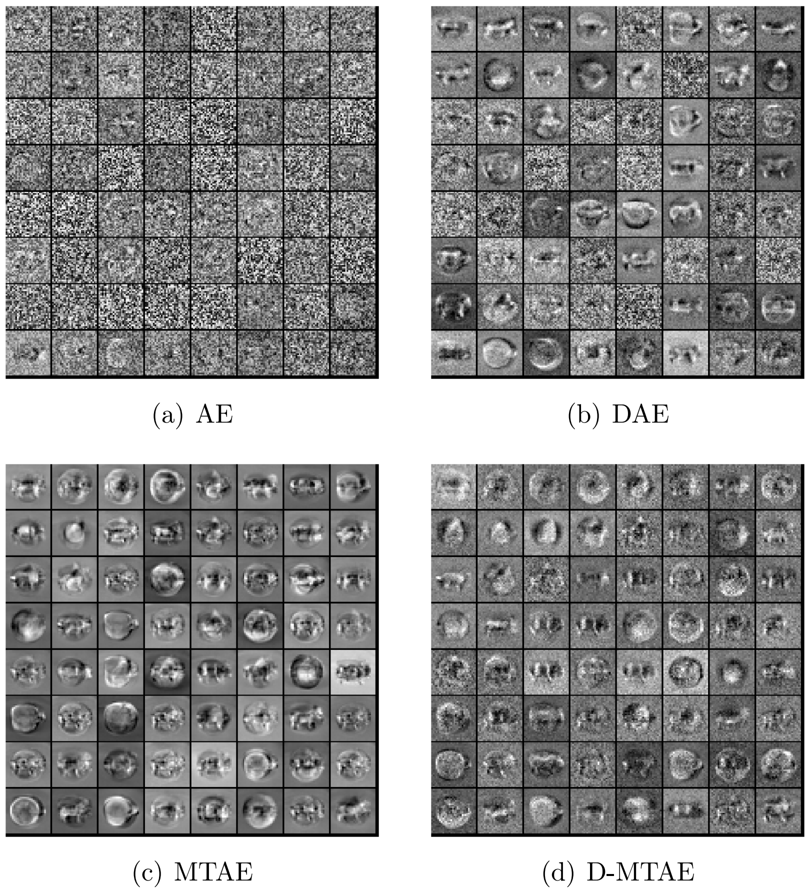


Figure 5.5: The 2D visualization of 100 randomly chosen weights after pre-training on the ETH80-p dataset.

the generalization of features extracted from the current best deep convolutional neural network [118].

### 5.4.1 Data Setup

The first cross-domain dataset consists of images from PASCAL VOC2007 (V), LabelMe (L), Caltech-101 (C), and SUN09 (S) datasets, each of which represents one domain. C is an object-centric dataset, while V, L, and S are scene-centric. This dataset, which we abbreviate as **VLCS**, shares five object categories: ‘bird’,

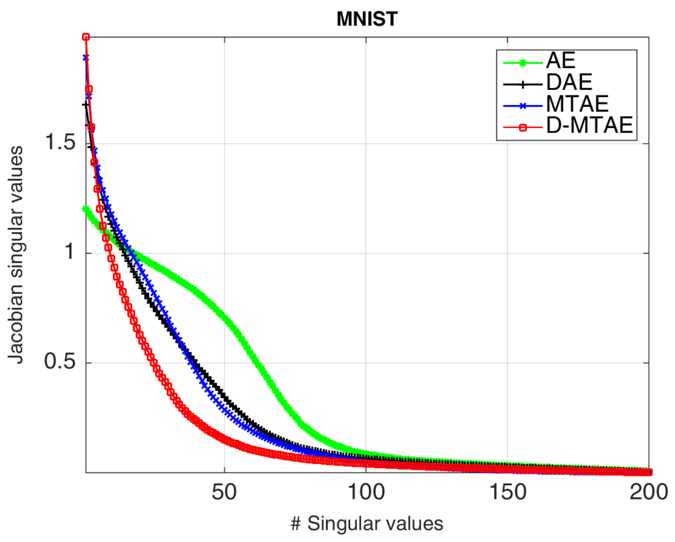


Figure 5.6: The average singular value spectrum of the Jacobian matrix over the MNIST-r and MNIST-s datasets.

'car', 'chair', 'dog', and 'person'. Each domain in the VLCS dataset was divided into a training set (70%) and a test set (30%) by random selection from the overall dataset. The detailed training-test configuration for each domain is summarized in Table 5.3. Instead of using the raw features directly, we employed the DeCAF$_6$ features [56] as inputs to the algorithms. These features have dimensionality of 4,096 and are publicly available.[3]

The second cross-domain dataset is referred to as the **Office+Caltech** [83,187] dataset that contains four domains: Amazon (A), Webcam (W), DSLR (D), and Caltech-256 (C), which share ten common categories. This dataset has 8 to 151 instances per category per domain, and 2,533 instances in total. We also used the DeCAF$_6$ features extracted from this dataset, which are also publicly available.[4]

## 5.4.2 Training Protocol

On these datasets, we utilized the MTAE or D-MTAE learning as *pretraining* for a fully-connected neural network with one hidden layer (1HNN). The number of

[3]`http://www.cs.dartmouth.edu/~chenfang/proj_page/FXR_iccv13/index.php`
[4]`http://vc.sce.ntu.edu.sg/transfer_learning_domain_adaptation/`

Table 5.3: Number of training and test instances for each domain in the VLCS dataset.

| Domain | VOC2007 | LabelMe | Caltech-101 | SUN09 |
|---|---|---|---|---|
| #training | 2,363 | 1,859 | 991 | 2,297 |
| #test | 1,013 | 797 | 424 | 985 |

hidden nodes was set at 2,000, which is less than the input dimensionality. In the pretraining stage, the number of output layers was the same as the number of source domains – each corresponds to a particular source domain. The sigmoid activation and linear activation functions were used for $\sigma_{\text{enc}}(\cdot)$ and $\sigma_{\text{dec}}(\cdot)$.

The MTAE pretraining was run with the learning rate at $5 \times 10^{-4}$, the number of epochs at 500, and the batch size at 10, which were empirically determined with respect to the smallest average reconstruction loss. D-MTAE has the same hyper-parameter setting as MTAE except for the additional zero-masking corruption level at 20%. After the pretraining is completed, we then performed backpropagation *fine-tuning* using 1HNN with softmax output, where the first layer weights were initialized by either the MTAE or D-MTAE learned weights. The supervised learning hyper-parameters were tuned using 10-fold cross validation (10FCV) on source domains. We denote the overall models by **MTAE+1HNN** and **D-MTAE+1HNN**.

### 5.4.3 Baselines

We compared our proposed models with six baselines:

- **L-SVM**: an SVM with linear kernel.
- **1HNN**: a single hidden layer neural network without pretraining.
- **DAE+1HNN**: a two-layer neural network with denoising autoencoder pretraining (DAE+1HNN).
- **Undo-Bias** [113]: a multi-task SVM-based algorithm for undoing dataset bias. Three hyper-parameters $(\lambda, C_1, C_2)$ require tuning by 10FCV.

- **UML** [64]: a structural metric learning-based algorithm that aims to learn a less biased distance metric for classification tasks. The initial tuning proposal for this method was using a set of weakly-labeled data retrieved from querying class labels to search engine. However, here we tuned the hyperparameters using the same strategy as others (10FCV) for a fair comparison.

- **LRE-SVM** [236]: a non-linear exemplar-SVMs model with a nuclear norm regularization to impose a low-rank *likelihood matrix*. Four hyper-parameters $(\lambda_1, \lambda_2, C_1, C_2)$ were tuned using 10FCV.

The last three are the state-of-the-art domain generalization algorithms for object recognition.

We report the performance in terms of the classification accuracy (%) following Xu et al. [236]. For all algorithms that are optimized stochastically, we ran independent training processes using the best performing hyper-parameters in 10 times and reported the average accuracies.

Table 5.4: The groundtruth L-SVM accuracies % on the standard training-test evaluation. The left-most column indicates the training set, while the upper-most row indicates the test set.

| Training/Test | VOC2007 | LabelMe | Caltech-101 | SUN09 |
|---|---|---|---|---|
| VOC2007 | **66.34** | 34.50 | 65.09 | 52.49 |
| LabelMe | 44.03 | **68.76** | 43.87 | 41.02 |
| Caltech-101 | 52.81 | 32.37 | **95.99** | 39.29 |
| SUN09 | 52.42 | 42.03 | 40.33 | **74.21** |

Table 5.5: The cross-recognition accuracy % on the VLCS dataset.

| Algorithms | L,C,S → V | V,C,S → L | V,L,S → C | V,L,C → S | Avg. |
|---|---|---|---|---|---|
| L-SVM | $58.86 \pm 0.00$ | $52.49 \pm 0.00$ | $77.67 \pm 0.00$ | $49.09 \pm 0.00$ | $59.93 \pm 0.00$ |
| 1HNN | $59.10 \pm 0.09$ | $58.20 \pm 0.04$ | $86.87 \pm 0.12$ | $57.86 \pm 0.15$ | $65.46 \pm 0.10$ |
| DAE+1HNN | $\mathbf{62.00 \pm 0.11}$ | $59.23 \pm 0.00$ | $87.50 \pm 0.09$ | $54.12 \pm 0.08$ | $65.75 \pm 0.07$ |
| Undo-Bias | $54.29 \pm 0.00$ | $58.09 \pm 0.00$ | $87.50 \pm 0.00$ | $54.21 \pm 0.00$ | $63.52 \pm 0.00$ |
| UML | $56.26 \pm 0.00$ | $58.50 \pm 0.00$ | $\mathbf{91.13 \pm 0.00}$ | $58.49 \pm 0.00$ | $65.85 \pm 0.00$ |
| LRE-SVM | $60.58 \pm 0.00$ | $\mathbf{59.74 \pm 0.00}$ | $88.11 \pm 0.00$ | $54.88 \pm 0.00$ | $65.83 \pm 0.00$ |
| **MTAE+1HNN** | $61.09 \pm 0.02$ | $\mathbf{59.24 \pm 0.00}$ | $\mathbf{90.17 \pm 0.08}$ | $\mathbf{60.20 \pm 0.06}$ | $\mathbf{67.81 \pm 0.04}$ |
| **D-MTAE+1HNN** | $\mathbf{63.90 \pm 0.07}$ | $\mathbf{60.13 \pm 0.00}$ | $\mathbf{89.05 \pm 0.06}$ | $\mathbf{61.33 \pm 0.08}$ | $\mathbf{68.60 \pm 0.05}$ |

### 5.4.4 Results on the VLCS Dataset

We first conducted the standard training-test evaluation using L-SVM, *i.e.*, learning the model on a training set from one domain and testing it on a test set from another domain, to check the groundtruth performance and also to identify the existence of the dataset bias. The performance is summarized in Table 5.4. We can see that the bias indeed exists in every domain despite the use of DeCAF$_6$, the sixth layer features of the state-of-the-art deep convolutional neural network. The performance gap between the best cross-domain performance and the groundtruth is large, with $\geq 14\%$ difference.

We then evaluated the domain generalization performance of each algorithm. We conducted *leave-one-domain-out* evaluation, which induces four cross-domain cases. Table 5.5 summarizes the algorithms' accuracy. In general, the dataset bias can be reduced by all algorithms after learning from multiple source domains (compare, e.g., the minimum accuracy over L,C,S $\rightarrow$ V tasks in Table 5.5 with the maximum cross-recognition accuracy over the VOC2007's column in Table 5.4). Caltech-101, which is object-centric, appears to be the easiest dataset to recognize, consistent with an investigation in [215]: scene-centric datasets tend to generalize well over object-centric datasets. Note that the performance of 1HNN has already achieved competitive accuracy compared to more complicated state-of-the-art algorithms, Undo-Bias, UML, and LRE-SVM. This suggests that it is not necessary anymore to apply complex domain generalization algorithms if operating on DeCAF$_6$ features. Furthermore, D-MTAE outperforms other algorithms on three out of four cross-domain cases and on average, while MTAE has the second best performance on average.

### 5.4.5 Results on the Office+Caltech Dataset

We report the experiment results on the Office+Caltech dataset. Table 7.6 summarizes the recognition accuracies of each algorithm over four cross-domain cases. D-MTAE+1HNN has the best performance on two out of four cross-domain cases and ranks second for the remaining cases. On average, D-MTAE+1HNN has better performance than the prior state-of-the-art on this dataset, LRE-SVM [236].

Table 5.6: The cross-recognition accuracy % on the Office+Caltech dataset.

| Algorithms | A,C → D,W | D,W → A,C | C,D,W → A | A,W,D → C | Avg. |
|---|---|---|---|---|---|
| L-SVM | 82.08 ± 0.00 | 76.12 ± 0.00 | 90.61 ± 0.00 | 84.51 ± 0.00 | 83.33 ± 0.00 |
| 1HNN | 83.41 ± 0.15 | 76.49 ± 0.11 | 92.13 ± 0.08 | 85.89 ± 0.03 | 84.48 ± 0.09 |
| DAE+1HNN | 82.05 ± 0.19 | 79.04 ± 0.09 | 92.02 ± 0.07 | 85.17 ± 0.04 | 84.57 ± 0.10 |
| Undo-Bias | 80.49 ± 0.00 | 69.98 ± 0.00 | 90.98 ± 0.00 | 85.95 ± 0.00 | 81.85 ± 0.00 |
| UML | 82.29 ± 0.00 | 79.54 ± 0.00 | 91.02 ± 0.00 | 84.59 ± 0.00 | 84.36 ± 0.00 |
| LRE-SVM | **84.59 ± 0.00** | **81.17 ± 0.00** | 91.87 ± 0.00 | **86.38 ± 0.00** | **86.00 ± 0.00** |
| **MTAE+1HNN** | 84.23 ± 0.11 | 79.30 ± 0.10 | **92.20 ± 0.04** | 85.98 ± 0.09 | 85.43 ± 0.09 |
| **D-MTAE+1HNN** | **85.35 ± 0.20** | **80.52 ± 0.15** | **93.13 ± 0.05** | **86.15 ± 0.08** | **86.29 ± 0.12** |

## 5.5 Chapter Summary

This chapter presents a new approach to multi-task representation learning that reduces *dataset bias* in visual object recognition. The main idea is to extract features shared across domains via a training protocol that, given an image from one domain, learns to reconstruct analogs of that image for all domains. The strategy yields two variants: the Multi-task Autoencoder (MTAE) and the Denoising MTAE (D-MTAE) which incorporates a denoising criterion using zero-masking noise. The algorithms were evaluated comprehensively over a wide range of cross-domain tasks.

Both MTAE and D-MTAE provides better domain generalization performance than existing single-task autoencoder-based models (AE, DAE, and CAE), especially on raw pixels of the transformed MNIST and ETH-80 images. This shows that the new algorithms successfully learn view-invariant features on those datasets. An observation of the visualization of the learned weights indicates that MTAE captures the underlying object transformation among the MNIST and ETH-80 domains. The learned weights of AE, DAE, or CAE appear to be different from that of MTAE, which only highlight either local or global *contents* of the objects. Furthermore, the learned weights of D-MTAE seem to combine both properties of DAE and MTAE.

A spectral analysis of the Jacobian matrix of the features with respect to the inputs shows that MTAE and D-MTAE do a better job of characterizing a low-dimensional submanifold near which samples concentrate. This is indicated by a few large singular values of MTAE and D-MTAE encoder's Jacobian compared to that of AE and DAE. Such an outcome suggests that our algorithms have found simple invariant features and will then provide good generalization.

On the VLCS and Office+Caltech datasets, MTAE and D-MTAE, which act as pretraining models for a neural network, provides competitive performance compared to prior state-of-the-art algorithms. In particular, D-MTAE achieves the best average accuracies on both datasets. MTAE performs second best on the VLCS and comes third on the Office+Caltech with a small performance gap compared to the second best model, LRE-SVM.

Our results suggest several directions for further study. Firstly, it is worth investigating whether stacking MTAEs improves performance. Secondly, more effective procedures for handling unbalanced samples are required, since these occur frequently in practice. Thirdly, the effectiveness of MTAE and D-MTAE for visual object recognition may be improved by incorporating a convolutional architecture [141]. Finally, a natural application of MTAEs is to streaming data such as *video*, where the appearance of objects transforms in real-time.

The problem of dataset bias remains far from solved: the best model on the VLCS dataset achieved accuracies less than 70% on average. A partial explanation for the poor performance compared to supervised learning is insufficient training data: the class-overlap across datasets is quite small (only 5 classes are shared across VLCS). Further progress in domain generalization requires larger datasets.

The next contribution chapter will present Scatter Component Analysis (SCA), a unified kernel-based feature learning algorithm for both domain adaptation and domain generalization. SCA is fundamentally different from MTAE, which finds a domain-invariant representation via a linear projection by solving a generalized eigenproblem.

# 6 Scatter Component Analysis: A Unified Framework for Domain Adaptation and Domain Generalization

*This chapter presents the fourth contribution of this thesis. We develop a fast, unified algorithm to reduce dataset bias that is compatible to both domain adaptation and domain generalization, Scatter Component Analysis (SCA). The algorithm finds a representation that trades between maximizing the separability of classes, minimizing the mismatch between domains, and maximizing the separability of data; each of which can be quantified through a single measure. We performed extensive experiments to evaluate the performance of SCA against a large suite of alternatives. We found that SCA performs considerably faster than the prior state-of-the-art across a range of visual object cross-domain recognition, with competitive or better accuracy.*

## 6.1 Introduction

As discussed in Section 2.6.1, domain adaptation and domain generalization are two almost similar learning settings with only one difference: the presence of the unlabeled target samples during training. If the unlabeled target samples are available, one can utilize domain adaptation algorithms to deal with dataset bias; otherwise, domain generalization algorithms should be used. However, typical existing domain adaptation and domain generalization algorithms are generally *not compatible* to each other – domain adaptation methods cannot be directly applied to domain generalization or vice versa.

Another important issue is that prior state-of-the-art domain adaptation and domain generalization algorithms for object recognition result in optimization problems that are inefficient to solve [133, 134, 195, 236]. They may not be suitable in situations that require a real-time learning stage. Therefore, it is highly desirable to develop algorithms that can be computed more efficiently, are compatible with both domain adaptation and domain generalization, and provide state-of-the-art performance.

### 6.1.1 Chapter Goals

The overall goal of this chapter is to develop a fast representation learning algorithm to reduce dataset bias that can be applied to both domain adaptation and domain generalization settings. The learned representations should incorporate four requirements: (i) separate points with different labels and (ii) separate the data as a whole (high variance), whilst (iii) not separating points sharing a label and (iv) not separating the two or more domains.

To achieve the overall goal, this chapter establishes several specific objectives as follows:

- Whether the above requirements needed to learn the domain-invariant representations can be quantified using a single measure;
- Whether the new algorithm can produce a linearly separable representation and can be computed through an optimization that admits a fast and exact solution;

- Whether the new algorithm can provide the state-of-the-art performance in terms of accuracy on both domain adaptation and domain generalization tasks;

- Whether the candidate solution of the algorithm is theoretically guaranteed in the context of domain adaptation, *i.e.*, the *target* generalization error of the hypothesis is bounded.

### 6.1.2 Chapter Organization

This chapter is organized as follows. Section 6.2 presents a formal description of *scatter*, a measure that can quantify all requirements needed in the proposed representation learning algorithm. Section 6.3 describes the corresponding algorithm, which we refer to as *Scatter Component Analysis* (SCA). The theoretical domain adaptation bound for SCA is presented in Section 6.4. Comprehensive evaluation results and analyses are provided in Sections 6.5 and 6.6. Finally, Section 6.7 concludes the chapter.

## 6.2 Scatter

The algorithm proposed in this chapter operates on a feature space referred to as a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$, see Section 2.5.1 for the theoretical background of the RKHS. The main motivation is to transform original inputs onto $\mathcal{H}$, which is a high or possibly infinite dimensional space, with the hope that the new features are linearly separable. The most important property of RKHS is perhaps to allow a computationally feasible transformation onto $\mathcal{H}$ by virtue of the *kernel trick*.

Before introducing scatter, it is convenient to first represent domains as points in RKHS using the mean map [199]:

**Definition 10 (Mean map).** *Suppose that $\mathcal{X}$ is equipped with a kernel, and that $\mathcal{H}$ is the corresponding RKHS with feature map $\phi : \mathcal{X} \to \mathcal{H}$. Let $\Delta_{\mathcal{X}}$ denote the set of probability distributions on $\mathcal{X}$. The mean map takes distributions on $\mathcal{X}$ to* *points in $\mathcal{H}$:*

$$\mu : \Delta_{\mathcal{X}} \to \mathcal{H} : \mathbb{P} \mapsto \mathop{\mathbb{E}}_{x\sim\mathbb{P}}\Big[\phi(x)\Big] =: \mu_{\mathbb{P}}.$$

Geometrically, the mean map is the centroid of the image of the distribution under $\phi$. We define *scatter* as the variance of points in the image around its centroid:

**Definition 11** **(Scatter).** *The* ***scatter*** *of distribution $\mathbb{P}$ on $\mathcal{X}$ relative to $\phi$ is*

$$\Psi_{\phi}(\mathbb{P}) := \mathop{\mathbb{E}}_{x\sim\mathbb{P}}\left[\Big\|\mu_{\mathbb{P}} - \phi(x)\Big\|_{\mathcal{H}}^2\right]$$

*where $\|\cdot\|_{\mathcal{H}}$ is the norm on $\mathcal{H}$.*

The scatter of a domain cannot be computed directly; instead it is estimated from observations. The scatter of a finite set of observations $\{x_1,\dots,x_n\}$ is computed with respect to the empirical distribution

$$\widehat{\mathbb{P}}(x) := \frac{1}{n}\sum_{i=1}^{n}\delta_{x_i}(x) \quad \text{where} \quad \delta_{x_i}(x) = \begin{cases} 1 & \text{if } x_i = x \\ 0 & \text{else.}\end{cases}$$

We provide a theorem that shows how the difference between the true scatter and a finite sample estimate decreases with the sample size. To do so, we need a concentration of measure bound referred to as McDiarmid's inequality [143].

**Theorem 10** **(McDiarmid's Inequality).** *Let $X_1,\dots,X_n$ be independent random variables taking values in the set $\mathcal{X}$ under a distribution $\mathbb{P}$. Further, let $f : \mathcal{X}^n \to \mathbb{R}$ be a function of $X_1,..,X_n$ that satisfies*

$$\sup_{x_1\dots x_n, x_i'\in\mathcal{X}}\Big|f(x_1\dots x_i\dots x_n) - f(x_1\dots x_i'\dots x_n)\Big| \le c_i,$$

*where $x_i \neq x_i'$ and $1 \le i \le n$. The following inequality holds for all $\epsilon > 0$*

$$\mathbb{P}\Big\{\Big|\mathbb{E}[f] - f\Big| \ge \epsilon\Big\} \le 2\exp\left(\frac{-2\epsilon^2}{\sum_{i=1}^{n} c_i^2}\right).$$

Now it is convenient to bound the difference between the *empirical* scatter and the *true* scatter given by the following theorem:

**Theorem 11** (**Scatter Bound**)**.** *Suppose $\mathbb{P}$ is a true distribution over all samples of size $n$ and $\hat{\mathbb{P}}$ is its empirical distribution. Further suppose that $\|\phi(x)\|^2 \leq M$ for all $x \in \mathcal{X}$. Then, with probability $\geq 1 - \delta$,*

$$\left|\Psi_\phi(\mathbb{P}) - \Psi_\phi(\hat{\mathbb{P}})\right| \leq M\sqrt{\frac{2\log(\frac{2}{\delta})}{n}}.$$

*Proof.* Let $S = \{x_1, \dots, x_{i-1}, x_i, x_{i+1}, \dots, x_n\}$ and $\tilde{S} = \{x_1, \dots, x_{i-1}, \tilde{x}_i, x_{i+1}, \dots, x_n\}$ be two samples with one point difference only. Using Triangle Inequality and the fact that $\|\phi(x)\|^2_{\mathcal{H}} \leq M$, the following statement holds

$$\sup_{x_1 \dots x_n, \tilde{x}_i \in \mathcal{X}} |\Psi_\phi(S) - \Psi_\phi(S')| \leq \sup_{x_1 \dots x_n, \tilde{x}_i \in \mathcal{X}} |\Psi_\phi(S)| + |\Psi_\phi(S')|$$
$$= \sup_{\tilde{x}_i \in \mathcal{X}} \frac{1}{n}\|\mu_{\hat{\mathbb{P}}} - \phi(x_i)\|^2_{\mathcal{H}} + \frac{1}{n}\|\mu_{\hat{\mathbb{P}}} - \phi(\tilde{x}_i)\|^2_{\mathcal{H}} \leq \frac{2M}{n}.$$

By McDiarmid's inequality, for all $\epsilon > 0$ and $c_i = \frac{2M}{n}$

$$\mathbb{P}\left\{\left|\Psi_\phi(\mathbb{P}) - \Psi_\phi(\hat{\mathbb{P}})\right| > \epsilon\right\} \leq 2\exp\left(-\frac{\epsilon^2 \cdot n}{2M^2}\right).$$

Setting $\delta = 2\exp\left(-\frac{\epsilon^2 \cdot n}{2M^2}\right)$, the results follows directly. □

We provide an example for later use. If the input space is a vector space and $\phi$ is the identity then it follows immediately that

**Lemma 12** (**Total variance as scatter**)**.** *The scatter of the set of $d$-dimensional points (in a matrix form) $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]^\top \in \mathbb{R}^{n \times d}$ relative to the identity map $\phi : \mathbb{R}^d \to \mathbb{R}^d$, i.e., $\phi(\mathbf{x}) := \mathbf{x}$, is the total variance:*

$$\Psi(\mathbf{X}) = \mathrm{Tr}(\mathbf{X} - \bar{\mathbf{X}})^\top(\mathbf{X} - \bar{\mathbf{X}}) = \mathrm{Tr}\,\mathrm{Cov}(\mathbf{X}),$$

*where* $\mathrm{Tr}(\cdot)$ *denotes the trace operation and* $\bar{\mathbf{X}} = [\bar{\mathbf{x}}, \dots, \bar{\mathbf{x}}]^\top$ *with* $\bar{\mathbf{x}} = \sum_{i=1}^n \mathbf{x}_i$.

We utilize *scatter* to formulate a feature learning algorithm referred to as Scatter Component Analysis (SCA). Specifically, *scatter* quantifies requirements needed in SCA to develop an effective solution for both domain adaptation and generalization, which will be described in the next section.

## 6.3 Scatter Component Analysis (SCA)

SCA aims to convert the observations into a configuration of points in feature space such that the domain mismatch is reduced. SCA then finds a representation of the problem (that is, a linear transformation of feature space) for which (i) the source and target domains are similar and (ii) elements with the same label are similar; whereas (iii) elements with different labels are well separated and (iv) the variance of the whole data is maximized. Each requirement can be quantified through *scatter* that leads to four consequences: (i) *domain scatter*, (ii) *between-class scatter*, (iii) *within-class scatter*, and (iv) *total scatter.*

The remainder of the subsection defines the above four scatter quantities in more detail (along the way relating the terms to principal component analysis, the maximum mean discrepancy, and Fisher's linear discriminant) and describes the SCA's learning algorithm. We will also see that SCA can be easily switched to either domain adaptation or domain generalization by modifying the configuration of the input domains.

### 6.3.1 Total Scatter

Given $m$ domains $\mathbb{P}_X^1, \ldots, \mathbb{P}_X^m$ on $\mathcal{X}$, we define the total scatter as the average over the domains $\bar{\mathbb{P}}_X = \frac{1}{m}\sum_{d=1}^m \mathbb{P}_X^d$. The *total scatter* is then defined by

$$\text{total scatter } = \Psi_\phi\left(\bar{\mathbb{P}}_X\right). \tag{6.1}$$

It is worth emphasizing that this definition is general in the sense that it covers both domain adaptation ($m = 2$ and one of them is the target domain) and domain generalization ($m > 2$).

Total scatter is estimated from data as follows. Let $\mathbf{X} = [\mathbf{x}_1, ..., \mathbf{x}_n]^\top \in \mathbb{R}^{n\times p}$ be the matrix of unlabeled samples from all $m$ domains ($n = \sum_{d=1}^m n_d$, where $n_d$ is the number of examples in the $d$-th domain). Given a feature map $\phi : \mathbb{R}^p \to \mathcal{H}$ corresponding to kernel $\kappa$, define a set of functions arranged in a column vector $\mathbf{\Phi} = [\phi(\mathbf{x}_1), ..., \phi(\mathbf{x}_n)]^\top$. After centering $\{\phi(\mathbf{x}_i)\}_{i=1}^n$ by subtracting the mean, the covariance matrix is $\mathrm{Cov}(\mathbf{\Phi}) = \mathbf{\Phi}^\top\mathbf{\Phi}$. By Lemma 12,

$$\Psi_\phi\left(\hat{\hat{\mathbb{P}}}_X\right) = \mathrm{Tr}\,\mathrm{Cov}(\mathbf{\Phi}). \tag{6.2}$$

We are interested in performing dimensionality reduction on the input data to select only relevant information, that is, by applying a linear transform $\mathbf{W}$ to a finite subspace $\mathbb{R}^k$. In this context, the transformed data or features should have a property that the total scatter is maximized. Suppose that the inputs are first mapped to a reproducing kernel Hilbert space $\mathcal{H}$ to encourage linear separability. To avoid the direct computation of $\phi : \mathcal{X} \to \mathcal{H}$, which could be expensive or undoable, we use the *kernel trick*. Let $\mathbf{Z} = \mathbf{\Phi W} \in \mathbb{R}^{n\times k}$ be the $n$ transformed feature vectors and $[\mathbf{K}]_{ij} = [\mathbf{\Phi\Phi}^\top]_{ij} = [\kappa(\mathbf{x}_i, \mathbf{x}_j)]$. After fixing $\mathbf{B} \in \mathbb{R}^{n\times k}$ such that $\mathbf{W} = \mathbf{\Phi}^\top\mathbf{B}$, the total transformed scatter is

$$\Psi_{\mathbf{B}\circ\phi}\left(\hat{\hat{\mathbb{P}}}_X\right) = \mathrm{Tr}(\underbrace{\mathbf{B}^\top\mathbf{KKB}}_{\mathrm{Cov}(\mathbf{Z})}). \tag{6.3}$$

We remark that, in our notation, Kernel Principal Component Analysis (KPCA) [192] corresponds to the optimization problem

$$\max \Psi_{\mathbf{B}\circ\phi}\left(\hat{\hat{\mathbb{P}}}_X\right) \text{ s.t. } \mathbf{B}^\top\mathbf{KB} = \mathbf{I}. \tag{6.4}$$

### 6.3.2 Domain Scatter

Suppose we are given $m$ domains $\mathbb{P}_X^1, \ldots, \mathbb{P}_X^m$ on $\mathcal{X}$. We can think of the *set* $\{\mu_{\mathbb{P}_X^1}, \ldots, \mu_{\mathbb{P}_X^m}\} \subset \mathcal{H}$ as a sample from some latent distribution on domains. Equipping the sample with the empirical distribution and computing scatter relative to the identity map on $\mathcal{H}$ yields *domain scatter*:

$$\Psi\Big(\{\mu_{\mathbb{P}_X^1}, \ldots, \mu_{\mathbb{P}_X^m}\}\Big) = \frac{1}{m}\sum_{i=1}^m \left\|\bar{\mu} - \mu_{\mathbb{P}^i}\right\|^2, \tag{6.5}$$

where $\bar{\mu} = \frac{1}{m}\sum_{i=1}^m \mu_{\mathbb{P}^i}$. Note that domain scatter coincides with the *distributional variance* introduced in [152]. Domain scatter is also essentially equivalent to the Maximum Mean Discrepancy (MMD), used in some domain adaptation algorithms [104, 135, 166].

**Definition 12.** *Let $\mathcal{F}$ be a set of functions $f : \mathcal{X} \to \mathbb{R}$. The* ***maximum mean discrepancy*** *between domains $\mathbb{P}$ and $\mathbb{Q}$ is*

$$MMD_{\mathcal{F}}[\mathbb{P}, \mathbb{Q}] := \sup_{f \in \mathcal{F}} \left( \underset{\mathbb{P}}{\mathbb{E}}[f(x)] - \underset{\mathbb{Q}}{\mathbb{E}}[f(x)] \right).$$

The MMD measures the extent to which two domains resemble one another from the perspective of function class $\mathcal{F}$. The following theorem relates domain scatter to MMD given two domains, where the case of interest is bounded linear functions on the feature space:

**Theorem 13** (**Scatter recovers MMD**)**.** *The scatter of domains $\mathbb{P}$ and $\mathbb{Q}$ on $\mathcal{X}$ is their (squared) maximum mean discrepancy:*

$$\Psi(\{\mu_{\mathbb{P}}, \mu_{\mathbb{Q}}\}) = \frac{1}{4} MMD^2_{\mathcal{F}}[\mathbb{P}, \mathbb{Q}],$$

*where $\mathcal{F} = \{f : \mathcal{X} \to \mathbb{R} \,|\, f$ is linear and $\|f\|_{\mathcal{F}} \leq 1\}$.*

*In particular, if $\phi$ is induced by a characteristic kernel on $\mathcal{X}$ then $\Psi(\{\mu_{\mathbb{P}}, \mu_{\mathbb{Q}}\}) = 0$ if and only if $\mathbb{P} = \mathbb{Q}$.*

*Proof.* Note that the theorem involves two levels of probability distributions: (i) the domains $\mathbb{P}$ and $\mathbb{Q}$ on $\mathcal{X}$, and (ii) the empirical distribution on $\mathcal{F}$ that assigns probability $p = \frac{1}{2}$ to the points $\mu_{\mathbb{P}}$ and $\mu_{\mathbb{Q}}$, and $p = 0$ to everything else. Let $\bar{\mu} = \frac{1}{2}(\mu_{\mathbb{P}} + \mu_{\mathbb{Q}})$. Using (6.7),

$$\Psi(\{\mu_{\mathbb{P}}, \mu_{\mathbb{Q}}\}) = \frac{1}{2}\|\bar{\mu} - \mu_{\mathbb{P}}\|^2_{\mathcal{F}} + \frac{1}{2}\|\bar{\mu} - \mu_{\mathbb{Q}}\|^2_{\mathcal{F}} = \frac{1}{4}\|\mu_{\mathbb{P}} - \mu_{\mathbb{Q}}\|^2_{\mathcal{F}}.$$

The result follows from Theorem 2.2 of [28]. □

Theorem 13 also tells us that the domain scatter is a valid metric if the kernel on $\mathcal{X}$ is *characteristic*, that is, the mean map (see Definition 10) associated with the kernel is injective [201]. We also remark that MMD can be estimated from observed data with bound provided in [87], which is analogous to Theorem 11.

Domain scatter in a transformed feature space in $\mathbb{R}^k$ is estimated as follows. Suppose we have $m$ samples $S^d_u = \{\mathbf{x}^d_i\}_{i=1}^{n_d} \sim \mathbb{P}^d_X$. Recall that $\mathbf{Z} = \mathbf{\Phi W} =$

$\mathbf{K}^\top\mathbf{B}$, where $\mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_n]^\top$ contains projected samples from all domains: $\mathbf{z}_i = \mathbf{W}^\top\phi(\mathbf{x}_i)$ and

$$\mathbf{K} = \begin{bmatrix} \mathbf{K}^{11} & \cdots & \mathbf{K}^{1m} \\ \vdots & \ddots & \vdots \\ \mathbf{K}^{m1} & \cdots & \mathbf{K}^{mm} \end{bmatrix} \in \mathbb{R}^{n\times n} \tag{6.6}$$

is the corresponding kernel matrix, where $[\mathbf{K}^{kl}]_{ij} = \kappa(\mathbf{x}_i^k, \mathbf{x}_j^l)$. By some algebra, the domain scatter is

$$\Psi_{\mathbf{B}}\left(\{\mu_{\hat{\mathbb{P}}_X^d}\}_{d=1}^m\right) = \mathrm{Tr}(\mathbf{B}^\top\mathbf{KLKB}), \tag{6.7}$$

where $\mathbf{L}$ is a coefficient matrix

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}^{11} & \cdots & \mathbf{L}^{1m} \\ \vdots & \ddots & \vdots \\ \mathbf{L}^{m1} & \cdots & \mathbf{L}^{mm} \end{bmatrix} \in \mathbb{R}^{n\times n}$$

with $[\mathbf{L}^{kl}]_{ij} = \frac{m-1}{m^2 n_k^2}$ if $k = l$, and $-\frac{1}{m^2 n_k n_l}$ otherwise.

### 6.3.3 Class Scatter

For each class $k \in \{1, \ldots, C\}$, let $\mathbb{P}^l_{X|k}$ denote the conditional distribution on $\mathcal{X}$ induced by the total labeled domain $\mathbb{P}^l_{XY} = \frac{1}{q}\sum_{j=1}^q \mathbb{P}^j_{XY}$ when $Y = k$ (the number of labeled domains $q$ does not necessarily equal to the number of source domains $m$). We define the *within-class scatter* and *between-class scatter* as

$$\underbrace{\Psi_\phi(\mathbb{P}^l_{X|k})}_{\text{within-class-}k\text{ scatter}} \quad \text{and} \quad \underbrace{\Psi\left(\{\mu_{\mathbb{P}^l_{X|k=1}}, \ldots, \mu_{\mathbb{P}^l_{X|k=C}}\}\right)}_{\text{between-class scatter}}. \tag{6.8}$$

The class scatters are estimated as follows. Let $\mathbf{S}^w_k = \left(\phi(\mathbf{x}_j)\right)_{\mathbf{x}_j\in k}$ denote the $n_k$-tuple of source samples in class $k$. The centroid of $\mathbf{S}^w_k$ is $\boldsymbol{\mu}_k = \frac{1}{n_k}\sum_{\mathbf{x}_i\in k}\phi(\mathbf{x}_i)$. Furthermore, let $\mathbf{S}^b = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_{|C|})$ denote the $n$-tuple of all class centroids *where centroid $k$ appears $n_k$ times in* $\mathbf{S}^b$. The centroid of $\mathbf{S}^b$ is then the centroid of the source domain: $\bar{\boldsymbol{\mu}}^s = \frac{1}{n}\sum_{k=1}^{|C|} n_k \boldsymbol{\mu}_k$. It follows that the within-class scatter is

$$\Psi_\phi\left(\hat{\mathbb{P}}^l_{X|k}\right) = \mathrm{Tr}\left(\sum_{j=1}^{n_k} \left(\phi(\mathbf{x}_{jk}) - \boldsymbol{\mu}_k\right)\left(\phi(\mathbf{x}_{jk}) - \boldsymbol{\mu}_k\right)^\top\right)$$

and the between-class scatter is

$$\Psi\left(\left\{\mu_{\hat{\mathbb{P}}^l_{X|k}}\right\}_{k=1}^C\right) = \mathrm{Tr}\left(n_k(\boldsymbol{\mu}_k - \bar{\boldsymbol{\mu}})(\boldsymbol{\mu}_k - \bar{\boldsymbol{\mu}})^\top\right).$$

The right-hand sides of the above equations are the classical definitions of within- and between- class scatter [69]. The classical linear discriminant is thus a ratio of scatters

$$\text{Fisher's linear discriminant} = \frac{\Psi\left(\left\{\mu_{\hat{\mathbb{P}}^l_{X|k}}\right\}_{k=1}^C\right)}{\sum_{k=1}^C \Psi_\phi\left(\hat{\mathbb{P}}^l_{X|k}\right)}.$$

Maximizing Fisher's linear discriminant increases the separation of the data points with respect to the class clusters.

Given a linear transformation $\mathbf{W} : \mathcal{H} \to \mathbb{R}^k$, it follows from Lemma 12 that the class scatters in the projected feature space $\tilde{\mathcal{H}}$ are

$$\begin{aligned}
\Psi_{\mathbf{B}}\left(\left\{\mu_{\hat{\mathbb{P}}^l_{X|k}}\right\}_{k=1}^C\right) &= \mathrm{Tr}(\mathbf{W}^\top \mathrm{Cov}(\mathbf{S}^b)\mathbf{W}) \\
&= \mathrm{Tr}(\mathbf{B}^\top \mathbf{P}_s \mathbf{B}), && (6.9) \\
\sum_{k=1}^C \Psi_{\mathbf{B}\circ\phi}\left(\hat{\mathbb{P}}^s_{X|k}\right) &= \sum_{k=1}^C \mathrm{Tr}(\mathbf{W}^\top \mathrm{Cov}(\mathbf{S}^w_k)\mathbf{W}) \\
&= \mathrm{Tr}(\mathbf{B}^\top \mathbf{Q}_s \mathbf{B}), && (6.10)
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{P}_s &= \sum_{k=1}^C n_k(\mathbf{m}_k - \bar{\mathbf{m}})(\mathbf{m}_k - \bar{\mathbf{m}})^\top, && (6.11) \\
\mathbf{Q}_s &= \sum_{k=1}^C \mathbf{K}_k \mathbf{H}_k \mathbf{K}_k^\top, && (6.12)
\end{aligned}$$

with $\mathbf{m}_k = \frac{1}{n_k}\sum_{j=1}^{n_k}\kappa(\cdot,\mathbf{x}_{jk})$, $\bar{\mathbf{m}} = \frac{1}{n}\sum_{j=1}^{n}\kappa(\cdot,\mathbf{x}_j)$, $[\mathbf{K}_k]_{ij} = [\kappa(\mathbf{x}_{ik},\mathbf{x}_{jk})]$, and the centering matrix $\mathbf{H}_k = \mathbf{I}_{n_k} - \frac{1}{n_k}\mathbf{1}_{n_k}\mathbf{1}_{n_k}^\top$, where $\mathbf{I}_{n_k}$ denotes a $n_k \times n_k$ identity matrix and $\mathbf{1}_{n_k} \in \mathbb{R}^{n_k}$ denotes a vector of ones.

### 6.3.4 The Algorithm

Here we formulate the SCA's learning algorithm by incorporating the above four quantities. The objective of SCA is to seek a representation by solving an optimization problem in the form of the following expression

$$\sup \frac{\{\text{total scatter}\} + \{\text{between-class scatter}\}}{\{\text{domain scatter}\} + \{\text{within-class scatter}\}}. \tag{6.13}$$

Using (6.3), (6.7), (6.9), and (6.10), the above expression can then be specified in more detail:

$$\underset{\mathbf{B}}{\operatorname{argmax}} \frac{\Psi_{\mathbf{B}\circ\phi}\left(\hat{\hat{\mathbb{P}}}_X\right) + \Psi_{\mathbf{B}}\left(\left\{\mu_{\hat{\mathbb{P}}^l_{X|k}}\right\}_{k=1}^{C}\right)}{\Psi_{\mathbf{B}}\left(\left\{\mu_{\hat{\mathbb{P}}^d_X}\right\}_{d=1}^{m}\right) + \sum_{k=1}^{C}\Psi_{\mathbf{B}\circ\phi}\left(\hat{\mathbb{P}}^l_{X|k}\right)}. \tag{6.14}$$

Maximizing the numerator encourages SCA to preserve the total variability of the data and the separability of classes. Minimizing the denominator encourages SCA to find a representation for which the source and target domains are similar, and source samples sharing a label are similar.

**Objective function.** We reformulate (6.14) in three ways. First, we express it in terms of linear algebra. Second, we insert hyper-parameters that control the trade-off between scatters as one scatter quantity could be more important than others in a particular case. Third, we impose the constraint that $\mathbf{W}^\top\mathbf{W} = \mathbf{B}^\top\mathbf{K}\mathbf{B} = \mathbf{I}$ to control the scale of the solution.

Explicitly, SCA finds a projection matrix $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_k]$ that solves the constrained optimization

$$\underset{\mathbf{B}\in\mathbb{R}^{n\times k}}{\operatorname{argmax}} \frac{\operatorname{Tr}\left(\mathbf{B}^\top((1-\beta)\mathbf{K}\mathbf{K} + \beta\mathbf{P})\mathbf{B}\right)}{\operatorname{Tr}\left(\mathbf{B}^\top(\delta\mathbf{K}\mathbf{L}\mathbf{K} + \mathbf{Q})\mathbf{B}\right)} \quad \text{s.t.} \quad \mathbf{B}^\top\mathbf{K}\mathbf{B} = \mathbf{I}, \tag{6.15}$$

where

$$\mathbf{P} = \begin{bmatrix} \mathbf{P}_s & \mathbf{0}_{n_s \times n_t} \\ \mathbf{0}_{n_t \times n_s} & \mathbf{0}_{n_t \times n_t} \end{bmatrix}, \mathbf{Q} = \begin{bmatrix} \mathbf{Q}_s & \mathbf{0}_{n_s \times n_t} \\ \mathbf{0}_{n_t \times n_s} & \mathbf{0}_{n_t \times n_t} \end{bmatrix},$$

and $\beta, \delta > 0$ are the trade-off parameters controlling the total and between-class scatter, and domain scatter respectively.

Observe that the above optimization is invariant to rescaling $\mathbf{B} \mapsto \alpha\mathbf{B}$. Therefore, optimization (6.15) can be rewritten as

$$\begin{aligned} &\underset{\mathbf{B} \in \mathbb{R}^{n \times k}}{\operatorname{argmax}} \operatorname{Tr}\left(\mathbf{B}^\top (\frac{(1-\beta)}{n}\mathbf{K}\mathbf{K} + \beta\mathbf{P})\mathbf{B}\right) \\ &\text{s.t.} \quad \operatorname{Tr}\left(\mathbf{B}^\top (\delta\mathbf{KLK} + \mathbf{Q} + \mathbf{K})\mathbf{B}\right) = 1, \end{aligned} \tag{6.16}$$

which results in Lagrangian

$$\begin{aligned} J(\mathbf{B}) = \operatorname{Tr}(\mathbf{B}^\top (\frac{(1-\beta)}{n}\mathbf{K}\mathbf{K} + \beta\mathbf{P})\mathbf{B}) - \\ \operatorname{Tr}((\mathbf{B}^\top (\delta\mathbf{KLK} + \mathbf{Q} + \mathbf{K})\mathbf{B} - \mathbf{I}_k)\mathbf{\Lambda}) \end{aligned} \tag{6.17}$$

with $\mathbf{\Lambda} \in \mathbb{R}^{k \times k}$ is a symmetric matrix. To solve (6.15), set the first derivative $\frac{\partial J(\mathbf{B})}{\partial \mathbf{B}} = 0$, inducing the generalized eigenproblem

$$(\frac{(1-\beta)}{n}\mathbf{K}\mathbf{K} + \beta\mathbf{P})\mathbf{B}^* = (\delta\mathbf{KLK} + \mathbf{K} + \mathbf{Q})\mathbf{B}^*\mathbf{\Lambda}, \tag{6.18}$$

where $\mathbf{\Lambda} = \operatorname{diag}(\lambda_1, ..., \lambda_k)$ are the $k$ leading eigenvalues and $\mathbf{B} = [\mathbf{b}_1, ..., \mathbf{b}_k]$ contains the corresponding eigenvectors. [1] Algorithm 6 provides a complete summary of SCA.

### 6.3.5 Relation to other methods

SCA is closely related to a number of feature learning and domain adaptation methods. Setting the hyper-parameters $\beta = \delta = 0$ and $\mathbf{Q} = \mathbf{0}$ recovers KPCA. Setting $\beta = 1$ and $\delta = 0$ recovers the Kernel Fisher Discriminant method [146].

[1]In the implementation, a numerically more stable variant is obtained by using (6.18) using $\delta\mathbf{KLK} + \mathbf{K} + \mathbf{Q} + \epsilon\mathbf{I}$, where $\epsilon > 0$ is a fixed small constant.

```
Algorithm 6 Scatter Component Analysis
Input:
• Sets of training datapoints $S_u^d = \{\mathbf{x}_i^d\}_{i=1}^{n_d}, \forall d = 1, \ldots, m$ and their corresponding matrices $\mathbf{X} = \left[\mathbf{X}^1; \ldots; \mathbf{X}^m\right] \in \mathbb{R}^{n \times p}$, where $\mathbf{X}^d = [\mathbf{x}_1^d, \ldots, \mathbf{x}_{n_d}^d]^\top$;
• Training labels $\mathbf{y}^l = [y_1^1, \ldots, y_{n_1}^1, \ldots, y_1^q, \ldots, y_{n_q}^q]^\top \in \mathbb{R}^n$;
• Hyper-parameters $\beta, \delta > 0$; kernel bandwidth $\sigma$;
• Number of subspace bases $k$;
1: Construct kernel matrix $\mathbf{K}$ from $\mathbf{X}$, matrices $\mathbf{L}$, $\mathbf{P}$ and $\mathbf{Q}$ based on (6.6), (6.11), (6.12), and (6.15), and apply the centering operation $\mathbf{K} \leftarrow \mathbf{K} - \mathbf{1}_n\mathbf{K} - \mathbf{K}\mathbf{1}_n + \mathbf{1}_n\mathbf{K}\mathbf{1}_n$, where $n = \sum_{d=1}^m n_d$ and $[1_n]_{ij} := \frac{1}{n}$;
2: Obtain the transformation $\mathbf{B}^*$ and its corresponding eigenvalues $\mathbf{\Lambda}$ by solving the generalized eigendecomposition problem in Eq. (6.18) and selecting the $k$ leading eigenvectors;
3: Target feature extraction: Let $S_u = \bigcup_{d=1}^m S_u^d$ be the total training sample and $S_u^t$ be a target sample (for domain adaptation, $S_u^t \subset S_u$). Construct a kernel matrix $[\mathbf{K}^t]_{ij} = \kappa(\mathbf{x}_i, \mathbf{t}_j), \forall \mathbf{x}_i \in S_u, \mathbf{t}_j \in S_u^t$. The extracted features are given by $\mathbf{Z}^t = \mathbf{K}^{t\top}\mathbf{B}^*\mathbf{\Lambda}^{-\frac{1}{2}}$
Output:
• Optimal transformation matrix $\mathbf{B}^* \in \mathbb{R}^{n \times k}$;
• Feature matrix $\mathbf{Z}^t \in \mathbb{R}^{n_t \times k}$.
```

KFD with linear kernel is equivalent to Fisher's linear discriminant, which is the basis of a domain adaptation method for object detection proposed in [206].

Setting $\beta = 0$ and $\mathbf{Q} = \mathbf{0}$ (that is, ignoring class separation) yields a new algorithm: unsupervised Scatter Component Analysis (uSCA), which is closely related to TCA. The difference between the two algorithms is that TCA constrains the total variance and regularizes the transform, whereas uSCA trades-off the total variance and constrains the transform (recall that $\mathbf{B}^\top\mathbf{K}\mathbf{B} = \mathbf{I}$) motivated by Theorem 11. Eliminating the orthogonality constraint in (6.15) from uSCA yields TCA [166]. It turns out that uSCA consistently outperforms TCA in the case of domain adaptation, see Section 6.5.

In addition, the semi-supervised extension SSTCA of TCA differs markedly from SCA. Instead of incorporating within- and between- class scatter into the objective function, SSTCA incorporates a term derived from the Hilbert-Schmidt Independence Criterion that maximizes the dependence of the embedding on labels.

uSCA is closely related to unsupervised Domain Invariant Component Analysis (uDICA) in the case where there are two domains [152]. However, as for SSTCA, *supervised* DICA incorporates label-information differently from SCA – via the notion of a central subspace. In particular, supervised DICA requires that all data points are labeled, and so it cannot be applied in our experiments.

## 6.3.6 Computational Complexity

Here we analyze the computation complexity of the SCA algorithm. Suppose that we have $m$ domains with $n_1, \ldots, n_m$ are the number of samples for each domain ($m > 2$ covers the domain generalization case). Denote the total number of samples by $n = n_1 + \ldots + n_m$ and the number of leading eigenvectors by $k \ll n$. Computing the matrices $\mathbf{K}$, $\mathbf{L}$, $\mathbf{P}$, and $\mathbf{Q}$ takes $O(n^2)$ (Line 1 at Algorithm 6). Hence, the total complexity of SCA after solving the eigendecomposition problem (Line 2) takes $O(kn^2)$, or quadratic in $n$. This complexity is similar to that of KPCA and Transfer Component Analysis [166].

In comparison to Transfer Joint Matching (TJM) [135], the prior state-of-the-art domain adaptation algorithm for object recognition, TJM uses an alternating eigendecomposition procedure in which $T$ iterations are needed. Using our notation, the complexity of TJM is $O(Tkn^2)$, i.e., TJM is $T$ times slower than SCA.

## 6.3.7 Hyper-parameter Settings

Before reporting the detailed evaluation results, it is important to first explain how SCA hyper-parameters were tuned. The formulation of SCA described in Section 6.3 has four hyper-parameters: 1) the choice of the kernel, 2) the number of subspace bases $k$, 3) the between-class and total scatters trade-off $\beta$, and 4) the domain scatter $\delta$. Tuning all those hyper-parameters using a standard strategy, e.g., a grid-search, might be impractical due to two reasons. The first is of the computational complexity. The second, which is crucial, is that cross-validating a large number of hyper-parameters may worsen the generalization on the target domain, since labeled samples from the target domain are not available.

Our strategy to deal with the issue is to reduce the number of tunable hyper-parameters. For the kernel selection, we chose the RBF kernel

$\exp(\frac{-\|\mathbf{a}-\mathbf{b}\|_2^2}{\sigma^2}), \forall \mathbf{a}, \mathbf{b} \in \mathcal{X}$, where the kernel bandwidth $\sigma$ was set analytically to the median distance between samples in the aggregate domain following [87],

$$\sigma = \text{median}(\|\mathbf{a}-\mathbf{b}\|_2^2), \forall \mathbf{a}, \mathbf{b} \in S^s \cup S^t. \tag{6.19}$$

For domain adaptation, $\delta$ was fixed at 1. Thus, only two hyper-parameters remain tunable: $k$ and $\beta$. For domain generalization, $\beta$ was set at 1, i.e., the total scatter was eliminated, and $\delta$ was allowed to be tuned – the number of tunable hyper-parameters remains unchanged. The configuration is based on an empirical observation that setting $0 < \beta < 1$ is no better (if not worse) than $\beta = 1$ in terms of both the cross-validation and test performance for domain generalization cases. In all evaluations, we used 5-fold cross validation using source labeled data to find the optimal $k$ and $\beta$. We found that this strategy is sufficient to produce good SCA models for both domain adaptation and generalization cases.

## 6.4 Analysis of Adaptation Performance

We derive a bound for domain adapation that shows how the MMD controls generalization performance in the case of the squared loss $\ell(y, y') = (y - y')^2$. Despite the widespread use of the MMD for domain adaptation [60,75,133,134,166], to the best of our knowledge, this is the first generalization bound. The main idea is to incorporate the MMD (that is, domain scatter) into the adaptation bound proven for the *discrepancy distance* [140]. A generalization bound for domain generalization in terms of domain scatter is given in [152], see remark 1.

Let Hyp $:= \{h : \mathcal{X} \to \mathcal{Y}\}$ denote a hypothesis class of functions from $\mathcal{X}$ to $\mathcal{Y}$ where $\mathcal{X}$ is a compact set. Given a loss function defined over pairs of labels $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ and a distribution $\mathbb{D}$ over $\mathcal{X}$, let $\mathcal{L}_{\mathbb{D}}\Big(h, h'\Big) = \mathbb{E}_{x\sim\mathbb{D}}[\ell(h(x), h'(x))]$ denote the expected loss for any two hypotheses $h, h' \in$ Hyp. We consider the case where the hypothesis set Hyp is a subset of an RKHS $\mathcal{H}$.

We first introduce discrepancy distance, $\text{disc}_{\text{Hyp}}(\mathbb{P}, \mathbb{Q})$, which measures the difference between two distributions $\mathbb{P}$ and $\mathbb{Q}$.

**Definition 13** (**Discrepancy Distance** [140]). *Let* Hyp $\subset \{f : \mathcal{X} \to \mathcal{Y}\}$ *be a set of functions mapping from* $\mathcal{X}$ *to* $\mathcal{Y}$. *The* ***discrepancy distance*** *between two* *distributions $\mathbb{P}$ and $\mathbb{Q}$ over $\mathcal{X}$ is defined by*

$$\mathrm{disc}(\mathbb{P},\mathbb{Q}) = \sup_{h,h' \in \mathrm{Hyp}} \Big| \mathcal{L}_{\mathbb{P}}(h,h') - \mathcal{L}_{\mathbb{Q}}(h,h') \Big| \tag{6.20}$$

The discrepancy is symmetric and satisfies the triangle inequality, but it does not define a distance in general: $\exists \mathbb{P} \neq \mathbb{Q}$ such that $\mathrm{disc}_{\mathrm{Hyp}}(\mathbb{P},\mathbb{Q}) = 0$ [140]. However, it is a valid distance if $\mathrm{Hyp} = \{f : \|f\|_{\mathcal{H}} < k\} \subset \mathcal{H}$ for some $k > 0$, where $\mathcal{H}$ is an RKHS endowed with a *universal kernel* [203], and $\ell$ is the squared loss [47].

The first step of the proof is to find a relationship between domain scatter and the discrepancy distance in RKHS. To do so, we introduce the *multiplication operator*:

**Definition 14** (**Multiplication Operator**). *Let $C(\mathcal{X})$ be the space of continuous functions on the compact set $\mathcal{X}$ equipped with the supremum norm $\|\cdot\|_\infty$. Given $g \in C(\mathcal{X})$, define the multiplication operator as the bounded linear operator $\mathbf{M}_g : C(\mathcal{X}) \to C(\mathcal{X})$ given by*

$$\mathbf{M}_g(h)(x) = g(x)h(x).$$

Note that a general RKHS is *not* closed under the multiplication operator [90]. However, if the kernel is a *universal kernel* [145], i.e. satisfies $\mathcal{H} = C(\mathcal{X})$ as topological spaces, then $\mathcal{H}$ is closed under multiplication since the space of continuous functions $C(\mathcal{X})$ is closed under multiplication. The most important example of a universal kernel is the Gaussian RBF kernel, which is the kernel used in the experiments below.

The following Lemma provides the upper bound for norm of multiplication operator, which will be useful to prove our main theorem.

**Lemma 14.** *Given $g$, $h \in \mathcal{H}$, where $\mathcal{H}$ is equipped with a universal kernel, it holds that $\|\mathbf{M}_g(h)\|_{\mathcal{H}} = \|g \cdot h\|_{\mathcal{H}} \leq \|g\|_\infty \cdot \|f\|_{\mathcal{H}}$.*

*Proof.* Straightforward calculation. The Lemma requires a universal kernel since $\|g \cdot h\|_{\mathcal{H}}$ is only defined if $g \cdot h \in \mathcal{H}$. □

We now provide a theorem that shows that domain scatter of two distributions provides an upper bound for discrepancy distance.

**Theorem 15** (**Domain scatter bounds discrepancy**). *Let $\mathcal{H}$ be an RKHS with a universal kernel. Suppose that $\ell(y,y')=(y-y')^2$ is the square loss, and consider the hypothesis set*

$$\mathrm{Hyp}=\{f\in\mathcal{H}\,:\,\|f\|_{\mathcal{H}}\le 1 \text{ and } \|f\|_\infty\le r\},$$

*where $r>0$ is a constant Let $\mathbb{P}$ and $\mathbb{Q}$ be two domains over $\mathcal{X}$. Then the following inequality holds:*

$$\underbrace{\mathrm{disc}_\ell(\mathbb{P},\mathbb{Q})}_{discrepancy}\le\underbrace{8r\sqrt{\Psi_\phi(\{\mu_\mathbb{P},\mu_\mathbb{Q}\})}}_{domain\ scatter}. \tag{6.21}$$

*Proof.* Let $h,h'\in\mathrm{Hyp}$. Observe that

$$\begin{aligned}
\mathrm{disc}_\ell(\mathbb{P},\mathbb{Q}) &= \sup_{h,h'\in\mathrm{Hyp}}\left|\mathop{\mathbb{E}}_{x\sim\mathbb{P}}\left[(h(x)-h'(x))^2\right]-\mathop{\mathbb{E}}_{x\sim\mathbb{Q}}\left[(h(x)-h'(x))^2\right]\right|\\
&= \sup_{h,h'\in\mathrm{Hyp}}\Big|\mathop{\mathbb{E}}_{x\sim\mathbb{P}}[h(x)h(x)-2h(x)h'(x)+h'(x)h'(x)]\\
&\quad-\mathop{\mathbb{E}}_{x\sim\mathbb{Q}}[h(x)h(x)-2h(x)h'(x)+h'(x)h'(x)]\Big|\\
&= \sup_{h,h'\in\mathrm{Hyp}}\Big|\mathop{\mathbb{E}}_{x\sim\mathbb{P}}\left[\left\langle\mathbf{M}_h h-2\mathbf{M}_{h'}h+\mathbf{M}_{h'}h',\phi(x)\right\rangle_{\mathcal{H}}\right]\\
&\quad-\mathop{\mathbb{E}}_{x\sim\mathbb{Q}}\left[\left\langle\mathbf{M}_h h-2\mathbf{M}_{h'}h+\mathbf{M}_{h'}h',\phi(x)\right\rangle_{\mathcal{H}}\right]\Big|\\
&= \sup_{h,h'\in\mathrm{Hyp}}\left|\left\langle\mathbf{M}_h h-2\mathbf{M}_{h'}h+\mathbf{M}_{h'}h',\mu_\mathbb{P}-\mu_\mathbb{Q}\right\rangle_{\mathcal{H}}\right|\\
&\le \|\mathbf{M}_h h-2\mathbf{M}_{h'}h+\mathbf{M}_{h'}h'\|_{\mathcal{H}}\cdot\|\mu_\mathbb{P}-\mu_\mathbb{Q}\|_{\mathcal{H}}\\
&\le \Big(\|\mathbf{M}_h h\|_{\mathcal{H}}+2\|\mathbf{M}_{h'}h\|_{\mathcal{H}}+\|\mathbf{M}_{h'}h'\|_{\mathcal{H}}\Big)\mathrm{MMD}_{\mathrm{Hyp}}[\mathbb{P},\mathbb{Q}]\\
&\le \Big(\|h\|_\infty\|h\|_{\mathcal{H}}+2\|h'\|_\infty\|h\|_{\mathcal{H}}+\|h'\|_\infty\|h'\|_{\mathcal{H}}\Big)\mathrm{MMD}_{\mathrm{Hyp}}[\mathbb{P},\mathbb{Q}]\\
&\le 4r\cdot\mathrm{MMD}_{\mathrm{Hyp}}[\mathbb{P},\mathbb{Q}],
\end{aligned}$$

where the second-to-last inequality follows from Lemma 14. The result then follows after observing that $\mathrm{disc}(\mathbb{P},\mathbb{Q})=\sup_{h,h'\in\mathrm{Hyp}}\left|\mathcal{L}_\mathbb{P}(h,h')-\mathcal{L}_\mathbb{Q}(h,h')\right|$ and applying Theorem 13. □

Theorem 15 relates *domain scatter* to generalization bounds for domain adaptation proven in [140]. Before stating the bounds, we introduce Rademacher complexity [12], which measures the degree to which a class of functions can fit random noise. This measure is the basis of bounding the empirical loss and expected loss.

**Definition 15 (Rademacher Complexity).** *Let $G$ be a family of functions mapping from $\mathcal{X} \times \mathcal{Y}$ to $[a,b]$ and $S = (z_1, ..., z_n) \in \mathcal{X} \times \mathcal{Y}$ be a fixed sample of size $n$. The empirical Rademacher complexity of $G$ with respect to the sample $S$ is*

$$\hat{\mathfrak{R}}_S(G) = \mathop{\mathbb{E}}_{\boldsymbol{\sigma}} \left[ \sup_{g \in G} \frac{1}{n} \sum_{i=1}^{n} \sigma_i g(z_i) \right], \tag{6.22}$$

*where $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)^\top$ are Rademacher variables, with $\sigma_i$s independent uniform random variables taking values in $\{-1, +1\}$. The* ***Rademacher complexity*** *over all samples of size $n$ is*

$$\mathfrak{R}_n(G) = \mathop{\mathbb{E}}_{S} \left[ \hat{\mathfrak{R}}_S(G) \right]. \tag{6.23}$$

Note that the family of functions $G$ can be associated with a bounded loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to [0, B]$ and the hypothesis set Hyp, that is, $G$ contains mappings from $(x, y) \mapsto \ell(h(x), y)$, where $h \in \text{Hyp}$. The following theorem bounds the difference between the empirical loss and the true loss by means of measuring complexity on Hyp.

**Theorem 16 (Rademacher Bound).** *Let $\ell : \mathcal{Y} \times \mathcal{Y} \to [0, B]$ be a $q$-Lipschitz loss function, i.e., for all $a, b \in \mathcal{Y} \times \mathcal{Y}$, $|\ell(a) - \ell(b)| = q|a - b|$. Then, for any $\delta > 0$, with probability at least $1 - \delta$ over all i.i.d. samples $S_{\mathcal{X}} = (x_1, \ldots, x_n)$ of size $n$, each of the following holds for any $h \in \text{Hyp}$:*

$$\mathcal{L}_{\mathbb{D}}(h, f) \leq \mathcal{L}_{\hat{\mathbb{D}}}(h, f) + 2q\hat{\mathfrak{R}}_{S_{\mathcal{X}}}(\text{Hyp}) + 3B\sqrt{\frac{\log \frac{2}{\delta}}{2n}} \tag{6.24}$$

*Proof.* This theorem follows from the standard generalization bound using Definition 15 (see, for example, Theorem 3.1 in [151])

$$\mathcal{L}_{\mathbb{D}}(h, f) \leq \mathcal{L}_{\hat{\mathbb{D}}}(h, f) + 2\hat{\mathfrak{R}}_S(G) + 3B\sqrt{\frac{\log \frac{2}{\delta}}{2n}}$$

and Ledoux and Talagrand's contraction principle [127]

$$\hat{\mathfrak{R}}_S(G) \leq q\hat{\mathfrak{R}}_{S_{\mathcal{X}}}(\text{Hyp}). \tag{6.25}$$

□

We now have all the ingredients to derive domain adaptation bounds in terms of domain scatter. Let $f_{\mathbb{P}}$ and $f_{\mathbb{Q}}$ be the true labeling functions on domain $\mathbb{P}$ and $\mathbb{Q}$ respectively, and $h^*_{\mathbb{P}} := \text{argmin}_{h\in\text{Hyp}}\mathcal{L}_{\mathbb{P}}(h, f_{\mathbb{P}})$ and $h^*_{\mathbb{Q}} := \text{argmin}_{h\in\text{Hyp}}\mathcal{L}_{\mathbb{Q}}(h, f_{\mathbb{Q}})$ be the minimizers. For a successful domain adaptation, we shall assume that $\mathcal{L}_{\mathbb{P}}(h^*_{\mathbb{P}}, h^*_{\mathbb{Q}})$ is small. The following theorem provides a domain adaptation bound in terms of scatter (recall that the MMD is a special case of scatter by Theorem 13).

**Theorem 17** (**Adaptation bounds with domain scatter**). *Let* Hyp *be a family of functions mapping from $\mathcal{X}$ to $\mathbb{R}$, $S^{\mathbb{P}}_{\mathcal{X}} = (x^t_1, \ldots, x^t_{n_s}) \sim \mathbb{P}$ and $S^{\mathbb{Q}}_{\mathcal{X}} = (x^t_1, \ldots, x^t_{n_t}) \sim \mathbb{Q}$ be a source and target sample respectively. Let the rest of the assumptions be as in Theorems 15 and 16. For any hypothesis $h \in$* Hyp*, with probability at least $1-\delta$, the following adaptation bound holds:*

$$\begin{aligned}\overbrace{\mathcal{L}_{\mathbb{Q}}(h, f_{\mathbb{Q}}) - \mathcal{L}_{\mathbb{Q}}(h^*_{\mathbb{Q}}, f_{\mathbb{Q}})}^{\textit{regret on target domain}} \leq{} & \overbrace{\mathcal{L}_{\hat{\mathbb{P}}}(h, h^*_{\mathbb{P}})}^{\textit{empirical loss}} + \overbrace{2q\hat{\mathfrak{R}}_{S^{\mathbb{P}}_{\mathcal{X}}}(\text{Hyp})}^{\textit{Rademacher complexity}} \\ + \underbrace{3B\sqrt{\frac{\log\frac{2}{\delta}}{2n_t}}}_{O(1/\sqrt{\textit{sample size}})} & + \underbrace{8r\sqrt{\Psi_\phi(\{\mu_{\mathbb{Q}}, \mu_{\mathbb{P}}\})}}_{\textit{domain scatter}} + \underbrace{\mathcal{L}_{\mathbb{P}}(h^*_{\mathbb{P}}, h^*_{\mathbb{Q}})}_{\textit{deviation of optimal solns}}\end{aligned} \tag{6.26}$$

*Proof.* Fix $h \in$ Hyp. Since the square loss is symmetric and obeys the triangle inequality, Theorem 8 in [140] implies that

$$\begin{aligned}\mathcal{L}_{\mathbb{Q}}(h, f_{\mathbb{Q}}) - \mathcal{L}_{\mathbb{Q}}(h^*_{\mathbb{Q}}, f_{\mathbb{Q}}) \leq \mathcal{L}_{\mathbb{P}}(h, h^*_{\mathbb{P}}) + \text{disc}_\ell(\mathbb{Q}, \mathbb{P}) \\ + \mathcal{L}_{\mathbb{P}}(h^*_{\mathbb{P}}, h^*_{\mathbb{Q}}).\end{aligned} \tag{6.27}$$

The result then follows by Theorem 15 combined with Theorem 16. □

It is instructive to compare Theorem 17 above with Theorem 9 in [140], which is the analog if we expand $\text{disc}_l(\mathbb{Q}, \mathbb{P})$ in (6.27) with its empirical measure. It is also straightforward to rewrite the bound in term of the *empirical scatter* $\Psi_\phi(\{\mu_{\hat{\mathbb{P}}}, \mu_{\hat{\mathbb{Q}}}\})$ by applying Theorem 11.

The significance of Theorem 17 is twofold. First, it highlights that the scatter $\Psi_\phi(\{\mu_{\mathbb{P}}, \mu_{\mathbb{Q}}\})$ controls the generalization performance in domain adaptation. Second, the bound shows a direct connection between *scatter* (also MMD) and the domain adaptation theory proposed in [140]. Note that the bound might not be useful for practical purposes, since it is loose and pessimistic as they hold for all hypotheses and all possible data distributions.

**Remark 1** (**The role of scatter in domain generalization**)**.** *Theorem 5 of [152] shows that the domain scatter (or, alternatively, the distributional variance) is one of the key terms arising in a generalization bound in the setting of domain generalization.*

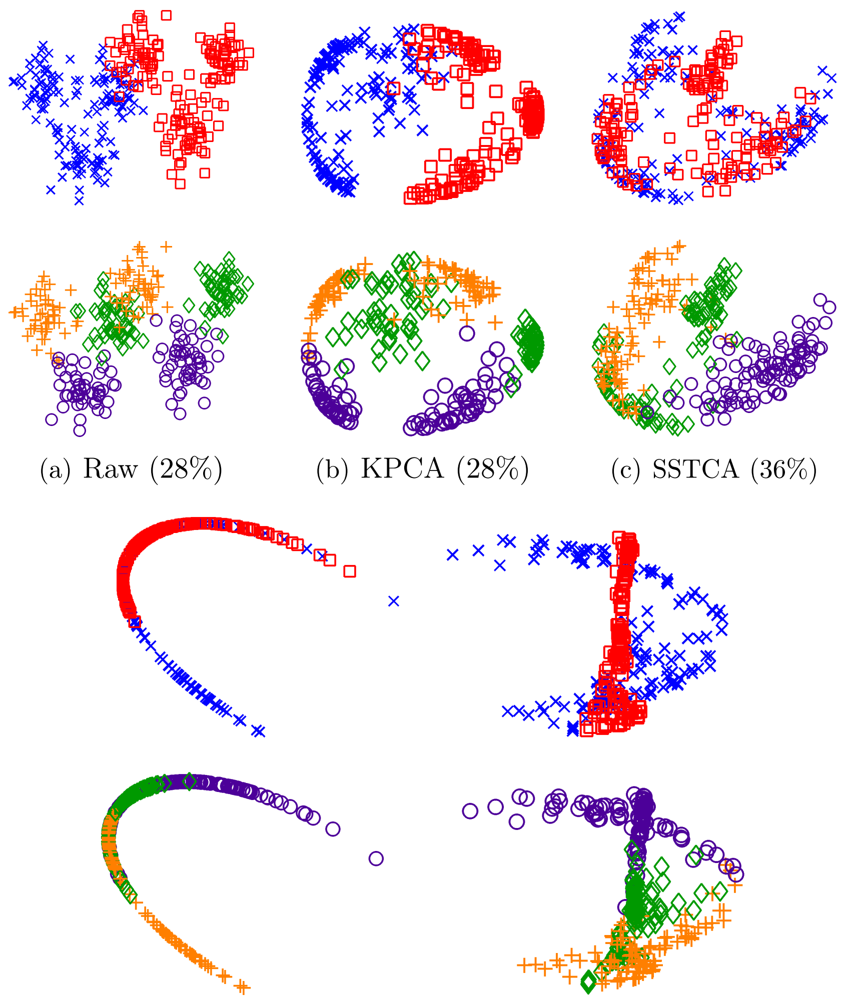
## 6.5 Experiment 1 : Domain Adaptation

The first set of experiments evaluated the domain adaptation performance of SCA on synthetic data and real-world object recognition tasks. The synthetic data was designed to understand the behavior of the learned features compared to other algorithms, whereas the real-world images were utilized to verify the performance of SCA.

The experiments are divided into three parts. Section 6.5.1 visualizes performance on synthetic data. Section 6.5.2 evaluates performance on a range of cross-domain object recognition tasks with a standard yet realistic hyper-parameter tuning. Section 6.5.3 reports some results with a tuning protocol established in the literature for completeness.

### 6.5.1 Synthetic data

Figures 6.1 depicts synthetic data that consists of two dimensional data points under three classes with six clusters. The data points in each cluster were generated from a Gaussian distribution $x_i^c \sim \mathcal{N}(\mu^c, \sigma^c)$, where $\mu^c$ and $\sigma^c$ is the mean and standard deviation of the $c$-th cluster. The RBF kernel $k(\mathbf{a}, \mathbf{b}) = \exp(-\frac{\|\mathbf{a}-\mathbf{b}\|_2^2}{\sigma^2})$ was used for all algorithms. All tunable hyper-parameters were selected according



(a) Raw (28%) (b) KPCA (28%) (c) SSTCA (36%)

(d) TJM (44%) (e) SCA (77%)

Figure 6.1: Projections of the synthetic data onto the first two leading eigenvectors. Numbers in brackets indicate the classification accuracy on the target using 1-nearest neighbor (1NN). The top and bottom rows show the domains and classes respectively.

to 1-nearest neighbor's test accuracy. We compare features extracted from Kernel Principal Component Analysis (KPCA), Semi-Supervised Transfer Component Analysis (SSTCA) [166], Transfer Joint Matching (TJM) [135], and SCA.

The top row of Figures 6.1 illustrates how the features extracted from the MMD-based algorithms (SSTCA, TJM, and SCA) reduce the domain mismatch.

Red and blue colors indicate the source and target domains, respectively. Good features for domain adaptation should have a configuration of which the red and blue colors are mixed. This effect can be seen in features extracted from SSTCA, TJM, and SCA, which indicates that the domain mismatch is successfully reduced in the feature space. In classification, domain adaptive features should also have a certain level of class separability. The bottom row highlights a major difference between SCA and the other algorithms in terms of the class separability: the SCA features are more clustered with respect to the classes, with more prominent gaps among clusters. This suggests that it would be easier for a simple function to correctly classify SCA features.

### 6.5.2 Real world object recognition

We summarize the complete domain adaptation results over a range of cross-domain object recognition tasks. Several real-world image datasets were utilized such as handwritten digits (MNIST [125] and USPS [106]) and general objects (MSRC [235], VOC2007 [62], Caltech-256 [89], Office [187]). Three cross-domain pairs were constructed from these datasets: USPS+MNIST, MSRC+VOC2007, and Office+Caltech.

**Data setup.** The USPS+MNIST pair consists of raw images subsampled from datasets of handwritten digits. MNIST contains 60,000 training images and 10,000 test images of size $28 \times 28$. USPS has 7,291 training images and 2,007 test images of size $16 \times 16$ [125] . The pair was constructed by randomly sampling 1,800 images from USPS and 2,000 images from MNIST. Images were uniformly rescaled to size $16 \times 16$ and encoded into feature vectors representing the gray-scale pixel values. Two SOURCE $\rightarrow$ TARGET classification tasks were constructed: USPS $\rightarrow$ MNIST and MNIST $\rightarrow$ USPS.

The MSRC+VOC2007 pair consist of 240-dimensional images that share 6 object categories: “aeroplane”, “bicycle”,“bird”, “car”, “cow”, and “sheep” taken from the MSRC and VOC2007 [62] datasets. The pair was constructed by selecting all 1,269 images in MSRC and 1,530 images in VOC2007. As in [133], features were extracted from the raw pixels as follows. First, images were uniformly rescaled to be 256 pixels in length. Second, 128-dimensional dense SIFT (DSIFT) features were extracted using the VLFeat open source package [223]. Finally, a 240-dimensional codebook was created using K-means clustering to obtain the codewords.

The Office+Caltech consists of 2,533 images of ten categories (8 to 151 images per category per domain), that forms four domains: $(A)$ AMAZON, $(D)$ DSLR, $(W)$ WEBCAM, and $(C)$ CALTECH. AMAZON images were acquired in a controlled environment with studio lighting. DSLR consists of high resolution images captured by a digital SLR camera in a home environment under natural lighting. WEBCAM images were acquired in a similar environment to DSLR, but with a low-resolution webcam. Finally, CALTECH images were collected from Google Images [89]. Taking all possible source-target combinations yields 12 cross-domain datasets denoted by $A \to W, A \to D, A \to C, \ldots, C \to D$. We used two types of extracted features from these datasets that are publicly available: SURF-BoW[2] [187] and DeCAF$_6$[3] [56]. **SURF-BoW** features were extracted using SURF [15] and quantized into 800-bin histograms with codebooks computed by K-means on a subset of AMAZON images. The final histograms were standardized to have zero mean and unit standard deviation in each dimension. **Deep Convolutional Activation Features (DeCAF)** were constructed by [56] using the deep convolutional neural network architecture in [118]. The model inputs are the mean-centered raw RGB pixel values that are forward propagated through 5 convolutional layers and 3 fully-connected layers. We used the outputs from the 6th layer as the features, leading to $4,096$ dimensional DeCAF$_6$ features.

**Baselines and protocol.** We evaluated the following algorithms: 1) a classifier on raw features (Raw), 2) KPCA, 3) Transfer Component Analysis (TCA) [166], 4) SSTCA, 5) Geodesic Flow Kernel (GFK) [83], 6) Transfer Sparse Coding (TSC) [133], 7) Subspace Alignment (SA) [67], 8) TJM [135], 9) unsupervised Scatter Component Analysis, and 10) SCA. For a realistic setting, the tunable hyper-parameters were optimized according to labels from source domains only.

The above feature learning algorithms were evaluated on three different classifiers: 1) 1-nearest neighbor (1NN), 2) support vector machines with linear kernel (L-SVM) [29], and 3) domain adaptation machines (DAM) [60]. 1NN and L-SVM are

[2]`http://www-scf.usc.edu/~boqinggo/da.html`

[3]`http://vc.sce.ntu.edu.sg/transfer_learning_domain_adaptation/domain_adaptation_home.html`

the standard off-the-shelf classifiers, while DAM is specifically designed for domain adaptation. DAM is an extension of SVM that incorporates a domain-dependent regularization to encourage the target classifier sharing similar prediction values with the source classifiers. We also utilize the linear kernel for DAM.

**Classification accuracy with 1-nearest neighbor.** Table 6.1 summarizes the classification accuracy on the USPS+MNIST and MSRC+VOC2007 pairs. We can see that SCA is the best model on average, while the prior state-of-the-art TJM is the second best. Other domain adaptation algorithms (TCA, SSTCA, GFK, and TSC) do not perform well, even worse than one without adaptation strategy: KPCA. Surprisingly, the unsupervised version of our algorithm, uSCA, has the highest accuracy on two MSRC+VOC2007 cases. This indicates that the label incorporation does not help improve domain adaptation on the MSRC+VOC2007, while it clearly does on the USPS+MNIST. Furthermore, SCA and uSCA always provide improvement over the raw features, while other algorithms, including TJM, fail to do so in MNIST $\rightarrow$ USPS case.

Surprisingly, SSTCA, which also incorporates label information during training, does not perform competitively. The first possible explanation is that SCA *directly* improves class separability, whereas SSTCA maximizes a dependence criterion that relates *indirectly* to separability. The second is that SSTCA incorporates the manifold regularization that requires a similarity graph, i.e., affinity matrix. This graph is parameterized by k-nearest neighbor with $l_2$ distance, which might not be suitable in these cases.

The results on the Office+Caltech pair are summarized in Table 6.2 (SURF-BoW) and Table 6.3 (DeCAF$_6$). In general, DeCAF$_6$ induces stronger discriminative performance than SURF-BoW features, since DeCAF$_6$ with 1NN only has already provided significantly better performance. SCA consistently has the best average performance on both features, slightly better than the prior state-of-the-art, TJM. On SURF-BoW, SCA is the best model on 3 out of 12 cases and the second best on other 4 cases. The trend on DeCAF$_6$ is better – SCA has the best performance on 5 out of 12 cases, while comes second on other 6 cases. Although the closest competitor, TJM, has the highest number of individual best cross-domain performance, it requires higher computational complexity than SCA, see the next paragraph on runtime performance analysis.

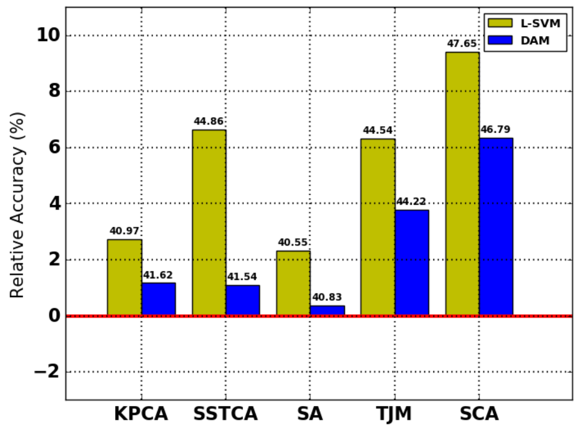
Table 6.1: Accuracy % on the USPS+MNIST and MSRC+VOC2007 datasets. Bold-red and bold-black indicate the best and second best performance.

| Dataset | Raw | KPCA | TCA | SSTCA | GFK | TSC | SA | TJM | uSCA | SCA |
|---|---|---|---|---|---|---|---|---|---|---|
| USPS → MNIST | 34.80 | 42.55 | 41.75 | 40.07 | 43.50 | 40.95 | 41.50 | **52.65** | 44.86 | **48.00** |
| MNIST → USPS | 63.06 | 62.61 | 59.44 | 60.13 | 61.22 | 59.56 | 63.95 | 62.00 | **64.67** | **65.11** |
| MSRC → VOC | 28.63 | 29.35 | 31.70 | 30.95 | 30.63 | 28.80 | 30.90 | 32.48 | **33.14** | **32.75** |
| VOC → MSRC | 41.06 | 47.12 | 45.78 | 46.06 | 44.47 | 40.58 | 46.88 | 46.34 | **49.80** | **48.94** |
| Avg. | 41.89 | 45.51 | 44.67 | 44.30 | 44.96 | 42.47 | 45.81 | **48.37** | 48.12 | **48.70** |

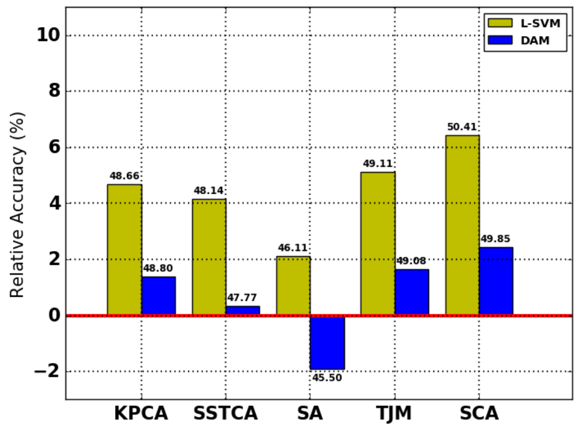
Table 6.2: Accuracy % on the Office+Caltech images with SURF-BoW features. 1NN was used as the base classifier.

| Dataset | Raw | KPCA | GFK | TCA | SSTCA | SA | TJM | uSCA | SCA |
|---|---|---|---|---|---|---|---|---|---|
| $A \to W$ | 29.83 | 31.86 | **39.32** | 25.08 | 28.15 | **37.63** | 33.56 | 32.88 | 33.90 |
| $A \to D$ | 25.48 | 33.76 | 28.66 | 31.21 | 32.25 | **34.49** | **35.67** | 33.85 | 34.21 |
| $A \to C$ | 26.00 | 37.04 | **39.27** | 33.93 | 32.48 | 37.80 | 37.58 | 37.13 | **38.29** |
| $W \to A$ | 22.96 | 29.44 | **34.03** | 22.86 | 25.56 | **34.34** | 29.85 | 30.41 | 30.48 |
| $W \to D$ | 59.24 | **89.81** | 84.71 | 65.61 | 80.81 | 80.89 | 86.62 | **89.81** | **92.36** |
| $W \to C$ | 19.86 | 27.60 | 28.76 | 23.06 | 25.39 | 28.76 | **29.72** | 28.52 | **30.63** |
| $D \to A$ | 28.50 | 31.00 | 32.25 | 30.17 | 29.16 | **34.24** | 30.06 | 31.00 | **33.72** |
| $D \to W$ | 63.39 | 84.41 | 80.34 | 64.75 | 78.90 | 82.37 | **90.85** | 84.41 | **88.81** |
| $D \to C$ | 26.27 | 27.78 | 29.12 | 28.05 | 28.05 | 31.17 | **30.72** | 27.78 | **32.32** |
| $C \to A$ | 23.70 | 40.40 | 41.75 | 41.02 | 40.67 | 41.34 | **45.41** | 40.40 | **43.74** |
| $C \to W$ | 25.76 | 31.53 | **36.61** | 23.39 | 26.62 | 32.20 | **33.90** | 29.15 | 33.56 |
| $C \to D$ | 25.48 | 40.76 | 40.13 | 34.49 | 36.45 | **42.86** | 40.31 | **42.04** | 39.49 |
| Avg. | 31.37 | 42.12 | 42.91 | 35.29 | 38.17 | 43.21 | **43.67** | 42.28 | **44.29** |

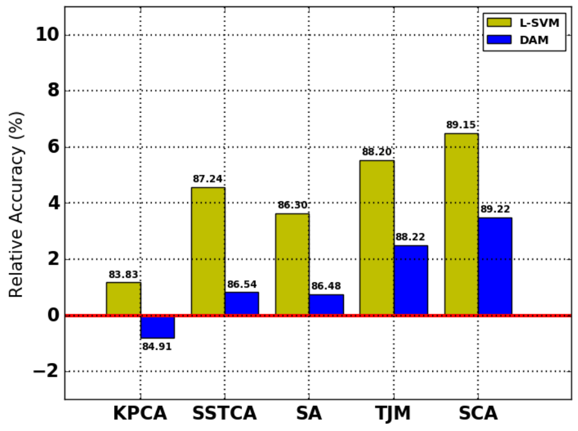
**Classification accuracy with L-SVM and DAM.** Next we report the results with L-SVM and DAM as the base classifiers for the feature learning algorithms. For succinctness, we compare the performance of five algorithms: KPCA, SSTCA, SA, TJM, and SCA, presented in Figure 6.2. The bar chart shows the average accuracies relative to the performance on Raw features (indicated by line $y = 0$ in red); the numbers alongside the bars indicate the absolute accuracies. Table 6.4 summarizes the absolute accuracies on Raw features.



(a) MNIST+USPS, MSRC+VOC



(b) Office+Caltech (SURF-BoW)



(c) Office+Caltech (DeCAF$_6$)

Figure 6.2: L-SVM and DAM average performance accuracy (%) relative to the performance on Raw features. The numbers on the top or bottom of the bars show the absolute accuracy. The red line indicates the Raw baseline performance, see Table 6.4 for the exact numbers.

Table 6.3: Accuracy % on the Office+Caltech images with DeCAF$_6$ features. 1NN was used as the base classifier.

| Dataset | Raw | KPCA | GFK | TCA | SSTCA | SA | TJM | uSCA | SCA |
|---|---|---|---|---|---|---|---|---|---|
| $A \to W$ | 57.29 | 67.80 | 68.47 | 71.86 | 70.73 | 68.81 | 72.54 | **73.22** | **75.93** |
| $A \to D$ | 64.97 | 80.89 | 79.62 | 78.34 | 80.13 | 78.34 | **85.99** | 79.43 | **85.35** |
| $A \to C$ | 70.35 | 74.53 | 76.85 | 74.18 | 72.25 | **80.05** | 78.45 | 74.62 | **78.81** |
| $W \to A$ | 62.53 | 69.42 | 75.26 | 79.96 | 75.65 | 77.77 | **82.46** | 79.52 | **86.12** |
| $W \to D$ | **98.73** | **100** | **100** | **100** | **100** | **100** | **100** | **100** | **100** |
| $W \to C$ | 60.37 | 65.72 | 74.80 | 72.57 | 69.30 | **74.89** | **79.61** | 72.81 | 74.80 |
| $D \to A$ | 62.73 | 80.06 | 85.80 | 88.20 | 87.30 | 82.67 | **91.34** | 88.71 | **89.98** |
| $D \to W$ | 89.15 | 98.31 | **98.64** | 97.29 | 97.56 | **99.32** | 98.31 | 98.31 | **98.64** |
| $D \to C$ | 52.09 | 75.16 | 74.09 | 73.46 | 74.45 | 75.69 | **80.77** | 74.98 | **78.09** |
| $C \to A$ | 85.70 | 88.73 | 88.41 | 89.25 | 88.90 | **89.46** | **89.67** | 88.52 | **89.46** |
| $C \to W$ | 66.10 | 77.29 | **80.68** | 80.00 | 81.22 | 75.93 | **80.68** | 76.27 | **85.42** |
| $C \to D$ | 74.52 | 86.62 | 86.62 | 83.44 | 84.56 | 83.44 | **87.26** | 86.62 | **87.90** |
| Avg. | 70.38 | 80.38 | 82.44 | 82.38 | 81.84 | 82.20 | **85.59** | 82.75 | **85.88** |

Table 6.4: Average accuracy (%) on Raw features.

| Dataset | 1-NN | L-SVM | DAM |
|---|---|---|---|
| MNIST+USPS, MSRC+VOC | **41.89** | 38.23 | 40.45 |
| Office+Caltech (SURF-BoW) | 31.37 | 43.98 | **47.42** |
| Office+Caltech (DeCAF$_6$) | 70.38 | 82.66 | **85.72** |

In general, all feature learning algorithms rectify the domain adaptation performances over Raw features, except in two cases: SA on the Office+Caltech with SURF-BoW features and KPCA on the Office+Caltech with DeCAF$_6$ features. Considering the absolute accuracies, we find that the best average performances on each dataset are still provided by SCA, a similar trend as in the 1NN results. This confirms the effectiveness of SCA regardless of the classifier choice, at least, among 1NN, L-SVM, and DAM.

Let us now compare the absolute average performance of L-SVM and DAM with the performance of 1NN. L-SVM and DAM evidently provide a considerable performance improvement only on the Office+Caltech dataset. Their performances on less powerful features, that is, the features extracted from the MNIST+USPS and MSRC+VOC, are even worse than 1NN. A useful lesson from this finding is that one should make use better features to take the real benefit of more advanced classifiers in the context of domain adaptation.

Finally, we seek to investigate the performance impact induced by DAM in comparison to L-SVM. DAM is expected to provide a better performance, since it is specifically designed for domain adaptation. From Table 6.4 we can see that DAM outperforms L-SVM when operating on Raw features. Surprisingly, that is not always the case when a feature learning algorithm is applied. Moreover, L-SVM always produces higher performance gain relative to Raw features.than DAM. This could be attributed to overfitting considering that DAM has more hyper-parameters than L-SVM. That is, combining DAM with a feature learning algorithm complicates the whole processs – recall that the hyper-parameter selection is based on a validation on source data.

**Runtime performance.** Table 6.5 compares the average runtime performance of SCA over all cross-domain cases in a particular domain pair with some other algorithms: KPCA, TCA, TSC, and TJM. All algorithms were executed with MATLAB R2014b by a machine with Intel Core i5-240 CPU, Arch Linux 64-bit OS, and 4GB RAM. Note that KPCA, TCA, and SCA basically utilizes the same optimization procedure: a single iteration of the eigenvalue decomposition. TJM requires several iterations of the eigenvalue decomposition with an additional gradient update in each iteration, while TSC solves the dictionary learning and sparse coding problem with an iterative procedure.

In general, SCA is significantly faster than TJM and TSC, while it is slightly slower than KPCA and TCA. Specifically, SCA is 3 to 6$\times$ faster than TJM, and $> 50\times$ faster than TSC. SCA performs at most 3$\times$ slower than TCA on relatively small datasets (MSRC+VOC and Office+Caltech), but at about the same level on larger datasets (MNIST+USPS). Considering the accuracy gap between SCA and TCA shown in the previous section, this runtime gap is non-issue. The main point of this observation is that SCA, an algorithm optimized with a simpler, less complex-in-time procedure than that of the recent proposals, can already achieve the state-of-the-art performance.

Table 6.5: Average runtime (seconds) over all cross-domain tasks in each domain pair.

| Dataset | KPCA | TCA | TJM | TSC | SCA |
|---|---|---|---|---|---|
| MNIST+USPS | 9.83 | 41.55 | 269.44 | 3072.25 | 42.74 |
| MSRC+VOC | 3.23 | 20.92 | 127.35 | 2051.05 | 36.86 |
| Office+Caltech | 0.84 | 3.03 | 29.65 | 1070.98 | 8.82 |

### 6.5.3 Results with Parameter Tuning on Target

Finally, we report results obtained using a protocol for hyper-parameter tuning, which we refer to as $\mathbf{val}_t$, where the optimal hyper-parameters were selected according using the target labels [133, 134, 236]. Since the protocol makes use of *target labels* to tune parameters, it is not valid as an unsupervised domain adaptation algorithm (that is, an algorithm for which target labels are unavailable). Nevertheless, it is established in the literature, and some of the best results were obtained under this protocol [133,134,236]. We therefore evaluate SCA under $\mathbf{val}_t$ for completeness.

Our algorithm was evaluated on the USPS+MNIST, MSRC+VOC2007, and Office+Caltech, similarly to that used in Section 6.5.2: experiments with tuning on source labels. We investigated the use of two different base classifiers on the top of feature learning-based algorithms: 1-nearest neighbor (1NN) and Logistic Regression (LR).[4] The following algorithms were compared: 1) a classifier on raw features, 2) TCA, 3) SSTCA, 4) GFK, 5) TSC, 6) TJM, 7) Domain Adaptation Machine (DAM) [60], 8) uSCA, and 9) SCA. Note that DAM is not a feature learning algorithm, but a classifier based on adapting SVM classifiers.

Tables 6.6 and 6.7 summarize the classification accuracy with 1NN and LR on the USPS+MNIST and MSRC+VOC2007, respectively. In general, LR yields better performance than 1NN on MSRC+VOC2007 cases, while 1NN is a better classifier on USPS+MNIST. SCA consistently has the average best performance on both classifiers. Lastly, we investigated the classification performance on the Office+Caltech dataset with DeCAF$_6$ features. Table 6.8 reports the results using

[4]The logistic regression we used was according to LIBLINEAR [63] library, with one hyper-parameter controlling the loss penalty, $c$. We grid-search $c$ from $\{10^{-4}, 10^{-3}, 10^{-2}, 0.1, 1, 10, 10^2, 10^3, 10^4\}$

Table 6.6: Accuracy % on the USPS+MNIST and MSRC+VOC2007 datasets. 1NN was used as the base classifier. The model hyper-parameters were tuned according to **val**$_t$ (validation on target).

| Dataset | 1NN | TCA | SSTCA | GFK | TSC | TJM | uSCA | SCA |
|---|---|---|---|---|---|---|---|---|
| USPS → MNIST | 34.80 | 44.15 | 44.30 | 46.45 | 48.56 | **52.25** | 43.65 | **53.60** |
| MNIST → USPS | 63.06 | 58.78 | 60.44 | 61.22 | 59.51 | 63.28 | **65.39** | **67.33** |
| MSRC → VOC | 28.63 | **37.12** | 36.67 | 34.18 | **37.84** | 32.75 | 36.67 | 37.06 |
| VOC → MSRC | 41.06 | 45.86 | 45.86 | 44.47 | 47.54 | **49.41** | 45.95 | **55.67** |
| Avg. | 41.89 | 46.48 | 46.82 | 46.58 | 48.36 | **49.42** | 47.92 | **53.42** |

Table 6.7: Accuracy % on the USPS+MNIST and MSRC+VOC2007 datasets. Logistic Regression (LR) was used as the base classifier for TCA, TSC, TJM, and SCA. The model hyper-parameters were tuned according to **val**$_t$ (validation on target).

| Dataset | LR | TCA | TSC | DAM | TJM | SCA |
|---|---|---|---|---|---|---|
| USPS → MNIST | 32.75 | 49.10 | **57.77** | 31.35 | 51.25 | **53.40** |
| MNIST → USPS | 55.39 | 55.22 | **60.83** | 60.44 | 59.61 | **63.06** |
| MSRC → VOC | 30.46 | 36.99 | 36.47 | 36.08 | **38.69** | **39.54** |
| VOC → MSRC | 60.20 | 62.25 | 56.74 | 60.76 | **63.28** | **63.67** |
| Avg. | 44.70 | 50.89 | 52.95 | 47.79 | **53.21** | **54.92** |

LR as the base classifier. In this dataset, 1NN classifier underperforms LR in all cross-domain cases. Thus, we do not include 1NN results here. SCA achieves the highest average accuracy over 12 cross-domain cases at lower computational cost than its closest competitor TJM, with 8 best and 2 second best cross-domain performance.

## 6.6 Experiment 2 : Domain Generalization

In the second set of experiments, we show that our proposed algorithm is also applicable for domain generalization and achieves state-of-the-art performance on visual object recognition and action recognition tasks. We evaluated our algorithms on three cross-domain datasets: the VLCS, Office+Caltech, and IXMAS [229].

Table 6.8: Accuracy % on the Office+Caltech images with **DeCAF$_6$** features. Logistic Regression (LR) was used as the base classifier. The model hyper-parameters were selected using **val**$_t$ (validation on target).

| Dataset | LR | TCA | DAM | TJM | uSCA | SCA |
|---|---|---|---|---|---|---|
| $C \to A$ | 92.28 | 92.17 | 92.38 | **93.53** | 93.00 | **93.11** |
| $C \to W$ | 81.02 | 83.39 | **86.10** | **89.15** | 83.05 | 85.42 |
| $C \to D$ | **89.17** | 87.26 | **89.81** | **89.81** | **89.17** | **89.81** |
| $A \to C$ | 85.75 | 86.46 | 86.38 | **86.82** | **92.00** | **92.00** |
| $A \to W$ | 77.29 | 85.76 | 82.71 | **89.49** | 81.02 | **86.10** |
| $A \to D$ | 87.90 | 87.26 | 85.35 | **88.54** | 88.40 | **89.17** |
| $W \to C$ | 73.91 | 82.37 | 76.49 | **82.64** | 81.74 | **84.42** |
| $W \to A$ | 77.14 | **90.40** | 79.65 | **89.46** | 88.94 | 89.24 |
| $W \to D$ | **100** | **100** | 98.09 | **98.73** | **100** | **100** |
| $D \to C$ | 79.52 | 82.99 | 81.21 | 82.73 | **85.75** | **86.02** |
| $D \to A$ | 86.95 | 90.50 | 88.94 | 90.19 | **91.54** | **91.75** |
| $D \to W$ | **98.98** | **98.98** | 98.64 | 97.63 | **99.66** | **99.66** |
| Avg. | 85.83 | 88.96 | 87.15 | **89.89** | 89.52 | **90.56** |

### 6.6.1 Data setup

The first cross-domain dataset, which we refer to as the **VLCS** consists of images from PASCAL VOC2007 (V) [62], LabelMe (L) [186], Caltech-101 (C) [89], and SUN09 (S) [40] datasets, each of which represents one domain. This dataset shares five object categories: *bird*, *car*, *chair*, *dog*, and *person*. Each domain in the VLCS dataset was divided into a training set (70%) and a test set (30%) by random selection from the overall dataset. Readers can look back to Table 5.3 for the detailed VLCS training-test configuration. We employed the DeCAF$_6$ features [56] with dimensionality of 4,096 as inputs to the algorithms. These features are publicly available.[5]

The second cross-domain dataset is the **Office+Caltech** dataset, see Section 6.5.2 for a detailed explanation about this dataset. We also used DeCAF$_6$ features extracted from this dataset.[6] The third dataset is the **IXMAS** dataset [229] that contains videos of the 11 actions, recorded with different actors, cameras, and view-

[5]`http://www.cs.dartmouth.edu/~chenfang/proj_page/FXR_iccv13/index.php`
[6]`http://vc.sce.ntu.edu.sg/transfer_learning_domain_adaptation/`

points. This dataset has been used as a benchmark for evaluating human action recognition models. To simulate the domain generalization problem, we followed the setup proposed in [236]: only frames from five actions were utilized (*check watch*, *cross arms*, *scratch head*, *sit down*, and *get up*) with domains represented represented by camera viewpoints (Cam 0, Cam 1, ..., Cam 4). The task is to learn actions from particular camera viewpoints and classify actions on unseen viewpoints. In the experiment, we used the dense trajectories features [228] extracted from the raw frames and applied K-means clustering to build a codebook with 1,000 clusters for each of the five descriptors, *i.e.*, dense trajectory, HOG, HOF, MBHx, and MBHy. The bag-of-words features were then concatenated forming a 5,000 dimensional features for each frame.

### 6.6.2 Baselines and Protocol

We compared our algorithms, uSCA and SCA, with the following baselines:

1. **Raw**: a classifier is applied directly on the raw features.

2. **KPCA** [192]: Kernel Principal Component Analysis.

3. **Undo-Bias** [113]: a multi-task SVM-based algorithm for undoing dataset bias. Three hyper-parameters $(\lambda, C_1, C_2)$ require tuning. Since the original formulation was designed for binary classification, we performed the following setup for multi-class classification purposes. We trained $C$ individual Undo-Bias classifiers $f_k^{ub} : \mathbb{R}^d \to \{-1, 1\}, \forall k = 1, \ldots, C$, where $C$ is the number of classes. At the prediction stage, given a test instance $(\hat{\mathbf{x}}, \hat{y})$ we computed $\hat{Y} := \{k | \forall k = 1, \ldots, C : f_k^{ub} = 1\}$. Finally, we verified whether $\hat{y} \in \hat{Y}$.

4. **UML** [64]: a structural metric learning-based algorithm that aims to learn a less biased distance metric for classification tasks. The initial tuning proposal for this method was using a set of weakly-labeled data retrieved from querying class labels to search engine. However, here we tuned the hyper-parameters using the same k-fold cross-validation strategy as others for a fair comparison.

5. **DICA** [152]: a kernel feature extraction method for domain generalization. DICA has three tunable hyper-parameters.

6. **LRE-SVM** [236]: a non-linear exemplar-SVMs model with a nuclear norm regularization to impose a low-rank *likelihood matrix*. LRE-SVM has four hyper-parameters ($\lambda_1$, $\lambda_2$, $C_1$, $C_2$) that require tuning.

Undo-Bias, UML, and LRE-SVM are the prior state-of-the-art domain generalization algorithms for object recognition tasks. We used 1-nearest neighbor (1NN) as the base classifier for all feature learning-based algorithms: Raw, KPCA, DICA, uSCA/uDICA, and SCA. The tunable hyper-parameters were selected according to labels from source domains. For all kernel-based methods, the kernel function is the RBF kernel, $k(\mathbf{a}, \mathbf{b}) = \exp(-\frac{\|\mathbf{a}-\mathbf{b}\|^2}{\sigma^2})$, with a kernel bandwidth $\sigma$ computed by *median heuristic*. Note that the unsupervised DICA (uDICA) is almost identical to uSCA in this case. The only difference is that uSCA has a control parameter $\delta > 0$ for the domain scatter/distributional variance term.

### 6.6.3 Results on the VLCS Dataset

On this dataset, we first conducted the standard training-test evaluation using 1-nearest neighbor (1NN), i.e., learning the model on a training set from one domain and testing it on a test set from another domain, to check the groundtruth performance and also to identify the existence of the dataset bias. The groundtruth evaluation results are summarized in Table 6.9. In general, the dataset bias indeed exists despite the use of the state-of-the-art deep convolution neural network features DeCAF$_6$ For example, the average cross-domain performance, i.e., "Mean others", is 56.63%, which is 25% drop from the corresponding in-domain performance: 75.96%. In particular, Caltech-101 has the highest bias, while LabelMe is the least biased dataset indicated by the largest and smallest performance drop, respectively.

We then evaluated the domain generalization performance over seven cross-domain recognition tasks. The complete results are summarized in Table 6.10. We can see that SCA is the best model on 5 out of 7 tasks, outperforms the prior state-of-the-art, LRE-SVM. It almost always has better performance than the 'raw' baseline, except when Caltech-101 is the target domain. On average, SCA is about 2% better than its closest competitor on this dataset, Undo-Bias. The VLCS cross-domain recognition is a hard task in general, since the best model (SCA) only

Table 6.9: The groundtruth 1NN accuracy % of five-class classification when training on one dataset (the left-most column) and testing on another (the upper-most row). The bold black numbers indicate *in-domain* performance, while the plain black indicate *cross-domain* performance. "Self" refers to training and testing on the same dataset, same as the bold black numbers and "mean others" refers to the average performance over all cross-domain cases. Dividing "self" and "mean others" results in the (percent) performance drop indicated by the red color.

| Training/Test | VOC2007 | LabelMe | Caltech-101 | SUN09 | Self | Mean others | Percent drop |
|---|---|---|---|---|---|---|---|
| VOC2007 | **72.46** | 52.45 | 89.17 | 60.00 | **72.46** | 67.20 | ∼ 7% |
| LabelMe | 54.99 | **63.74** | 79.72 | 46.90 | **63.74** | 60.54 | ∼ 5% |
| Caltech-101 | 53.70 | 44.79 | **99.53** | 44.87 | **99.53** | 47.49 | ∼ 52% |
| SUN09 | 51.63 | 50.69 | 50.71 | **68.12** | **68.12** | 51.01 | ∼ 25% |
| Mean others | 53.44 | 49.31 | 73.19 | 50.59 | **75.96** | 56.63 | ∼ 25% |

provides $< 4\%$ average improvement over the raw baseline. Furthermore, three algorithms, two of which are the domain generalization-based methods (uSCA, DICA), cannot achieve even better performance than the raw baseline.

Table 6.10: The domain generalization performance accuracy (%) on the VLCS dataset with DeCAF$_6$ features as inputs. The accuracy of all feature learning-based algorithms: Raw, KPCA, uSCA, DICA, SCA is according to 1-nearest neighbor (1NN) classifier. Bold red and bold black indicate the best and the second best performance, respectively.

| Source | Target | Raw | KPCA | Undo-Bias | UML | LRE-SVM | uSCA | DICA | SCA |
|---|---|---|---|---|---|---|---|---|---|
| L,C,S | V | 57.26 | 60.22 | 54.29 | 56.26 | **60.58** | 58.54 | 59.62 | **64.36** |
| V,C,S | L | 52.45 | 51.94 | 58.09 | 58.50 | **59.74** | 54.08 | 51.82 | **59.60** |
| V,L,S | C | **90.57** | 90.09 | 87.50 | **91.13** | 88.11 | 85.14 | 78.30 | 88.92 |
| V,L,C | S | 56.95 | 55.03 | 54.21 | **58.49** | 54.88 | 55.63 | 55.33 | **59.29** |
| C,S | V,L | 55.08 | 55.64 | **59.28** | 56.47 | 55.04 | 53.98 | 50.90 | **59.50** |
| C,L | V,S | 52.60 | 50.70 | **55.80** | 54.72 | 52.87 | 49.05 | 55.47 | **55.96** |
| V,C | L,S | 56.62 | 54.66 | **62.35** | 55.49 | 58.84 | 55.89 | 58.08 | **60.77** |
| Avg. | | 60.22 | 59.47 | **61.65** | 61.58 | 61.44 | 58.90 | 58.50 | **64.06** |

## 6.6.4 Results on the Office+Caltech Dataset

We evaluated our algorithms on several cross-domain cases constructed from the Office+Caltech dataset. The detailed evaluation results on four cases with DeCAF$_6$

are reported in Table 6.11. We do not report other cross-domain cases that are possibly constructed from this dataset, such as $A, D, C \to W$ and $A, W, C \to D$, since the simple 1NN on raw features is extremely accurate ($> 95\%$ accuracy) and there is little room for improvement.

The closest competitor to SCA is LRE-SVM. Although LRE-SVM performs best on average, SCA has the best performance on three out of four cross-domain cases and comes second on average. The only case when SCA underperforms LRE-SVM is that of $D, W \to A, C$. Note that the LRE-SVM algorithm is more complex than SCA both in the optimization procedure and in the number of tunable hyper-parameters.

However, the unsupervised version of our algorithm, uSCA, which is the same as uDICA [152] in the domain generalization case, cannot compete with the state-of-the-art models. It is only slightly better than KPCA on average. This suggests that incorporating labeled information from source domains during feature learning does improve domain generalization on the Office+Caltech cases.

Table 6.11: The domain generalization performance accuracy (%) on the Office+Caltech dataset with DeCAF$_6$ features as inputs.

| Methods | W,D,C → A | A,W,D→C | A,C→D,W | D,W→A,C | Average |
|---|---|---|---|---|---|
| Raw | 85.39 | 73.73 | 67.92 | 67.09 | 72.28 |
| KPCA | 89.14 | 75.87 | 78.99 | 68.84 | 77.71 |
| Undo-Bias | 90.98 | 85.95 | 80.49 | 69.98 | 81.85 |
| UML | 91.02 | 84.59 | 82.29 | **79.54** | 84.36 |
| LRE-SVM | **91.87** | **86.38** | **84.59** | **81.17** | **86.00** |
| uSCA | 89.46 | 77.15 | 78.10 | 71.74 | 79.11 |
| DICA | 90.40 | 84.33 | 79.65 | 69.73 | 81.02 |
| SCA | **92.38** | **86.73** | **85.84** | 75.54 | **85.12** |

## 6.6.5 Results on the IXMAS dataset

Table 6.12 summarizes the classification accuracies on the IXMAS dataset over three cross-domain cases. We can see that the standard baselines (Raw, KPCA) cannot match other algorithms with domain generalization strategies. In this dataset, SCA has the best performance on two out of three cases and on average. In particular, SCA is significantly better than others on Cam 2,3,4 → Cam

0,1 case. LRE-SVM remains the closest competitor of SCA – it has the second best average performance with one best cross-domain case.

Table 6.12: The domain generalization performance accuracy (%) on the IXMAS dataset with dense trajectory-based features.

| Methods | Cam 0,1 → 2,3,4 | Cam 2,3,4 → 0,1 | Cam 0,1,2,3 → Cam 4 | Average |
|---|---|---|---|---|
| Raw | 58.24 | 20.33 | 39.56 | 39.38 |
| KPCA | 67.77 | 41.21 | 59.34 | 56.10 |
| Undo-Bias | 69.03 | 60.56 | 56.84 | 62.14 |
| UML | 74.14 | 63.79 | 60.73 | 66.10 |
| LRE-SVM | **79.96** | **80.15** | 74.97 | **78.36** |
| uSCA | 66.67 | 51.09 | 61.54 | 59.77 |
| DICA | 65.93 | 78.02 | 62.64 | 68.86 |
| SCA | 80.59 | 85.16 | **70.33** | 78.69 |

### 6.6.6 Runtime Performance

Next we report the average (training) runtime performance over all cross-domain recognition tasks in each dataset. All algorithms were executed using the same software and machine as described in Section 6.5.2. From Table 6.13, we can see that the runtime of SCA is on par with KPCA and DICA, which is expected since they utilize the same optimization procedure: a single run with a generalized eigenvalue decomposition. In the previous subsections, we have shown that SCA provides better performance accuracy than KPCA and DICA.

SCA is significantly faster than some prior state-of-the-art domain generalization methods (Undo-Bias, UML, and LRE-SVM). For example, on the VLCS dataset, Undo-Bias, UML, and LRE-SVM require $\sim$ 30 minutes, while SCA only needs $\sim$ 5 minutes average training time. An analogous trend can also be seen in the case of Office+Caltech and IXMAS datasets. This outcome indicates that SCA is better suited for domain generalization tasks than the competing algorithms if a training stage in real time is required.

## 6.7 Chapter Summary

This chapter presents a fast representation learning algorithm that is compatible with both domain adaptation and domain generalization settings. The algorithm is

Table 6.13: Average domain generalization runtime (seconds) over all cross-domain recognition tasks in each dataset.

| Dataset | KPCA | Undo-Bias | DICA | UML | LRE-SVM | SCA |
|---|---|---|---|---|---|---|
| VLCS | 201.99 | $1,925.54$ | 336.92 | $1,652.67$ | $2,161.60$ | 300.94 |
| Office+Caltech | 6.53 | 589.50 | 17.90 | 413.25 | 695.31 | 18.49 |
| IXMAS | 0.50 | 49.14 | 0.79 | 57.67 | 65.47 | 0.96 |

built upon a simple geometrical measure, *scatter*, which is a variance that operates on reproducing kernel Hilbert space (RKHS). We refer to the resulting algorithm as Scatter Component Analysis (SCA). SCA uses variances between subsets of the data to construct a linear transformation on RKHS that dampens unimportant distinctions (within labels and between domains) and amplifies useful distinctions (between labels and overall variability). The scatter-based objective function in SCA is, to our best knowledge, the simplest way to encode the relevant structure of the domain adaptation and domain generalization problems, and also admits a fast, exact solution.

SCA is a natural extension of Kernel PCA, Kernel Fisher Discriminant and TCA. In contrast, many domain adaptation methods use objective functions that combine the total variance and MMD with quantities that are fundamentally different in kind such as the graph Laplacian [133], sparsity constraints [133,135], the Hilbert-Schmidt independence criterion [166] or the central subspace [152].

Our theoretical analysis shows that the scatter with two input domains, *i.e.*, *domain scatter*, controls the generalization performance in the setting of domain adaptation. Specifically, if scatter is endowed with a characteristic kernel [145], which implies that the RKHS is close under the multiplication operator, it provides an upper bound for discrepancy distance [140] (recall Theorem 15). Scatter with more than two input domains coincides with distributional variance [152], which provides a generalization bound in the setting of domain generalization.

From extensive experiments on performing cross-domain object recognition and action recognition tasks, we found that SCA is generally much faster than competing algorithms and provides the state-of-the-art performance on both domain adaptation and domain generalization cases. In particular, SCA runs 3 to 6 times faster than TJM and $>$ 50 times faster than TSC, and also provides better domain adaptation performance in terms of accuracy. In the case of domain generalization, SCA is also much faster than the prior state-of-the-art model, LRE-SVM. The performance accuracy of SCA is considerably higher on the VLCS dataset and competitive on the Office+Caltech and IXMAS datasets than that of LRE-SVM.

SCA can naturally be extended to semi-supervised domain adaptation without any significant changes to the algorithm, that is, only by incorporating target labels into the class scatters. Finally, we remark that it should be possible to further speed up SCA for large-scale problems using random features [173, 234].

In general, the domain adaptation and domain generalization problem, that is, dataset bias, in object recognition is a hard problem. Our feature learning algorithm can obtain the actual good performance in some cases after taking advantage of other powerful features such as DeCAF$_6$ (for images) and dense trajectory-based features (for videos). None of the comparing algorithms can significantly reduce dataset bias when applied on raw data or less powerful features. Thus, more fundamental algorithms that can generalize well in all environments given a few related domains only are crucial.

The next chapter will present a novel domain adaptation algorithm based on deep convolutional networks, which we refer to as Deep Reconstruction-Classification Networks (DRCN). DRCN attempts to find domain correspondences between two domains via a joint backpropagation training of classification (on source data) and reconstruction (on target data) tasks, which is substantially different from SCA. A possible advantage of DRCN over SCA is the scalability: DRCN is more suitable for large-scale problems.

# 7 Deep Reconstruction-Classification Networks for Domain Adaptation

*This chapter presents the final contribution of this thesis. We propose a new unsupervised domain adaptation algorithm based on a deep convolutional architecture for object recognition. The model, which we refer to as Deep Reconstruction-Classification Network (DRCN), jointly learns two tasks through the standard back-propagation: i) supervised classification of labeled source data and ii) unsupevised reconstruction of unlabeled target data. The tasks use a shared encoding representation which endows the model with a label prediction pipeline that generalizes onto the target domain. The performance of DRCN is evaluated on a series of cross-domain object recognition tasks, where DRCN provides higher accuracy than the prior state-of-the-art model in almost all cases. An interesting observation is that the DRCN's reconstruction pipeline transforms images from the source domain into images whose appearance resembles the target dataset. This suggests that DRCN's performance is due to constructing a single composite representation that encodes* *information about both the structure of target images and the classification of source images. Finally, we provide a theoretical analysis to justify the algorithm's objective in domain adaptation context.*

## 7.1 Introduction

Deep convolutional networks (ConvNets) have been long known as powerful models for object recognition [118,125,198]. Recently, ConvNets have established successes in other computer vision problems such as object detection [77], image captioning [226], and visual sentiment analysis [239]. The success of ConvNets mainly derives from two factors: i) a large amount of (labeled) training data and ii) the scalability of the ConvNet's training algorithm (that is, backpropagation).

Despite the effectiveness of ConvNets, dataset bias may still occur in particular circumstances, *e.g.*, the situation in which the target environment does not come from the same distribution/domain as that of the training data. To deal with dataset bias, the ConvNet's training should be beyond the standard machine learning strategy, which assumes that the training and target data are drawn from the same distribution. One can augment the ConvNet training algorithm such that it turns into a domain adaptation algorithm that can reduce dataset bias by leveraging knowledge from unlabeled target data. Domain adaptive ConvNets will be much more desirable than existing domain adaptation algorithms for large scale problems due to the scalability of backpropagation. The existing domain adaptation algorithms generally require either quadratic programming [34, 81] or eigen-analysis [67, 135, 166] that do not scale well as the size of training dataset increases.

Some ConvNet-based algorithms for visual domain adaptation have been proposed. DLID [41] creates an "interpolating path" between the source domain and target domain by training several convolutional feature extractors; each corresponds to a particular combination of source and target images. The outputs of all feature extractors are then concatenated to form a single representation as an input to a classifier. Another work shows that a ConvNet trained on ImageNet [185], *i.e.*, AlexNet, can be reused to extract features of other images, referred to as DeCAF [56], which provides significantly better accuracy than SURF-based features [15] on the Office dataset [187]. However, the core problem, *i.e.*, dataset bias, remains essentially unsolved since model accuracy has yet to reach a level that is satisfactory for real-world applications.

### 7.1.1 Chapter Goals

Motivated by the above issues, the goal of this chapter is to develop a new deep learning model that provides the state-of-the-art performance for unsupervised domain adaptation. Our deep model, which we refer to as *Deep Reconstruction-Classification Network* (DRCN), has a convolutional architecture that jointly learns two tasks: i) supervised *source* label prediction and ii) unsupervised *target* data reconstruction. The tasks use a shared encoding representation. DRCN can thus be viewed as a particular form of multitask learning [6,35]. It also can be viewed as an extension of the standard *unsupervised pretraining-supervised finetuning* strategy [97]; learning in DRCN alternates between unsupervised and supervised training.

Intuitively, a useful approach to domain adaptation algorithm should uncover *true correspondences* between instances from different domains – which we refer to as *domain correspondence.* This is in the same spirit as in [26,227]. The problem in unsupervised domain adaptation is that there appears to be insufficient information to construct a domain correspondence, since target labels are not available.

To achieve the overall goal of this chapter, we establish several objectives as follows:

- Whether DRCN indeed provides good domain adaptation performance on benchmark visual object datasets, better than the prior ConvNet-based domain adaptation model;

- Whether the effectiveness of DRCN relates to a property that of uncovering the intrinsic *domain correspondence* between the source and target domains;

- Whether the multitask learning strategy equipped in DRCN can be explained with a theoretically well founded framework.

### 7.1.2 Chapter Organization

This chapter is organized as follows. Section 7.2 explains the DRCN algorithm in detail. Section 7.3 presents the evaluation results of DRCN in comparison to several competing algorithms on benchmark object datasets. Section 7.4 analyzes the DRCN algorithm theoretically and shows that it is equivalent to solving semi-supervised learning problem on the target domain. Section 7.5 summarizes the chapter.

## 7.2 Deep Reconstruction-Classification Networks

This section presents the detailed description of the DRCN algorithm. It consists of the description of the architecture, the learning algorithm, and other useful aspects. We define a *domain* as a probability distribution $\mathbb{P}_{XY}$ (or just $\mathbb{P}$) on $\mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is the input space and $\mathcal{Y}$ is the output space. Denote the source domain by $\mathbb{P}^s$ and the target domain by $\mathbb{P}^t$, where $\mathbb{P}^s \neq \mathbb{P}^t$. Our concern is the *unsupervised domain adaptation* setting: given a labeled i.i.d. sample from a source domain $S^s = \{(x_i^s, y_i^s)\}_{i=1}^{n_s} \sim \mathbb{P}^s$ and an unlabeled sample from a target domain $S_u^t = \{(x_i^t)\}_{i=1}^{n_t} \sim \mathbb{P}_X^t$, find a good labeling function $f : \mathcal{X} \to \mathcal{Y}$ on $S_u^t$.

**Model architecture.** Ideally, a discriminative representation should model both the label and the structure of the data. Based on that intuition, we hypothesize that a domain-adaptive representation should satisfy two criteria: i) classify well the source domain labeled data and ii) reconstruct well the target domain unlabeled data, which can be viewed as an approximate of the ideal discriminative representation.

Our model is based on a convolutional architecture that has two pipelines with a shared encoding representation. The first pipeline is a standard convolutional network for *source label prediction* [125], while the second one is a convolutional autoencoder for *target data reconstruction* [141, 241]. Convolutional architectures are a natural choice for object recognition to capture spatial correlation of images. The model is optimized through multitask learning [35], that is, jointly learns the (supervised) source label prediction and the (unsupervised) target data reconstruction[1] tasks. The aim is that the encoding shared representation should learn the commonality between those tasks that provides useful information for cross-domain object recognition. Figure 7.1 illustrates the architecture of our proposed model, which we refer to as *Deep Reconstruction-Classification Networks (*DRCN*)*.

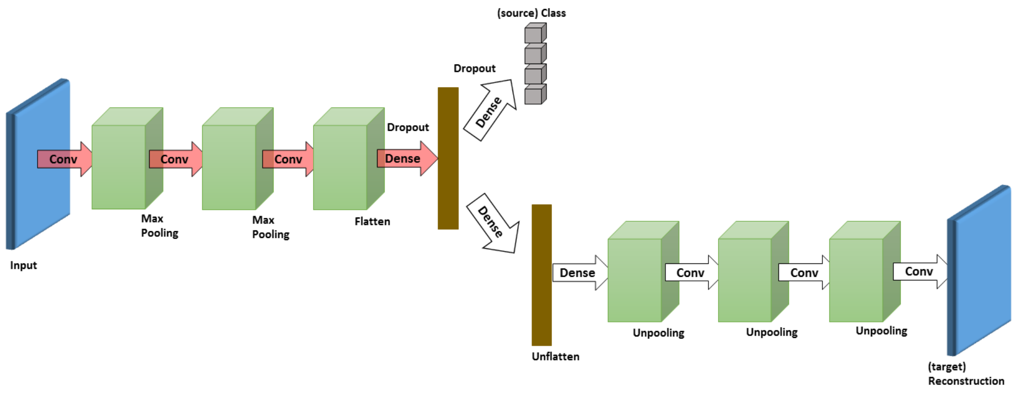


Figure 7.1: Illustration of the DRCN's architecture. It consists of two pipelines: i) label prediction and ii) data reconstruction pipelines. The shared parameters between those two pipelines are indicated by the red color.

**Formal description.** We now describe DRCN more formally. Let $f_c : \mathcal{X} \to \mathbb{R}^m$ be the (supervised) label prediction pipeline and $f_r : \mathcal{X} \to \mathcal{X}$ be the (unsupervised) data reconstruction pipeline of DRCN. Let us define three additional functions: 1) an encoder / feature mapping $g_{\text{enc}} : \mathcal{X} \to \mathcal{F}$, 2) a decoder $g_{\text{dec}} : \mathcal{F} \to \mathcal{X}$, and 3) a feature labeling $g_{\text{lab}} : \mathcal{F} \to \mathcal{Y}$. Given an input $x \in \mathcal{X}$, one can decompose $f_c$ and $f_r$ such that

$$f_c(x) = (g_{\text{lab}} \circ g_{\text{enc}})(x), \quad (7.1)$$

$$f_r(x) = (g_{\text{dec}} \circ g_{\text{enc}})(x). \quad (7.2)$$

The goal is to seek a single feature mapping $g_{\text{enc}}$ model that supports both $f_c$ and $f_r$.

[1]The unsupervised convolutional autoencoder is not trained via the greedy layer-wise fashion, but only with the standard back-propagation over the whole pipeline.

Let $\Theta_c = \{\Theta_{\text{enc}}, \Theta_{\text{lab}}\}$ and $\Theta_r = \{\Theta_{\text{enc}}, \Theta_{\text{dec}}\}$ denote the parameters of the supervised and unsupervised model. $\Theta_{\text{enc}}$ are shared parameters for the feature mapping $g_{\text{enc}}$. Note that $\Theta_{\text{enc}}, \Theta_{\text{dec}}, \Theta_{\text{lab}}$ may encode parameters of multiple or deep layers.

**Learning algorithm.** Suppose the inputs lie in $\mathcal{X} \subseteq \mathbb{R}^d$ and their labels lie in $\mathcal{Y} \subseteq \mathbb{R}^m$. Let $\ell_c : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ and $\ell_r : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be the classification and reconstruction loss respectively. Given labeled source sample $S^s = \{(\mathbf{x}_i^s, \mathbf{y}_i^s)\}_{i=1}^{n_s} \sim \mathbb{P}$, where $\mathbf{y}_i \in \{0,1\}^m$ is a *one-hot* vector, and unlabeled target sample $S_u^t = \{(\mathbf{x}_j^t)\}_{j=1}^{n_t} \sim \mathbb{Q}$, we define the empirical losses as:

$$\mathcal{L}_c^{n_s}(\{\Theta_{\text{enc}}, \Theta_{\text{lab}}\}) := \sum_{i=1}^{n_s} \ell_c\left(f_c(\mathbf{x}_i^s; \{\Theta_{\text{enc}}, \Theta_{\text{lab}}\}), \mathbf{y}_i^s\right), \tag{7.3}$$

$$\mathcal{L}_r^{n_t}(\{\Theta_{\text{enc}}, \Theta_{\text{dec}}\}) := \sum_{j=1}^{n_t} \ell_r\left(f_r(\mathbf{x}_j^t; \{\Theta_{\text{enc}}, \Theta_{\text{dec}}\}), \mathbf{x}_j^t\right). \tag{7.4}$$

Typically, $\ell_c$ is of the form *cross-entropy loss* $\sum_{k=1}^{m} y_k \log[f_c(\mathbf{x})]_k$ (recall that $f_c(\mathbf{x})$ is the softmax output) and $\ell_r$ is of the form *mean-squared loss* $\sum_{j=1}^{n_t} \left\|\mathbf{x}_j^t - f_r(\mathbf{x}_j^t)\right\|_2^2$.

Our aim is to solve the following objective:

$$\min \lambda \mathcal{L}_c^{n_s}(\{\Theta_{\text{enc}}, \Theta_{\text{lab}}\}) + (1-\lambda)\mathcal{L}_r^{n_t}(\{\Theta_{\text{enc}}, \Theta_{\text{dec}}\}), \tag{7.5}$$

where $0 \leq \lambda \leq 1$ is a hyper-parameter controlling the trade-off between classification and reconstruction. The objective is a convex combination of supervised and unsupervised loss functions. We justify the approach in Section 7.4.

Objective (7.5) can be achieved by alternately minimizing $\mathcal{L}_c^{n_s}$ and $\mathcal{L}_r^{n_t}$ using *stochastic gradient descent* (SGD). In the implementation, we used RMSprop [213], the variant of SGD with a gradient normalization – the current gradient is divided by a moving average over the previous root mean squared gradients. We utilize dropout regularization [202] during $\mathcal{L}_c^{n_s}$ minimization, which is effective to reduce overfitting. Note that dropout regularization is applied in the fully connected layers only, see Figure 7.1.

The stopping criterion for the algorithm is determined by monitoring the average reconstruction loss of the unsupervised model during training – the process is stopped when the average reconstruction loss stabilizes. Once the training is completed, the optimal parameters $\hat{\Theta}_{\text{enc}}$ and $\hat{\Theta}_{\text{lab}}$ are used to form a classification model $f_c(\mathbf{x}^t; \{\hat{\Theta}_{\text{enc}}, \hat{\Theta}_{\text{lab}}\})$ that is expected to perform well on the target domain. The DRCN learning algorithm is summarized in Algorithm 7 and implemented using Theano [13].

**Algorithm 7** The Deep Reconstruction-Classification Network (DRCN) learning algorithm.

```
Input:
• Labeled source data: $S^s = \{(\mathbf{x}_i^s, y_i^s)\}_{i=1}^{n_s}$;
• Unlabeled target data: $S_u^t = \{\mathbf{x}_j^t\}_{i=j}^{n_t}$;
• Learning rates: $\alpha_c$ and $\alpha_r$;
1: Initialize parameters $\Theta_{\text{enc}}, \Theta_{\text{dec}}, \Theta_{\text{lab}}$
2: while not stop do
3:     for all source batch of size $m_s$ do
4:         Do a forward pass according to (7.1);
5:         Let $\Theta_c = \{\Theta_{\text{enc}}, \Theta_{\text{lab}}\}$. Update $\Theta_c$:
               $\Theta_c \leftarrow \Theta_c - \alpha_c \lambda \nabla_{\Theta_c} \mathcal{L}_c^{m_s}(\Theta_c);$
6:     end for
7:     for all target batch of size $m_t$ do
8:         Do a forward pass according to (7.2);
9:         Let $\Theta_r = \{\Theta_{\text{enc}}, \Theta_{\text{dec}}\}$. Update $\Theta_r$:
               $\Theta_r \leftarrow \Theta_r - \alpha_r (1 - \lambda) \nabla_{\Theta_r} \mathcal{L}_r^{m_t}(\Theta_r).$
10:    end for
11: end while
Output:
• DRCN learnt parameters: $\hat{\Theta} = \{\hat{\Theta}_{\text{enc}}, \hat{\Theta}_{\text{dec}}, \hat{\Theta}_{\text{lab}}\}$;
```

**Data augmentation and denoising.** We use two well-known tricks to improve DRCN's performance: data augmentation and denoising. Data augmentation generates additional training data during the supervised training with respect to some plausible transformations over the original data, which improves generalization, see *e.g.* [197]. Denoising involves reconstructing *clean* inputs given their *noisy* counterparts. It is used to improve the feature invariance of denoising autoencoders (DAE) [225]. Generalization and feature invariance are two properties needed to improve domain adaptation. Since DRCN has both classification and reconstruction aspects, we can naturally apply these two tricks simultaneously in the training stage.

Let $\mathbb{Q}_{\tilde{X}|X}$ denote the noise distribution given the original data from which the noisy data are sampled from. The classification pipeline of DRCN $f_c$ thus actually observes additional pairs $\{(\tilde{\mathbf{x}}_i^s, y_i^s)\}_{i=1}^{n_s}$ and the reconstruction pipeline $f_r$ observes $\{(\tilde{\mathbf{x}}_i^t, \mathbf{x}_i^t)\}_{i=1}^{n_t}$. The noise distribution $\mathbb{Q}_{\tilde{X}|X}$ is typically of the form some geometric transformations (translation, rotation, skewing, and scaling) in data augmentation, while it is either zero-masked noise or Gaussian noise in the denoising strategy. In this work, we combine all the fore-mentioned types of noise for denoising and use only the geometric transformations for data augmentation.

## 7.3 Experiments and Results

This section reports the evaluation results of DRCN. It is divided into two parts. The first part focuses on the evaluation on large-scale datasets popular with deep learning methods, while the second part summarizes the results on the Office dataset [187].

### 7.3.1 Experiment I: SVHN, MNIST, USPS, CIFAR, and STL

The first set of experiments investigates the empirical performance of DRCN on five widely used benchmarks: MNIST [125], USPS [106], Street View House Numbers (SVHN) [156], CIFAR [117], and STL [44], see 7.1 a detailed summary. The task is to perform cross-domain recognition: *taking the training set from one dataset as the source domain and the test set from another dataset as the target domain.* We evaluate our algorithm's recognition accuracy over three cross-domain pairs: 1) MNIST vs USPS, 2) SVHN vs MNIST, and 3) CIFAR vs STL.

Table 7.1: Summary of five benchmark datasets.

| Dataset | Type | #train | #test | Dimension |
|---|---|---|---|---|
| MNIST | Digit (grayscale) | $50,000$ | $10,000$ | $28 \times 28$ |
| USPS | Digit (grayscale) | $7,291$ | $2,007$ | $16 \times 16$ |
| SVHN | Digit (RGB) | $73,257$ | $26,032$ | $32 \times 32$ |
| CIFAR-10 | Object (RGB) | $50,000$ | $10,000$ | $32 \times 32$ |
| STL-10 | Object (RGB) | $5,000$ | $8,000$ | $96 \times 96$ |

**Data setup:** MNIST (MN) vs USPS (US) contains 2D grayscale handwritten digit images of 10 classes. We preprocessed them as follows. USPS images were rescaled into $28 \times 28$ and pixels were normalized to $[0, 1]$ values. From this pair, two cross-domain recognition tasks were performed: MN $\rightarrow$ US and US $\rightarrow$ MN.

In SVHN (SV) vs MNIST (MN) pair, MNIST images were rescaled to $32 \times 32$ and SVHN images were grayscaled. The $[0, 1]$ normalization was then applied to all images. Note that we did not preprocess SVHN images using local contrast normalization as in [194]. We evaluated our algorithm on SV $\rightarrow$ MN and MN $\rightarrow$ SV cross-domain recognition tasks.

STL (ST) vs CIFAR (CI) consists of RGB images that share eight object classes: *airplane*, *bird*, *cat*, *deer*, *dog*, *horse*, *ship*, and *truck*, which forms $4,000$ (train) and $6,400$ (test) images for STL, and $40,000$ (train) and $8,000$ (test) images for CIFAR. STL images were rescaled to $32 \times 32$ and pixels were standardized into zero-mean and unit-variance. Our algorithm was evaluated on two cross-domain tasks, that is, ST $\rightarrow$ CI and CI $\rightarrow$ ST.

**The archiceture and learning setup.** The DRCN architecture used in the experiments is adopted from [141]. The label prediction pipeline has three convolutional layers: 100 5x5 filters (CONV1), 150 5x5 filters (CONV2), and 200 3x3 filters (CONV3) respectively, two max-pooling layers of size 2x2 after the first and the second convolutional layers (POOL1 and POOL2), and three fully-connected layers (FC4, FC5,and FC_OUT) – FC_OUT is the output layer. The number of neurons in FC4 or FC5 was treated as a tunable hyper-parameter in the range of $[300, 350, ..., 1000]$, chosen according to the best performance on the (source) validation set. The shared encoder $g_{\text{enc}}$ has thus a configuration of CONV1-POOL1-CONV2-POOL2-CONV3-FC4-FC5. The configuration of the decoder $g_{\text{dec}}$ is the inverse of that of $g_{\text{enc}}$.

We employ ReLU activations [153] in all hidden layers and linear activations in the output layer of the reconstruction pipeline. Updates in both classification and reconstruction tasks were computed via RMSprop with learning rate of $10^{-4}$ and moving average decay of 0.9. The control penalty $\lambda$ was selected according to accuracy on the source validation data – typically, the optimal value was in the range $[0.4, 0.7]$.

**Benchmark algorithms.** We compare DRCN with the following algorithms.

1. ConvNet$_{src}$: a supervised convolutional network trained on the labeled source domain only, with the same network configuration as that of DRCN's label prediction pipeline,
2. SCAE: ConvNet preceded by the layer-wise pretraining of stacked convolutional autoencoders on all unlabeled data [141],
3. SCAE$_t$: similar to SCAE, but only unlabeled data from the target domain are used during pretraining,
4. SDA$_{sh}$ [79]: the deep network with three fully connected layers, which is a successful domain adaptation model for sentiment classification,
5. Subspace Alignment (SA) [67],[2] and
6. ReverseGrad [72]: a recently published domain adaptation model based on deep convolutional networks that provides the state-of-the-art performance.

All deep learning based models above have the same architecture as DRCN for the label predictor. For ReverseGrad, we also evaluated the "original architecture" devised in [72] and chose whichever performed better of the original architecture or our architecture. Finally, we applied the data augmentation to all models similarly to DRCN. The ground-truth model is also evaluated, that is, a convolutional network trained from and tested on images from the target domain only (ConvNet$_{tgt}$), to measure the difference between the cross-domain performance and the ideal performance.

[2]The setup follows one in [72]: the inputs to SA are the last hidden layer activation values of ConvNet$_{src}$.

Table 7.2: Accuracy (*mean* ± *std* %) on six cross-domain recognition tasks over ten independent runs. Bold-red and bold-black indicate the best and second best domain adaptation performance. ConvNet$_{tgt}$ denotes the ground-truth model: training and testing on the target domain only.

| Methods | MN → US | US → MN | SV → MN | MN → SV |
|---|---|---|---|---|
| ConvNet$_{src}$ | 85.55 ± 0.12 | 65.77 ± 0.06 | 62.33 ± 0.09 | 25.95 ± 0.04 |
| SDA$_{sh}$ [79] | 43.14 ± 0.16 | 37.30 ± 0.12 | 55.15 ± 0.08 | 8.23 ± 0.11 |
| SA [67] | 85.89 ± 0.13 | 51.54 ± 0.06 | 63.17 ± 0.07 | 28.52 ± 0.10 |
| SCAE [141] | 85.78 ± 0.08 | 63.11 ± 0.04 | 60.02 ± 0.16 | 27.12 ± 0.08 |
| SCAE$_t$ [141] | 86.24 ± 0.11 | 65.37 ± 0.03 | 65.57 ± 0.09 | 27.57 ± 0.13 |
| ReverseGrad [72] | **91.11 ± 0.07** | **74.01 ± 0.05** | **73.91 ± 0.07** | **35.67 ± 0.04** |
| **DRCN** | **91.80 ± 0.09** | **73.67 ± 0.04** | **81.97 ± 0.16** | **40.05 ± 0.07** |
| ConvNet$_{tgt}$ | 96.12 ± 0.07 | 98.67 ± 0.04 | 98.67 ± 0.04 | 91.52 ± 0.05 |

**Classification accuracy.** Tables 7.2 and 7.3 summarize the cross-domain recognition accuracy (*mean* ± *std*) of all algorithms over ten independent runs. DRCN performs best in all but one cross-domain tasks, better than the prior state-of-the-art ReverseGrad. Notably on the SV → MN task, DRCN outperforms ReverseGrad with ∼ 8% accuracy gap. DRCN also provides a considerable improvement over ReverseGrad (∼ 5%) on the reverse task, MN → SV, but the gap to the groundtruth is still large – this case was also mentioned in previous work as a failed case [72]. In the case of CI → ST, the performance of DRCN almost matches the performance of the target baseline.

DRCN also convincingly outperforms the greedy-layer pretraining-based algorithms (SDA$_{sh}$, SCAE, and SCAE$_t$). This indicates the effectiveness of the simultaneous reconstruction-classification training strategy over the standard pretraining-finetuning in the context of domain adaptation

**ReverseGrad vs DRCN *t*-tests.** For completeness, we perform a statistical significance test to measure how significance the average accuracy difference between ReverseGrad and DRCN is. Welch's *t*-test [230] is utilized to perform the test, since we assume that the accuracy variances of ReverseGrad and DRCN are unequal, Table 7.4 shows the significance test results for each cross-domain case.

Picking a significance level at 0.01 and observing the obtained $P$-values, we reject the null hypothesis that the average accuracy scores between ReverseGrad

Table 7.3: Accuracy (*mean* ± *std* %) on six cross-domain recognition tasks – cont'd.

| Methods | ST → CI | CI → ST |
|---|---|---|
| ConvNet$_{src}$ | 54.17 ± 0.21 | 63.61 ± 0.17 |
| SDA$_{sh}$ [79] | 35.82 ± 0.07 | 42.27 ± 0.12 |
| SA [67] | 54.04 ± 0.19 | 62.88 ± 0.15 |
| SCAE [141] | 54.25 ± 0.13 | 62.18 ± 0.04 |
| SCAE$_t$ [141] | 54.68 ± 0.08 | 61.94 ± 0.06 |
| ReverseGrad [72] | **56.91 ± 0.05** | **66.12 ± 0.08** |
| **DRCN** | **58.86 ± 0.07** | **66.37 ± 0.10** |
| ConvNet$_{tgt}$ | 78.81 ± 0.11 | 66.50 ± 0.07 |

Table 7.4: Welch's *t*-test on the performance of ReverseGrad and DRCN.

| Tasks | $t$ value | d.f. | $P$ value |
|---|---|---|---|
| MN → US | −19.137 | 16.971 | $6.341 \times 10^{-13}$ |
| US → MN | 16.791 | 17.173 | $4.305 \times 10^{-12}$ |
| SV → MN | −145.944 | 12.324 | $2.493 \times 10^{-21}$ |
| MN → SV | −171.798 | 14.311 | $3.970 \times 10^{-25}$ |
| ST → CI | −71.683 | 16.287 | $8.479 \times 10^{-22}$ |
| CI → ST | −6.173 | 17.173 | $9.763 \times 10^{-6}$ |

and DRCN are equal. In other words, we can safely conclude that DRCN performs better than ReverseGrad at the 0.01 significance level.

**Comparison of different DRCN flavors.** Recall that DRCN uses only the unlabeled target images for the unsupervised reconstruction training. To verify the importance of this strategy, we further compare different flavors of DRCN: DRCN$_s$ and DRCN$_{st}$. Those algorithms are conceptually the same but different only in utilizing the unlabeled images during the unsupervised training. DRCN$_s$ uses only unlabeled source images, whereas DRCN$_{st}$ combines both unlabeled source and target images.

The experimental results in Table 7.5 confirm that DRCN always performs better than DRCN$_s$ and DRCN$_{st}$. While DRCN$_{st}$ occasionally outperforms ReverseGrad, its overall performance does not compete with that of DRCN. The only case where DRCN$_s$ and DRCN$_{st}$ flavors can closely match DRCN is on MN→ US. This suggests that the use of *unlabeled source data* during the reconstruction

Table 7.5: Accuracy (%) of DRCN$_s$ and DRCN$_{st}$. Bold indicates the best performance.

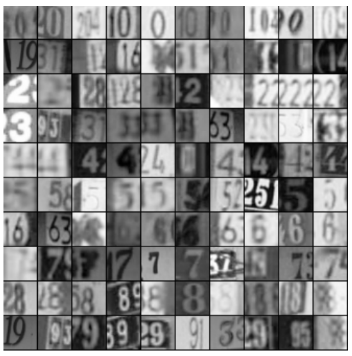
| Methods | MN → US | US → MN | SV → MN | MN → SV | ST → CI | CI → ST |
|---|---|---|---|---|---|---|
| **DRCN$_s$** | 89.92 ± 0.12 | 65.96 ± 0.07 | 73.66 ± 0.04 | 34.29 ± 0.09 | 55.12 ± 0.12 | 63.02 ± 0.06 |
| **DRCN$_{st}$** | 91.15 ± 0.05 | 68.64 ± 0.05 | 75.88 ± 0.09 | 37.77 ± 0.06 | 55.26 ± 0.06 | 64.55 ± 0.13 |
| **DRCN** | **91.80 ± 0.09** | **73.67 ± 0.04** | **81.97 ± 0.16** | **40.05 ± 0.07** | **58.86 ± 0.07** | **66.37 ± 0.10** |

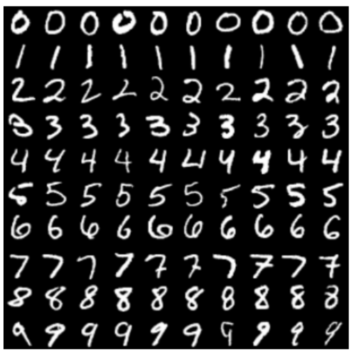
training do not contribute much to the cross-domain generalization, which verifies the DRCN strategy in using the unlabeled target data only.

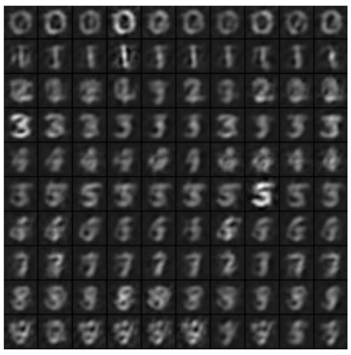
**Data reconstruction.** A useful insight was found when reconstructing source images through the reconstruction pipeline of DRCN. Specifically, we observe the visual appearance of $f_r(x_1^s), \ldots, f_r(x_m^s)$, where $x_1^s, \ldots, x_m^s$ are some images from the source domain. Note that $x_1^s, \ldots, x_m^s$ are unseen during the unsupervised reconstruction training in DRCN. We visualize such a reconstruction in the case of SV →MN training in Figure 7.2. Figure 7.2(a) and 7.3(a) display the original source (SVHN) and target (MNIST) images.

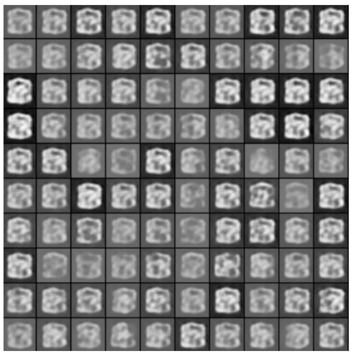
The main finding of this observation is depicted in Figure 7.3(c): the reconstructed images produced by DRCN given some SVHN images as the source inputs. We found that *the reconstructed SVHN images resemble MNIST-like digit appearances, with white stroke and black background*, see Figure 7.3(a). Remarkably, DRCN still can produce “correct" reconstructions of some noisy SVHN images. For example, all SVHN digits 3 displayed in Figure 7.2(a) are clearly reconstructed by DRCN, see the fourth row of Figure 7.3(c). DRCN seems to pick only the digit in the middle and ignore the remaining digits. This may explain the superior cross-domain recognition performance of DRCN on this task. However, such a cross-reconstruction appearance does not happen in the reverse task, MN → SV, which may be an indicator for the low accuracy relative to the groundtruth performance.

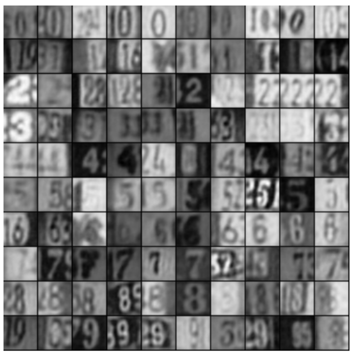
We also conduct such a diagnostic reconstruction on other algorithms that have the reconstruction pipeline. Figure 7.3(d) depicts the reconstructions of the SVHN images produced by ConvAE trained on the MNIST images only. They do not appear to be digits, suggesting that ConvAE recognizes the SVHN images as noise. Figure 7.3(e) shows the reconstructed SVHN images produced by DRCN$_{st}$.


(a) Source (SVHN)


(b) Target (MNIST)


(c) DRCN


(d) ConvAE


(e) DRCN$_{st}$

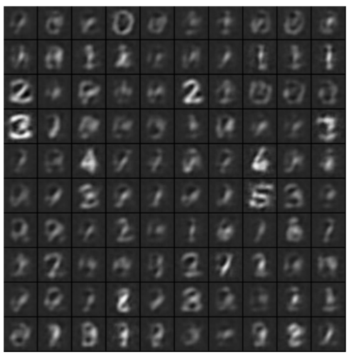
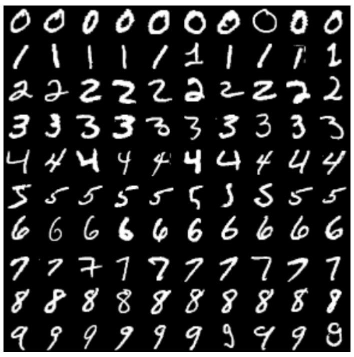
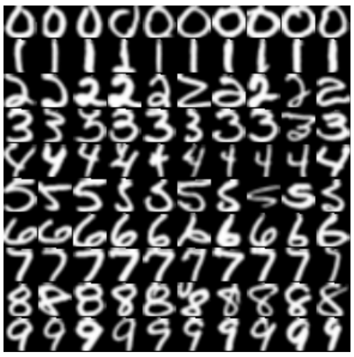
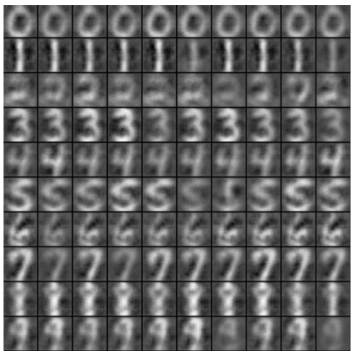
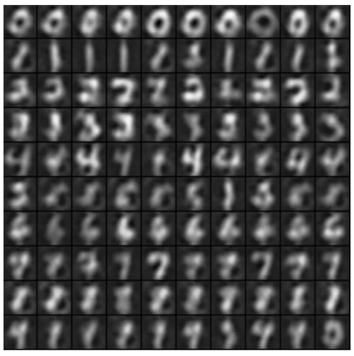

(f) ConvAE+ConvNet

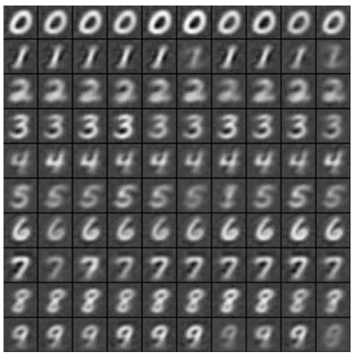
Figure 7.2: Data reconstruction after training from SVHN $\rightarrow$ MNIST. Fig. (a)-(b) show the original input pixels, and (c)-(f) depict the reconstructed source images (SVHN). The reconstruction of DRCN appears to be MNIST-like digits, see the main text for a detailed explanation.


(a) Source (MNIST)


(b) Target (USPS)


(c) DRCN


(d) ConvAE


(e) DRCN$_{st}$

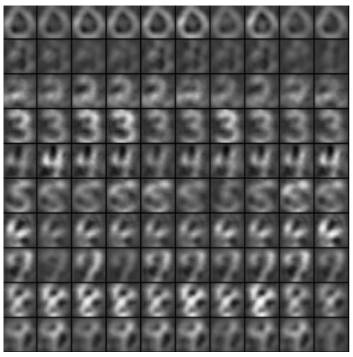
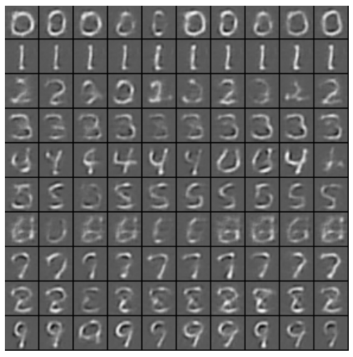
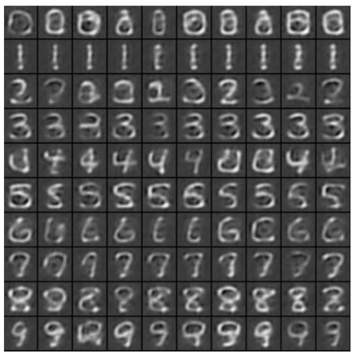

(f) ConvAE+ConvNet$_{src}$

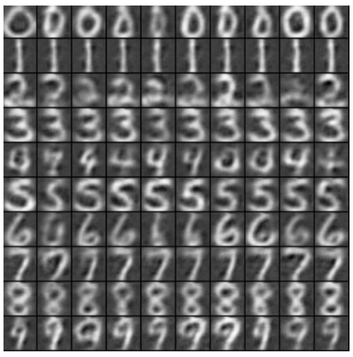
Figure 7.3: Data reconstruction after training from MNIST $\rightarrow$ USPS. Fig. (a)-(b) show the original input pixels, and (c)-(f) depict the reconstructed source images (MNIST). The reconstruction of DRCN appears to be USPS-like digits.

(a) Source (USPS)

(b) Target (MNIST)


(c) DRCN


(d) ConvAE


(e) DRCN$_{st}$

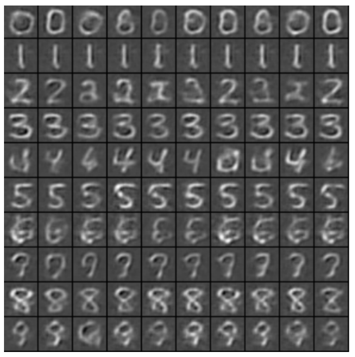
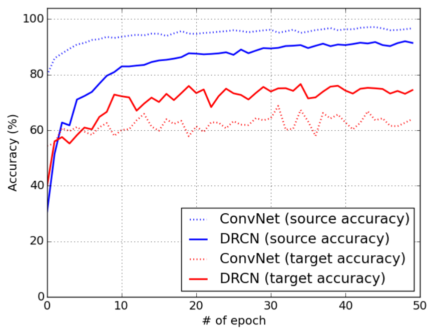

(f) ConvAE+ConvNet$_{src}$

Figure 7.4: Data reconstruction after training from USPS → MNIST. Fig. (a)-(b) show the original input pixels, and (c)-(f) depict the reconstructed source images (USPS). The reconstruction of DRCN appears to be MNIST-like digits.

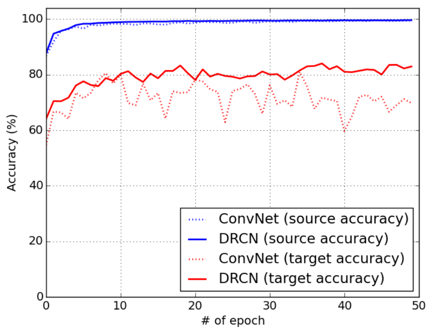
We can see that they look almost identical to the source images shown in Figure 7.2(a), which is not surprising since the source images are included during the reconstruction training.

Finally, we evaluated the reconstruction induced by ConvNet$_{src}$ to observe the difference with the reconstruction of DRCN. Specifically, we trained ConvAE on the MNIST images in which the encoding parameters were initialized from those of ConvNet$_{src}$ and not updated during training. We refer to the model as ConvAE+ConvNet$_{src}$. The reconstructed images are visualized in Figure 7.3(f). Although they resemble the style of MNIST images as in the DRCN's case, only a few source images are correctly reconstructed.

To summarize, the results from this diagnostic data reconstruction correlate with the cross-domain recognition performance. A similar trend of such outcome can be found in the case of MNIST vs USPS, see Figures 7.3 and 7.4.



(a) SVHN → MNIST training



(b) MNIST → USPS training

Figure 7.5: The source accuracy (blue lines) and target accuracy (red lines) comparison between ConvNet and DRCN during training stage on SVHN → MNIST cross-domain task. DRCN induces lower source accuracy, but higher target accuracy than ConvNet.

**Training progress.** Recall that DRCN has two pipelines with shared parameters; each corresponds to the classification and reconstruction task, respectively. One can consider that the unsupervised reconstruction learning acts as a regularization for the supervised classification to reduce overfitting onto the source domain. Figure 7.5 compares the source and target accuracy of DRCN with that of the standard ConvNet during training. The most prominent results indicating the overfitting reduction can be seen in the SVHN $\rightarrow$ MNIST case, *i.e.*, DRCN produces higher target accuracy but with lower source accuracy, than ConvNet. In the case of MNIST $\rightarrow$ USPS, the behavior is even "better" in the sense that DRCN not only produces better performance on the target dataset, but also does not degrade the performance on the source dataset.

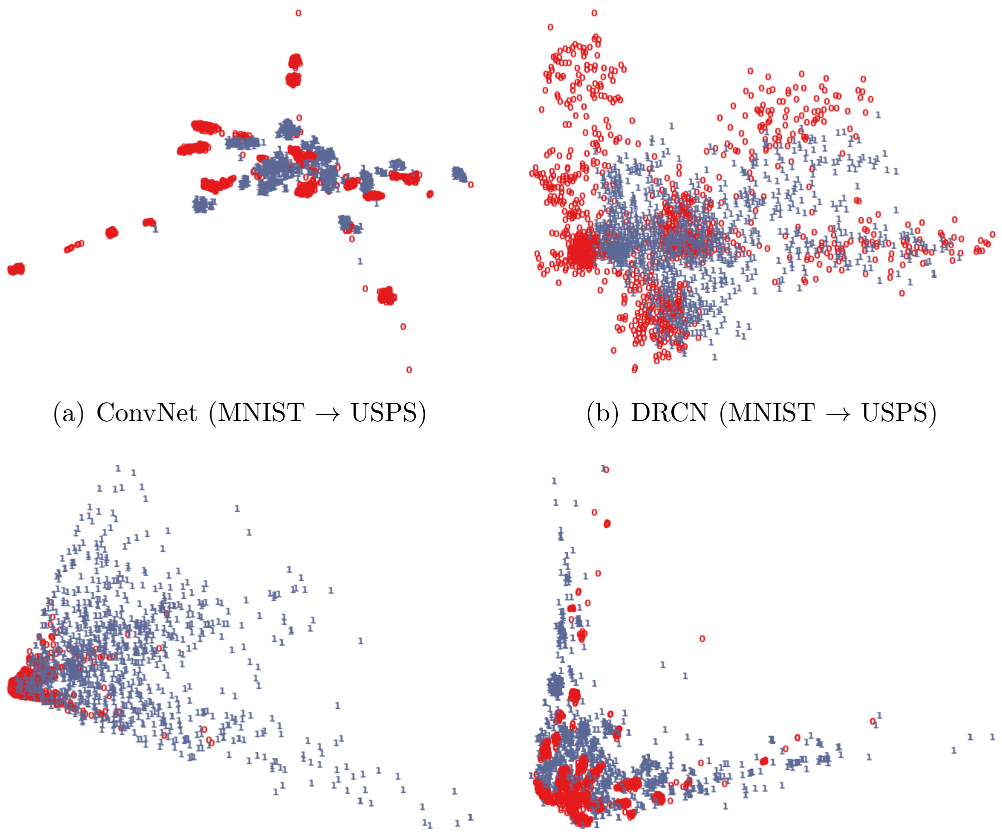
**t-SNE Visualization.** For completeness, we also visualize the 2D point cloud of the DRCN$'s$ last hidden layer activations using t-SNE [219] and compare it with that of the standard ConvNet. Figure 7.6 depicts the empirical feature distribution comparison between ConvNet (with no adaption) and DRCN in the cases of MNIST $\rightarrow$ USPS and SVHN $\rightarrow$ MNIST. Red and gray point clouds indicate the source and target feature clouds, respectively. A successful domain adaptation should be indicated by a datapoint configuration that the source and target point clouds should overlap each other. We can see that DRCN provides more apparent overlapping point clouds than those of the standard ConvNet.

### 7.3.2 Experiments II: Office dataset

In the second experiment, we evaluated DRCN on the standard domain adaptation benchmark for visual object recognition, OFFICE [187], which consists of three different domains: AMAZON (A), DSLR (D), and WEBCAM (W). OFFICE has 2817 labeled images in total distributed across 31 object categories. The number of images is thus relatively small compared to the previously used datasets.

We applied the DRCN algorithm to *finetune* AlexNet [118], as was done with different methods in previous work [72, 132, 217].[3] The fine-tuning was performed only on the fully connected layers of AlexNet, $fc6$ and $fc7$, and the last convolutional layer, *conv*5. Specifically, the label prediction pipeline of DRCN contains $conv4$-$conv5$-$fc6$-$fc7$-$label$ and the data reconstruction pipeline has $conv4$-$conv5$-$fc6$-$fc7$-$fc6'$-$conv5'$-$conv4'$ (the $'$ denotes the the inverse layer) – it thus does not reconstruct the original input pixels. The learning rate was selected following the

[3]Recall that AlexNet consists of five convolutional layers: $conv1, \ldots, conv5$ and three fully connected layers: $fc6, fc7$, and $fc8/output$.



(a) ConvNet (MNIST → USPS)

(b) DRCN (MNIST → USPS)

(c) ConvNet (SVHN → MNIST)

(d) DRCN (SVHN → MNIST)

Figure 7.6: t-SNE visualizations of the last layer's activations. Red and gray points indicate the source and target domain examples, respectively.

strategy devised in [132]: cross-validating the base learning rate between $10^{-5}$ and $10^{-2}$ with a multiplicative step-size $10^{1/2}$. We also tried the fine-tuning strategy proposed in [132] on DRCN: fixing *conv*1-*conv*3, fine-tuning *conv*4 and *conv*5, learning $fc6 - fc8$ from scratch. However, we dd not obtain a competitive performance for DRCN in doing so.

We followed the standard unsupervised domain adaptation training protocol used in previous work [41, 81, 132], that is, using *all* labeled source data and unlabeled target data. Table 7.6 summarizes the performance accuracy of DRCN based on that protocol in comparison to the state-of-the-art algorithms. We found that

DRCN is competitive against DAN and ReverseGrad – the performance is either the best or the second best except for one case. In particular, DRCN performs best with a convincing gap in situations when the target domain has relatively many data, *i.e.*, AMAZON as the target dataset. We hypothesize that the performance of DRCN could be further improved by providing a larger amount of unlabeled images from, *e.g.*, ImageNet, for data reconstruction training, which will be our future work.

Table 7.6: Accuracy (*mean* ± *std* %) on the Office dataset with the standard unsupervised domain adaptation protocol used in [41, 81].

| Method | A → W | W → A | A → D | D → A | W → D | D → W |
|---|---|---|---|---|---|---|
| DDC [217] | 61.8 ± 0.4 | 52.2 ± 0.4 | 64.4 ± 0.3 | 52.1 ± 0.8 | 98.5 ± 0.4 | 95.0 ± 0.5 |
| DAN [132] | 68.5 ± 0.4 | **53.1 ± 0.3** | **67.0 ± 0.4** | 54.0 ± 0.4 | **99.0 ± 0.2** | **96.0 ± 0.3** |
| ReverseGrad [72] | **72.6 ± 0.3** | 52.7 ± 0.2 | **67.1 ± 0.3** | **54.5 ± 0.4** | **99.2 ± 0.3** | **96.4 ± 0.1** |
| **DRCN** | **68.7 ± 0.3** | **54.9 ± 0.5** | 66.8 ± 0.5 | **56.0 ± 0.5** | **99.0 ± 0.2** | **96.4 ± 0.3** |

## 7.4 Analysis

This section shows that, under certain assumptions, optimizing (7.5) in DRCN is approximately equivalent to solving a semi-supervised learning problem on the target domain. The theoretical result also yields a consequence that the unsupervised training using unlabeled target data only is sufficient – the unlabeled source data might not help further improve domain adaptation.

Denote the labeled and unlabeled distributions as $\mathbb{P}_{XY} =: \mathbb{P}$ and $\mathbb{P}_X$ respectively. Let $P^\theta(\cdot)$ refer to a family of models, parameterized by $\theta \in \Theta$, that is used to learn a maximum likelihood estimator. The DRCN's learning algorithm for domain adaptation tasks can be interpreted probabilistically by assuming that $P^\theta(x)$ is Gaussian and $P^\theta(y|x)$ is a multinomial, fit by logistic regression.

The objective in Eq.(7.5) is then equivalent to the following (empirical) maximum likelihood estimate:

$$\hat{\theta}_\lambda = \underset{\theta}{\operatorname{argmax}}\, \lambda \sum_{i=1}^{n_s} \log P^\theta_{Y|X}(y_i^s|x_i^s) + (1-\lambda) \sum_{j=1}^{n_t} \log P^\theta_{X|\tilde{X}}(x_j^t|\tilde{x}_j^t), \tag{7.6}$$

where $\tilde{x}$ is the noisy input generated from $\mathbb{Q}_{\tilde{X}|X}$. The first term represents the modeled learned by the supervised convolutional network and the second term represents the model learned by the unsupervised convolutional autoencoder. Note that the discriminative model only observes labeled data from the source distribution $\mathbb{P}_X$ in objectives (7.5) and (7.6).

We now consider a semi-supervised learning problem formulated in [46]. Suppose that labeled and unlabeled samples are taken from the *target domain* with probabilities $\lambda$ and $(1-\lambda)$ respectively. By Theorem 5.1 in [46], the maximum likelihood estimate $\theta_\lambda^t$ is

$$\theta_\lambda^t = \underset{\theta}{\operatorname{argmax}} \, \lambda \underset{\mathbb{P}^t}{\mathbb{E}}[\log P^\theta(x,y)] + (1-\lambda) \underset{\mathbb{P}_X^t}{\mathbb{E}}[\log P_X^\theta(x)] \tag{7.7}$$

Unfortunately, $\theta_\lambda^t$ cannot be computed since we do not have access to target labels. As a proxy, we consider the maximum likelihood estimate in Eq. (7.6) that is computed from a mixture of source and target samples.

Theorem 18 shows the two objectives are approximately equivalent. We make the following assumptions:

(A1) *Consistency:* The model contains true distribution, so the MLE is consistent.

(A2) *Smoothness and measurability:* [233]

(A3) *Covariate shift:* $\mathbb{P}_{Y|X}^s = \mathbb{P}_{Y|X}^t$ [196].

(A4) *Constant ratio:* there is a constant $\alpha > 0$ such that $\frac{\mathbb{P}_X^t(x^s)}{\mathbb{P}_X^s(x^s)} \approx \alpha$ for all source samples $x^s \sim \mathbb{P}_X^s$.

Consistency and smoothness are standard assumptions used by Cohen and Cozman [46] to prove their result. Covariate shift is a common assumption in domain adaptation. It says, roughly, that only the distribution of data changes across domains – *i.e.*the labels are chosen by a *fixed* unknown function of the input data (which may be stochastic).

The constant ratio assumption, A4, is new and requires some elucidation. It states that the same relative probability is assigned to source samples by the source and target domains. Note that the assumption is consistent with the source domain assigning much more probability mass on every source sample than the mass placed by the target domain.

**Theorem 18** (reduction from domain adaptation $\rightarrow$ semi-supervised learning)**.** *Under the assumptions above there exists a $\lambda'$, not necessarily equal to $\lambda$, such that the source-target mixture in Eq.* (7.6) *approximates the purely target-based estimator defined in Eq.* (7.7)*:*

$$\hat{\theta}_{\lambda'} \approx \theta^t_\lambda \quad \textit{for some } \lambda'.$$

In experiments, the parameter $\lambda'$ is chosen by cross-validation with respect to classification performance on the source domain.

*Proof.* If $\mathbb{P}^s$ and $\mathbb{P}^t$ satisfy covariate shift, A3, then we can switch from an expectation over target samples to source samples:

$$\mathop{\mathbb{E}}_{\mathbb{P}^t}\left[\log P^\theta(x,y)\right] = \mathop{\mathbb{E}}_{\mathbb{P}^s}\left[\frac{\mathbb{P}^t_X(x)}{\mathbb{P}^s_X(x)} \cdot \log P^\theta(x,y)\right].$$

It follows that $\theta^t_\lambda$ can be empirically estimated as

$$\hat{\theta}^t_\lambda \approx \operatorname*{argmax}_\theta \lambda \sum_{i=1}^{n_s} \frac{\mathbb{P}^t_X(x^s_i)}{\mathbb{P}^s_X(x^s_i)} \log P^\theta(x^s_i, y^s_i) + (1-\lambda) \sum_{j=1}^{n_t} \log P^\theta_X(x^t_j) \tag{7.8}$$

It was shown in [23] that $P^\theta_{X|\tilde{X}}(x|\tilde{x})$ defines an ergodic Markov chain whose asymptotic marginal distribution of $X$ converges to the data-generating distribution $\mathbb{P}_X$. Thus, $\max P^\theta_{X|\tilde{X}}(x|\tilde{x}) \approx \max P^\theta_X(x)$.

Finally, assumption A4 allows to substitute a constant for the ratio $\frac{\mathbb{P}^t_X(x)}{\mathbb{P}^s_X(x)}$ and the result follows. ☐

The above theorem can be used to explain the outcome that the unlabeled samples from the source domain might not further contribute to domain adaptation in our setting. The first term of (7.8) can be written as

$$\lambda \sum_{i=1}^{n_s} \frac{\mathbb{P}^t_X(x^s_i)}{\mathbb{P}^s_X(x^s_i)} \log P^\theta_{Y|X}(y^s_i|x^s_i) + \lambda \sum_{i=1}^{n_s} \frac{\mathbb{P}^t_X(x^s_i)}{\mathbb{P}^s_X(x^s_i)} \log P^\theta_X(x^s_i).$$

Observe the second term above. As $n_s \rightarrow \infty$, $P^\theta_X$ will converge to $\mathbb{P}^s_X$. Hence, $\int_{x\sim\mathbb{P}^s_X} \frac{\mathbb{P}^t_X(x)}{\mathbb{P}^s_X(x)} \log \mathbb{P}^s_X(x) \leq \int_{x\sim\mathbb{P}^s_X} \mathbb{P}^t_X(x)$, which means that adding more unlabeled source data will only result in a constant. This implies an optimization procedure equivalent to (7.6), which justifies the *uselessness* of unlabeled source samples.

Note that the latter analysis does not necessarily imply that the incorporation of unlabeled source data degrades the performance. The fact that DRCN$_{st}$ performs worse than DRCN could be due to, e.g., the model capacity, which depends on the choice of the architecture.

## 7.5 Chapter Summary

This chapter presents a new deep multitask learning algorithm with convolutional architecture for unsupervised domain adaptation in visual object recognition. The algorithm learns two different tasks simultaneously: i) supervised source label prediction and ii) unsupervised target image reconstruction. The resulting deep model is referred to as Deep Reconstruction-Classification Network (DRCN), which is equipped with two pipelines with a shared encoding representation corresponding to the above tasks. We show that DRCN performs better than the standard ConvNet with traditional *unsupervised pretraining-supervised finetuning* strategy and provides the state-of-the-art performance on cross-domain object recognition tasks.

A useful insight into the effectiveness of DRCN can be found when observing the visual characteristic of the DRCN's data reconstruction. That is, the appearance of DRCN's reconstructed source images resemble that of the target images, which indicates that DRCN learns the domain correspondence without knowing the target labels. In the case of the SVHN $\rightarrow$ MNIST task, for example, the reconstructed images of the SVHN images appear to be similar to the MNIST images in terms of stroke style and background.

An observation of the performance of DRCN and the standard ConvNet over the training iterations shows that the unsupervised reconstruction task may act as a regularization for the supervised classification to reduce overfitting onto the source domain. DRCN provides lower or similar training/source accuracy and higher test/target accuracy than ConvNet.

This chapter also establishes the theoretical soundness of the DRCN's algorithm. We found that training DRCN is approximately equivalent to solving a semi-supervised learning problem on the target domain. Furthermore, the theoretical outcome also yields a consequence that the use of unlabeled target data only during the unsupervised reconstruction learning is sufficient – adding unlabeled source data might not help to further improve domain adaptation.

One direction for future work is a rigorous analysis of the relationship between multitask learning and uncovering domain correspondence. A further study about the role of data augmentation and denoising is also a potential direction. Results from the above directions would help design better domain adaptation algorithms in the future.

# 8 Conclusions and Future Work

This chapter concludes the thesis and discusses several possible directions for future work. The overall goal of this thesis was to progress towards solving domain shift / dataset bias in visual object recognition via representation learning in the context of domain adaptation and domain generalization. This goal was successfully achieved by developing a number of state-of-the-art domain adaptation and domain generalization algorithms based on deep learning and kernel methods. Our proposed algorithms were evaluated over a wide range of cross-domain object recognition tasks and provided either competitive or better performance accuracy than prior state-of-the-art methods. We also presented theoretical analyses for two of the proposed algorithms to justify the soundness of the algorithms.

The remainder of this chapter provides conclusions for each objective of this thesis and highlights the main findings from each individual contribution chapter. We then discuss some potential research directions for future work.

## 8.1 Achieved Objectives

The following research objectives have been achieved in this thesis:

- Develop a new deep learning algorithm that can reduce *dataset bias* induced by "clean" training images and "noisy" test images. The algorithm trains deep neural nets with a (sparse) autoencoder (SAE) in the first layer and restricted Boltzmann machines (RBMs) in the upper hidden layers on clean images. SAE is utilized to extract sparse features from raw images, which are expected to be *noise-invariant* over several types of corruption such as Gaussian noise, impulse noise, pixel border, local block, and background. The stacked RBMs then receive the sparse features as inputs to learn the top hierarchical features. The use of RBMs is motivated by the fact that the stacked RBMs generalize deep neural net in the standard evaluation setting, *i.e.*, "clean" training images and "clean" test images. We refer to the resulting model as Deep Hybrid Network (DHN). The evaluation results show that a DHN trained on clean handwritten digits is more robust in recognizing noisy handwritten digits than a Deep Belief Network (DBN) trained on the same examples. We also propose a flavor of DHN, referred to as sparse Deep Hybrid Network (sDHN) that is equipped with sparse, local connections in the first hidden layer. sDHN improves the robustness of DHN against local block noise.

- Develop a domain adaptation model, referred to as Domain Adaptive Neural Network (DaNN), which is based on feedforward neural networks trained using a regularized back-propagation algorithm. We propose a new regularization technique that controls the distribution difference between the source and target data hidden layer activations. The regularization is achieved by utilizing Maximum Mean Discrepancy (MMD) as an additional term in the DaNN's objective function. MMD measures the distribution mismatch between the source and target data hidden activations, which is minimized during the back-propagation training. From evaluations over a range of cross-domain object recognition tasks using the Office dataset, DaNN performs better than the standard feedforward neural nets and competitively to the prior state-of-the-art, Transfer Sparse Coding (TSC) [133]. When preceded by the denoising autoencoder pretraining, DaNN provides the best performance on raw pixels.

- Develop a novel multi-task representation learning algorithm for domain generalization, which we refer to as *Multi-task Autoencoder* (MTAE). MTAE extends the standard denoising autoenoder algorithm by substituting artificially induced corruption with naturally occurring inter-domain variability in the appearances of objects. Instead of reconstructing images from noisy versions, MTAE learns to transform the original image into its analogs in multiple related domains. It thereby learns features that are robust to variations across domains. We evaluate the cross-domain object recognition performance of MTAE on a range of benchmark datasets: the MNIST, USPS, Office, VOC, LabelMe, Caltech, and SUN09. We found that (denoising) MTAE outperforms alternative autoencoder-based models as well as the current state-of-the-art algorithms for domain generalization in terms of accuracy. From the visualization of the MTAE's learned parameters, it suggests that MTAE captures the underlying *object transformation* among source domains, which might be the reason of its effectiveness.

- Develop a fast, simple, and theoretically well founded representation learning algorithm based on the *kernel trick*, which can be used for both domain adaptation and domain generalization. SCA is built from a simple geometrical measure operating on reproducing kernel Hilbert space, *i.e.*, *scatter*. SCA seeks a representation of data that satisfies four requirements: i) separate datapoints with different labels and ii) separate the data as a whole, while iii) not separating points sharing the same label and iv) not separating the two or more domains. The optimization problem of SCA can be reduced to a generalized eigenvalue problem, which results in a fast and exact solution. Comprehensive experiments on benchmark cross-domain object recognition datasets verify that SCA performs much faster than several state-of-the-art algorithms, while provides competitive classification accuracy in both domain adaptation and domain generalization. We also show that *scatter* with two domains as inputs relates to *maximum mean discrepancy* [28] and *discrepancy distance* [140], which implies generalization bound for SCA in the domain adaptation case.

- Develop a novel deep convolutional neural network for unsupervised domain adaptation trained by multi-task learning. That is, the proposed algorithm simultaneously learns two tasks via a shared encoding representation: i) *source* label classification and ii) *target* data reconstruction. The resulting model is referred to as Deep Reconstruction-Classification Network (DRCN), which is equipped with a *label prediction pipeline* and an *data reconstruction pipeline* corresponding to the above tasks. From evaluations on a series of cross-domain object recognition tasks, the DRCN's label prediction pipeline provides better performance accuracy than the prior state-of-the-art Domain-Adversarial Neural Network (DANN) in almost all cases. An observation from the outputs of the DRCN's reconstruction pipeline suggests that DRCN successfully captures the domain correspondence – the reconstructed source images appear to be similar to the original target images. We also demonstrate that the DRCN's learning objective is approximately equivalent to solving a semi-supervised learning problem on the target domain.

## 8.2 Main Conclusions

Overall, this thesis finds that the representation learning approach based on deep learning and kernel methods is effective to mitigate the *dataset bias* problem, which improves the object recognition performance under domain adaptation or domain generalization settings. Most of our proposed algorithms successfully provide better object recognition performance than the prior state-of-the-art algorithms.

This section discusses the main conclusions for each research objective drawn from the five contribution chapters (Chapters 3 to 7).

### 8.2.1 Sparse Features and Sparse Connections in Deep Neural Networks for Domain Adaptation

Chapter 3 presents Deep Hybrid Network (DHN), our proposed algorithm based on a particular combination of a sparse autoencoder (AE) and stacked restricted Boltzmann machines (RBMs) as the pretraining models; the sparse AE trains the first hidden layer, while the RBMs train the top remaining layers. It also presents the sparsely-connected Deep Hybrid Network (sDHN), the variant of DHN with local sparse connections in the first hidden layer. We compared the performance of the proposed algorithms with Deep Belief Network (DBN) [97], sparsely-connected Deep Belief Network (sDBN) [209], and stacked sparse Autoencoders (SAE) [157].

Below we summarize our main findings in terms of two sparsity notions: i) sparse features and i) sparse local weight connections.

## Sparse Features

Chapter 3 finds that sparse hidden layer features induced by DHN or SAE can significantly reduce *dataset bias* in the *cross-domain* object recognition tasks with clean source images and noisy target images. This effect is more prominent when the models are trained on a small sized dataset. DHN and SAE are particularly robust against impulse noise, pixel border occlusion, and background noise, performs significantly better than DBN. Furthermore, DHN does not perform much worse than DBN in the standard *in-domain* setting: clean source images and clean target images.

We observe that the sparse autoencoder-RBMs combination, *i.e.*, the building blocks of DHN, has a considerable potential to further improve the cross-domain recognition performance. This is indicated by the superior performance of DHN over that of SAE. Note that SAE trains every hidden layer using the sparse autoencoder training. There are two possible explanations about this outcome: i) using RBMs on the top of the autoencoder indeed helps to generalize deep networks, consistent with our prior hypothesis, and ii) encouraging sparse features in every hidden layer (as in SAE) might be too strong so that deep networks loose useful information rather than provide a better generalization.

## Sparse Local Connections

Despite the effectiveness of sparse features as described above, there are two cases where sparse features are not sufficient for reducing dataset bias: i) recognizing handwritten digits with squared block occlusion and ii) handling different shapes or styles of digits (simulated by the MNIST vs USPS recognition). We find that sDBN and sDHN, which employs sparse local connections in the first hidden layer, have a superior performance in those cases. sDBN is the best performing model on recognizing digits with squared block occlusion, while sDHN comes second. This indicates that the sparse connections only are sufficient for producing good models in this particular case. sDHN, however, is the best model in performing the MNIST vs USPS cross-domain recognition, while DHN and sDBN come second and third, respectively. The combination of sparse features and sparse connections are thus helpful for this task.

### 8.2.2 Regularized Feedforward Neural Network Training using MMD

Chapter 4 proposes Domain Adaptive Neural Network (DaNN), a feedforward neural net that reduces the distribution mismatch between the source data and target data hidden layer activations. DaNN employs Maximum Mean Discrepancy (MMD) as a regularization embedded in the back-propagation training. While MMD has been widely used in domain adaptation, DaNN is, to the best of our knowledge, the first neural network-based approach that uses MMD to deal with dataset bias. It is found that the use of MMD can effectively improves the domain adaptation performance of feedforward neural nets over six cross-domain object recognition tasks on the Office dataset.

We highlight two main findings from our performance evaluation below.

#### Performance on SURF-BoW Features

Chapter 4 finds that DaNN performs better than a feedforward neural net (NN) trained using the standard backpropagation on SURF-BoW features of the Office images. This indicates the effectiveness of our MMD regularization to provide domain-invariant representations for DaNN. Overall, DaNN has the second best averaged accuracy over six cross-domain tasks, while the prior state-of-the-art Transfer Sparse Coding (TSC) provides the best averaged accuracy. The superior performance of TSC might be due to the sparse feature extraction, which requires an extra iterative process at test time. The computational cost of DaNN at test time is identical to NN, which is an advantage over TSC in a real time scenario.

#### Performance on Raw Pixels and Effect of Autoencoder Pretraining

Using raw pixels directly as inputs, DaNN provides better averaged performance than NN and TSC in both *unsupervised* and *semi-supervised domain* adaptation settings. As in the SURF-BoW case, we can conclude that the MMD regularization is effective in providing good domain adaptation performance on raw pixels. Furthermore, we investigate the use of the denoising autoencoder (DAE) as the pretraining model in both NN and DaNN. It is found that DAE significantly improves the performance of NN and DaNN, where DaNN with the DAE pretraining performs best. This strongly suggests that the DAE pretraining provides good initial weight values such that it is easier for DaNN to find optimal weights in the context of domain adaptation.

### 8.2.3 Multi-task Representation Learning for Domain Generalization

Chapter 5 presents Multi-task Autoencoders (MTAE), a novel representation learning algorithm that provides domain-invariant representations useful for a range of cross-domain object recognition tasks. Given an image from a particular source domain, MTAE learns to reconstruct its analogs in other source domains via a shared encoding representation. Chapter 5 also presents a variant of MTAE that incorporates a *denoising* strategy [225] that we refer to as D-MTAE.

In the following, we highlight the main findings in Chapter 5.

#### Performance Accuracy on Transformed Raw Pixels

The proposed algorithms were evaluated on the modified MNIST and ETH-80 datasets and compared with existing single-task autoencoder-based models, *i.e.*, autoendoer (AE), denoising autoencoder (DAE) [225], contractive autoencoder (CAE) [178], and the prior state-of-the-art uDICA [152]. The MNIST and ETH-80 images were transformed such that several domains from a particular dataset were constructed. In short, such a construction results in four new datasets associated with a particular transformation: MNIST-r (roll rotation), MNIST-s (scaling), ETH80-p (pitch rotation), and ETH80-y (yaw rotation) – each contains multiple domains corresponding to object viewpoints, see Section 5.3.

It is found that D-MTAE and MTAE provide the best and the second best average accuracy according to linear SVMs over a range of *leave-one-domain-out* recognition tasks. This indicates the effectiveness of representations extracted by MTAE and D-MTAE over those extracted by existing autoencoder-based models and uDICA in the domain generalization context.

### Weight Visualization

It is found that the weights of the learned MTAE and D-MTAE have an interesting visualization characteristic. That is, they form "filters" that capture the underlying *transformation* among source domains. Such filters, which may help to *undo* the unseen transformation appeared in the target domain, cannot be seen in the learned AE, DAE, and CAE; the learned weights of AE, DAE, and CAE describe more on the *contents* of the objects, *e.g.*, edge, local blob, and stroke detectors. This is a possible reason that MTAE and D-MTAE have a superior domain generalization performance.

We also find a difference between the weights of MTAE and those of D-MTAE. The D-MTAE's filters clearly describe both the domain transformation and the object contents. The object contents appear to be less prominent in the MTAE's weights. This effect implies a useful insight: *good representations can be induced by filters that capture both the underlying domain transformation and the object contents.*

### Analysis of Invariance

It is found that the singular value spectrum of the D-MTAE's Jacobian matrix of the hidden layer representations with respect to the inputs decays most rapidly, followed by MTAE and then DAE (with similar rates). The ranking of decay rates of the comparing algorithms matches their ranking in terms of performance accuracy. This indicates that D-MTAE does a better job than other autoencoder based models at representing the data variations near a lower-dimensional manifold where samples concentrate, which implies a higher level of representation invariance.

#### Performance Accuracy on Deep Convolutional Activation Features

Finally, Chapter 5 shows that the proposed algorithms provide competitive domain generalization performance on modern image recognition benchmarks: the Office and VLCS (VOC2007, Caltech, LabelMe, and SUN09) datasets, see Section 5.4. In this case, the algorithms act as the pretraining models for a single layer feedforward neural net. The inputs to the algorithms are in the form of Deep Convolutional Activation Features (DeCAF$_6$), which are extracted from the 6-th layer of AlexNet [56, 118].

Compared to the prior state-of-the-art methods such as Undo-Bias [113], UML [64], and LRE-SVM [236], MTAE and D-MTAE provide better averaged performance on the Office dataset. On the VLCS dataset, D-MTAE performs best, followed by LRE-SVM and then MTAE. Note that the above prior state-of-the-art methods require complicated optimization procedures, while the MTAE and D-MTAE learning can be obtained by standard back-propagation.

### 8.2.4 Fast Kernel-Based Representation Learning for Domain Adaptation and Domain Generalization

Chapter 6 presents Scatter Component Analysis (SCA), a fast representation learning algorithm that can be applied to both domain adaptation and domain generalization. SCA is based on a measure referred to as *scatter*, a geometrical tool to measure datapoint variance in the reproducing kernel Hilbert space (RKHS). SCA seeks a representation that satisfies four requirements: i) separate datapoints with different labels and ii) separate the data as a whole, while iii) not separating points sharing the same label and iv) not separating the two or more domains.

We summarize the main findings in Chapter 6 as follows.

#### Algorithm Complexity and Runtime Performance

Recall that the optimization of SCA reduces to a generalized eigenvalue problem that yields a fast and exact solution. The SCA's algorithm complexity is thus similar to that of kernel PCA [192], *i.e.*, $O(kn^2)$, where $n$ is the number of samples and $k$ is the number of leading eigenvectors. This is much lower than the complexity of

Transfer Joint Matching (TJM) [135], that is, $O(Tkn^2)$ – TJM uses an alternating eigendecomposition in which $T$ iterations are needed (typically, $T = 50$).

From a set of domain adaptation experiments on the MNIST, USPS, MSRC, VOC, and Office datasets, it is found that SCA performs much faster than TJM and Transfer Sparse Coding (TSC) [133] during training. SCA is 3 to 6$\times$ faster than TJM, and $> 50\times$ faster than TSC. SCA also runs significantly faster than prior state-of-the-art domain generalization algorithms. This outcomes indicate that SCA is better suited than other competitive algorithms if a real time training stage is required.

### Object Recognition Performance

In general, we find that SCA provides either competitive or better recognition performance accuracy compared to prior state-of-the-art algorithms for both domain adaptation and domain generalization. The evaluation results for all competing algorithms were according to the optimal hyper-parameters obtained from $k$-fold cross validation on the source domain. In SCA, only two out of four hyper-parameters were tuned, see Section 6.3.7. Such a tuning strategy is sufficient to produce the state-of-the-art results.

### Theoretical Bound of SCA

It is found that the *domain scatter* with two input domains is equal to the squared MMD operating on the unit ball in RKHS (Theorem 13). We also show that the domain scatter provides an upper bound for the *discrepancy distance* [140] if equipped with a universal kernel (Theorem 15). These outcomes result in a consequence that the domain scatter can control the generalization of any function in RKHS in the setting of domain adaptation (Theorem 17). We remark that the domain scatter with multiple input domains is one of the key terms arising in a bound in the setting of domain generalization (Remark 1).

### 8.2.5 Deep Multi-task Convolutional Networks for Domain Adaptation

Chapter 7 presents the Deep Reconstruction-Classification Network (DRCN), a deep convolutional network that jointly learns (source) label classification and (target) data reconstruction tasks via a shared encoding representation. In terms of the architecture, DRCN has two convolutional encoding-decoding pipelines with a shared encoder: i) the label prediction pipeline and ii) the data reconstruction pipeline, which correspond to the fore-mentioned tasks. In other words, DRCN combines a convolutional neural net [125] and a convolutional autoencoder [141] into one model.

We highlight the main findings of Chapter 7 below.

#### Object Recognition Accuracy

From a series of large scale cross-domain object recognition evaluations, it is found that DRCN outperforms the prior state-of-the-art Domain-Adversarial Neural Network (DANN) [73] in almost all cases. DRCN also performs better than a convolutional architecture pretrained by the greedy layer wise autoencoder learning and finetuned by supervised backpropagation. This indicates that the simultaneous *source classification-target reconstruction* strategy equipped in DRCN is more effective than the standard pretraining-finetuning strategy [97] for domain adaptation.

#### Contribution of Unlabeled Source Data

Recall that DRCN only utilizes unlabeled data from the *target* domain for the data reconstruction training. We find that it is not helpful to also include unlabeled source data as additional reconstruction training examples in DRCN. This is indicated by the empirical classification evaluation, where DRCN with additional unlabeled source data always underperforms the original DRCN.

#### Domain Correspondence

Learning *domain correspondence* is perhaps the core problem in domain adaptation. By observing the outputs of the data reconstruction pipeline qualitatively, it is found that DRCN successfully approximates the *correspondence* between the source and target domains. More specifically, given an image from the source domain, its reconstructed output produced by the data reconstruction pipeline resembles the appearance of images from the target domain.

#### Theoretical Soundness

Under certain assumptions, we find that the learning objective of DRCN is approximately equivalent to solving a semi-supervised learning problem on the target domain. This can be shown by interpreting the objective as a maximum likelihood estimate (MLE), where the data reconstruction pipeline is modeled by a Gaussian and the label predictor is a multinomial, fit by logistic regression. It is found that such an MLE approximates that of a semi-supervised learning problem studied in [46] (Theorem 18). The resulting theorem can be used to explain the *uselessness* of unlabeled source data as additional reconstruction training examples of DRCN.

## 8.3 Future Work

This section highlights several possible research directions for future work.

### 8.3.1 Better Hyperparameter Tuning Strategy

Domain adaptation and domain generalization algorithms with some tunable hyper-parameters suffer from a problem of choosing optimal hyper-parameters performed on *target* domains. The easiest way to deal with this problem is using a standard tuning strategy: grid-searching the hyper-parameters according to the best validation performance on *source* domains. All algorithms proposed in this thesis perform such a tuning strategy. However, the standard tuning strategy is not theoretically well motivated in the perspective of domain adaptation/generalization and may worsen the dataset bias, since it is only approaching to the source distribution.

Only few tuning strategies related to domain adaptation and transfer learning exist in literature to date [34, 242]. To the best of our knowledge, none of them have been used in computer vision tasks. More work towards finding best practical hyper-parameter tuning strategies for domain adaptation or domain generalization is needed.

### 8.3.2 Further Analyses on Domain Correspondence

Results from Chapter 7 indicate that learning domain correspondence without target labels may be the key to a successful domain adaptation. The proposed algorithm that involves the joint reconstruction-classification learning is one of the approaches to approximating the domain correspondence. However, we still do not fully understand the actual root that induces the *true* domain correspondence. Both empirical and theoretical investigations associated with the domain correspondence are needed. This may lead to a better proposal of, *e.g.*, distance measure or a radically new algorithm, for solving dataset bias or domain shift that explicitly models the domain correspondence. We hypothesize that a better approach to learning the domain correspondence implies better domain adaptation or domain generalization models.

### 8.3.3 Beyond Object Recognition Tasks

Current domain adaptation or domain generalization research in the area of computer vision, including this thesis, focuses on object recognition / classification and detection. Other computer vision tasks such as segmentation, motion analysis / tracking, and scene reconstruction, in which dataset bias or domain shift may exist, are still waiting for domain adaptation or domain generalization solutions.

In object tracking, for example, a popular approach is Tracking-by-detection [32] that relies on the performance of an object detector. One may encounter a situation where a large labeled dataset is not available to train a good performing object detector on a target domain. This is an example that domain adaptation or domain generalization may be applied to adapt an object detector trained on different but related source domains.

### 8.3.4 Beyond Vectorial Representations

In machine learning, an input sample is commonly represented as a vector in Euclidean space. When applying machine learning on computer vision problems, we usually encourage the vectorization of the image representation although it may not be a natural representation. Thus, a possible research direction is that of reducing dataset bias or domain shift for structured / non-vectorial data representations. One of our contributions, *i.e.*, Chapter 7, has attempted to retain the 2D image representations as inputs by virtue of convolutional neural nets. However, further investigations on adapting other non-vectorial representations such as shapes and contours, deformable and articulated 2D or 3D objects, graphs and random fields, and intrinsic images, are desirable.

### 8.3.5 Learning from Visual Motion

Most current visual recognition algorithms, if not all, are unaware of visual motion. They follow the standard statistical learning paradigm, where images or videos are treated as "passive" instances drawn from an i.i.d. distribution. Some research suggested that visual motion is essential in the development of biological visual systems [94, 147, 164].

Held and Hein investigated the impact of visual motion by conducting an experiment on neonatal kittens [94]. A pair of kittens, "active" and "passive" kittens, were kept in a dark environment for eight weeks but one hour a day in a carousel. During the time in the carousel, both followed the same trajectory, but the active kitten could move his body freely on its own desire while the passive kittens could not. This means that the active kitten received more exposure of object motion than the passive kitten. Held and Hein showed that the visual system of the passive kitten is far underdeveloped compared to that of the active kitten.

In the perspective of computational, however, the correct algorithms in leveraging useful knowledge from visual motion remain unknown. We hypothesize that such algorithms would further help reduce dataset bias or domain mismatch.

# Bibliography

[1] Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. A Learning Algorithm for Boltzmann machines. *Cognitive Science 9*, 1 (1985), 147–169.

[2] Aizerman, M., Braverman, E., and Rozonoer, L. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control 25* (1964), 821–837.

[3] Alpaydin, E. *Introduction to Machine Learning*, 2 ed. MIT Press, Cambridge, Massachusetts, 2010.

[4] Amari, S. A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers 16*, 3 (1967), 299–307.

[5] Angluin, D. Queries and concept learning. *Machine Learning 2*, 4 (1988), 319–342.

[6] Argyriou, A., Evgeniou, T., and Pontil, M. Multi-task Feature Learning. In *Advances in Neural Information Processing Systems 19* (2006), pp. 41–48.

[7] Argyriou, A., Evgeniou, T., and Pontil, M. Convex Multi-Task Feature Learning. *Machine Learning 73*, 3 (2008), 243–272.

[8] Aronszajn, N. Theory of reproducing kernels. *Transactions of the American Mathematical Society 68*, 3 (1950), 337–404.

[9] Bach, F. R., and Jordan, M. I. Kernel independent component analysis. *Journal of Machine Learning Research 3* (2002), 1–48.

[10] Baktashmotlagh, M., Harandi, M. T., Lovell, B. C., and Salzmann, M. Unsupervised Domain Adaptation by Domain Invariant Projection. In *ICCV* (2013), pp. 769–776.

[11] Baktashmotlagh, M., Harandi, M. T., Lovell, B. C., and Salzmann, M. Domain Adaptation on the Statistical Manifold. In *IEEE Conference on Computer Vision and Pattern Recognition* (2014).

[12] Bartlett, P. L., and Mendelson, S. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research 3* (2002), 463–482.

[13] Bastien, F., Lamblin, P., Pascanu, R., Bergstra, J., Goodfellow, I. J., Bergeron, A., Bouchard, N., and Bengio, Y. Theano: new features and speed improvements. In *Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop* (2012).

[14] Baxter, J. A Model of Inductive Bias Learning. *Journal of Artificial Intelligence Research 12* (2000), 149–198.

[15] Bay, H., Tuytelaars, T., and Gool, L. V. SURF: Speeded Up Robust Features. *Computer Vision and Image Understanding (CVIU) 110*, 3 (2008), 346–359.

[16] Ben-David, S., Blitzer, J., Crammer, K., and Pereira, F. Analysis of Representations for Domain Adaptation. In *Advances in Neural Information Processing Systems (NIPS)* (2007), pp. 137–144.

[17] Bengio, Y. Learning deep architectures for AI. *Foundations and Trends in Machine Learning 2*, 1 (2009), 1–127.

[18] Bengio, Y. Deep Learning of Representations: Looking Forward. In *Statistical Language and Speech Processing*, vol. 7978 of *Lecture Notes in Computer Science*. Springer, 2013, pp. 1–37.

[19] Bengio, Y., Courville, A. C., and Vincent, P. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence 35*, 8 (2013), 1798–1828.

[20] Bengio, Y., Goodfellow, I. J., and Courville, A. Deep Learning. Book in preparation for MIT Press, 2015.

[21] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. Greedy Layer-Wise Training of Deep Networks. In *Advances in Neural Information Processings Systems 19* (2007), vol. 19, pp. 153–160.

[22] Bengio, Y., and LeCun, Y. Scaling Learning Algorithms towards AI. In *Large-Scale Kernel Machines*, L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, Eds. MIT Press, 2007.

[23] Bengio, Y., Yao, L., Guillaume, A., and Vincent, P. Generalized denoising auto-encoders as generative models. In *NIPS* (2013), pp. 899–907.

[24] Bishop, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[25] Blanchard, G., Lee, G., and Scott, C. Generalizing from Several Related Classification Tasks to a New Unlabeled Sample. In *Advances in Neural Information Processing Systems (NIPS)* (2011), pp. 2178–2186.

[26] Blitzer, J., McDonald, R., and Pereira, F. Domain Adaptation with Structural Correspondence Learning. In *EMNLP* (2006), pp. 120–128.

[27] Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. K. Occam's razor. *Information Processing Letters 24* (1987), 377–380.

[28] Borgwardt, K. M., Gretton, A., Rasch, M. J., Kriegel, H.-P., Schölkopf, B., and Smola, A. J. Integrating structured biological data by Kernel Maximum Mean Discrepancy. *Bioinformatics 22*, 14 (2006), e49–e57.

[29] Boser, B., Guyon, I., and Vapnik, V. N. A training algorithm for optimal margin classifiers. In *5th Annual Workshop on Computational Learning Theory* (1992), pp. 144–152.

[30] Bourlard, H., and Kamp, Y. Auto-Association by Multilayer Perceptrons and Singular Value Decomposition. *Biological Cybernetics 59* (1988), 291–294.

[31] Bradley, D. M., and Bagnell, J. A. Differentiable Sparse Coding. In *Advances in Neural Information Processing Systems (NIPS)* (2009), vol. 21, pp. 113–120.

[32] Breitenstein, M. D., Reichlin, F., Leibe, B., Koller-Meier, E., and Gool, L. V. Robust tracking-by-detection using a detector confidence particle filter. In *Proceedings of International Conference on Computer Vision (ICCV)* (2009).

[33] Bremaud, P. *Markov Chains: Gibbs fields, Monte Carlo Simulations, and Queues.* Texts in Applied Mathematics. Springer, New York, Berlin, Heidelberg, 1999.

[34] Bruzzone, L., and Marconcini, M. Domain Adaptation Problems: A DASVM Classification Technique and a Circular Validation Strategy. *IIEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) 32*, 5 (2010), 770–787.

[35] Caruana, R. Multitask Learning. *Machine Learning 28* (1997), 41–75.

[36] Caseiro, R., Henriques, J. F., Martins, P., and Batista, J. Beyond the shortest path: Unsupervised domain adaptation by sampling ssubspace along the spline flow. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 3846–3854.

[37] Cauchy, M. A. Méthode générale pour la résolution des systèmes d'équations simultanées. *Comptes Rendus Hebd. Sèances Acad. Sci. 25* (1847), 536–538.

[38] Chen, M., Xu, Z., Weinberger, K., and Sha, F. Marginalized Denoising Autoencoders for Domain Adaptation. In *Proceedings of International Conference on Machine Learning (ICML)* (2012), pp. 767–774.

[39] Cho, K., Ilin, A., and Raiko, T. Improved Learning of Gaussian-Bernoulli Restricted Boltzmann Machines. In *ICANN 2011, Part 1, LCNS 6791*, T. H. et al. (Eds), Ed. Springer-Verlag Berlin Heidelberg, 2011, pp. 10–17.

[40] Choi, M. J., Lim, J. J., Torralba, A., and Willsky, A. S. Exploiting hierarchical context on a large database of object categories. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2010), pp. 129–136.

[41] Chopra, S., Balakrishnan, S., and Gopalan, R. DLID: Deep Learning for Domain Adaptation by Interpolating between Domains. In *ICML Workshop on Challenges in Representation Learning* (2013).

[42] Ciresan, D., Meier, U., and Schmidhuber, J. Multi-column deep neural network for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2012), pp. 3642–3649.

[43] Ciresan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J.-r. Flexible, High Performance Convolutional Neural Networks for Image Classification. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)* (2011), pp. 1237–1242.

[44] Coates, A., Lee, H., and Ng, A. Y. An Analysis of Single-Layer Networks in Unsupervised Feature Learning. In *Proceedings of International Conference on Artificial Intelligence and Statistics (AISTATS)* (2011), pp. 215–223.

[45] Coates, A., and Ng, A. Y. The Importance of Encoding Versus Training with Sparse Coding and Vector Quantization. In *Proceedings of the 28th International Conference on Machine Learning (ICML)* (2011), pp. 912–928.

[46] Cohen, I., and Cozman, F. G. Risks of semi-supervised learning: how unlabeled data can degrade performance of generative classifiers. In *Semi-Supervised Learning*. MIT Press, 2006.

[47] Cortes, C., and Mohri, M. Domain adaptation and sample bias correction theory and algorithm for regression. *Theoretical Computer Science 519* (2014), 103–126.

[48] Cortes, C., and Vapnik, V. N. Support-Vector Networks. *Machine Learning 20*, 3 (1995), 273–297.

[49] Crammer, K., and Singer, Y. On the algorithmic implementation of multiclass kernel-based vector machines. *JMLR 2* (2001), 265–292.

[50] Dahl, G. E., Sainath, T. N., and Hinton, G. Improving deep neural networks for LVCSR using rectified linear units and dropout. In *ICASSP* (2013).

[51] Dalal, N., and Triggs, B. Histograms of Oriented Gradients for Human Detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2005), pp. 886–893.

[52] Deng, L., Hinton, G. E., and Kingsbury, B. New types of deep neural network learning for speech recognition and related applications: An overview. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)* (2013), pp. 8599–8603.

[53] Deng, L., and Yu, D. Deep learning: Methods and applications. *Foundations and Trends in Signal Processing 7*, 3-4 (2014), 197–387.

[54] Do, C. B., and Ng, A. Y. Transfer learning for text classification. In *Advances in Neural Information Processing Systems (NIPS)* (2005), vol. 18, pp. 299–306.

[55] Domingos, P. A few useful thing to know about machine learning. *Communications of the ACM 55*, 10 (2012), 78–87.

[56] Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., and Darrell, T. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. In *International Conference on Machine Learning (ICML)* (2014), pp. 647–655.

[57] Donoho, D. L. For most large underdetermined systems of linear equations the minimal l1-norm solution is also the sparsest solution. In *Communications on pure and applied mathematics*, vol. 56 of *797–829*. Wiley Online Library, 2006.

[58] Dreyfus, S. E. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications 5*, 1 (1962), 30–45.

[59] Dreyfus, S. E. The computational solution of optimal control problems with time lag. *IEEE Transactions on Automatic Control 18*, 4 (1973), 383–385.

[60] Duan, L., Tsang, I. W., and Xu, D. Domain Adaptation from Multiple Sources via Auxiliary Classifiers. In *Proceedings of the International Conference on Machine Learning (ICML)* (2009), pp. 289–296.

[61] Elad, M., and Aharon, M. Image denoising via sparse and redundant representations over learned dictionaries. In *IEEE Transactions on Image Processing* (2006), vol. 15, pp. 3736–3745.

[62] Everingham, M., Van-Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. The PASCAL Visual Object Classes Challenge 2007 Results, 2007.

[63] Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research 9* (2008), 1871–1874.

[64] Fang, C., Xu, Y., and Rockmore, D. N. Unbiased Metric Learning: On the Utilization of Multiple Datasets and Web Images for Softening Bias. In *ICCV* (2013), pp. 1657–1664.

[65] Feldman, J. The structure of perceptual categories. *Journal of Mathematical Psychology 41* (1997), 145–170.

[66] Felzenszwalb, P., McAllester, D., and Ramanan, D. A Discriminatively Trained, Multiscale, Deformable Part Model. In *IEEE Conference on Computer Vision and Patter Recognition (CVPR)* (2008), pp. 1–8.

[67] Fernando, B., Habrard, A., Sebban, M., and Tuytelaars, T. Unsupervised Visual Domain Adaptation Using Subspace Alignment. In *ICCV* (2013), pp. 2960–2967.

[68] Fiesler, E., Caulfield, H. J., and Choudry, A. Some Theoretical Upperbounds on the Capacity of Neural Networks. In *Proceedings of the First Workshop on Neural Networks* (1990), vol. 2, pp. 51–58.

[69] Fisher, R. A. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics 7* (1936), 179–188.

[70] Fréchet, M. Sur les ensemblres de fonctions et les opérations linéaires. *Comptes rendus de l'Académie des sciences 144* (1907), 1414–1416.

[71] Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics 36* (1980), 193–202.

[72] Ganin, Y., and Lempitsky, V. S. Unsupervised domain adaptation by backpropagation. In *ICML* (2015), pp. 1180–1189.

[73] Ganin, Y., Ustinova, E., Ajakan, H., Germain, P., Larochelle, H., Laviolette, F., Marchand, M., and Lempitsky, V. S. Domain-adversarial training of neural networks. *CoRR arXiv:1505.07818* (2015).

[74] Geman, S., and Geman, D. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence 6*, 6 (1984), 721–741.

[75] Geng, B., Tao, D., and Xu, C. DAML: Domain Adaptation Metric Learning. *IEEE Transactions on Image Processing 20*, 10 (2011), 2980–2989.

[76] Gibbs, J. W. *Elementary Principles in Statistical Mechanics.* New York: Cnarles Scribner's Sons, 1902.

[77] Girshick, R., Donahue, J., Darrell, T., and Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2014), pp. 580–587.

[78] Glorot, X., Bordes, A., and Bengio, Y. Deep Sparse Rectifier Neural Network. In *AISTATS* (2011), pp. 315–323.

[79] Glorot, X., Bordes, A., and Bengio, Y. Domain Adaptation for Large-Scale Sentiment Classification: A Deep Learning Approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML)* (2011), pp. 513–520.

[80] Godfrey, J., and Holliman, E. Switchboard-1 release 2 ldc97s62. In *Philadelphia: Linguistic Data Consortium. Web Download* (1993).

[81] Gong, B., Grauman, K., and Sha, F. Connecting the Dots with Landmarks: Discriminatively Learning Domain-Invariant Features for Unsupervised Domain Adaptation. In *Proceedings of the International Conference on Machine Learning (ICML)* (2013), pp. 222–230.

[82] Gong, B., Grauman, K., and Sha, F. Reshaping Visual Datasets for Domain Adaptation. In *Advances in Neural Information Processing Systems 26 (NIPS)* (2013), pp. 1286–1294.

[83] Gong, B., Shi, Y., Sha, F., and Grauman, K. Geodesic Flow Kernel for Unsupervised Domain Adaptation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2012), pp. 2066–2073.

[84] Gopalan, R., Li, R., and Chellapa, R. Domain Adaptation for Object Recognition: An Unsupervised Approach. In *ICCV* (2011), pp. 999–1006.

[85] Goroshin, R., and LeCun, Y. Saturating auto-encoders. In *Proceedings of International Conference on Learning Representations (ICLR)* (2013).

[86] Gretton, A., Borgwardt, K. M., Rasch, M., Schölkopf, B., and Smola, A. J. A Kernel Method for the Two-Sample-Problem. In *Advances in Neural Information Processing System 19 (NIPS)* (2007), pp. 513–520.

[87] Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. A Kernel Two-Sample Test. *JMLR* (2012), 723–773.

[88] Griffin, G., Holub, A., and Perona, P. Caltech-256 object category dataset. Tech. rep., California Inst. of Tech., 2007.

[89] Griffin, G., Holub, A., and Perona, P. Caltech-256 object category dataset. Tech. rep., California Inst. of Tech., 2007.

[90] Grünewälder, S., Gretton, A., and Shawe-Taylor, J. Smooth Operators. In *Proceedings ot the International Conference on Machine Learning (ICML)* (2013), pp. 1184–1192.

[91] Hastings, W. K. Monte Carlo Sampling Methods Using Markov Chains and Their Applications. *Biometrika 57*, 1 (1970), 97–109.

[92] Haussler, D. Probably approximately correct learning. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)* (1990), pp. 1101–1108.

[93] He, K., Zhang, X., Ren, S., and Sun, J. Delving Deep into RRectifier: Surpassing Human-Level Performance on ImageNet Classification. *IEEE International Conference on Computer Vision (ICCV)* (2015).

[94] Held, R., and Hein, A. Movement-produced stimulation in the development of visualiy guided behavior. *Journal of Comparative and Physiological Psychology 56*, 5 (1963), 872–876.

[95] Hestenes, M., and Stiefel, E. Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards 49*, 6 (1952).

[96] Hinton, G. E. Training products of experts by minimizing contrastive divergence. *Neural Computation 14* (2002), 1771–1800.

[97] Hinton, G. E., and Osindero, S. A fast learning algorithm for deep belief nets. *Neural Computation 18*, 7 (2006), 1527–1554.

[98] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR abs/1207.0* (2012).

[99] Hoeffding, W. Probability inequalities for sums of bounded random variables. In *Journal of the American Statistical Association*, vol. 58. 1963, pp. 13–30.

[100] Hoffman, J., Darrell, T., and Saenko, K. Continuous Manifold Based Adaptation for Evolving Visual Domains. In *IEEE Conference on Computer Vision and Pattern Recognition* (2014), pp. 867–874.

[101] Hoffman, J., Rodner, E., Donahue, J., Darrell, T., and Saenko, K. Efficient Learning of Domain-invariant Image Representations. In *Proceedings of the International Conference on Learning Representation* (2013).

[102] Hoffman, J., Tzeng, E., Donahue, J., Jia, Y., Saenko, K., and Darrell, T. One-Shot Adaptation of Supervised Deep Convolutional Models. *CoRR abs/1312.6* (2013).

[103] Hotelling, H. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology 24* (1933), 417–441.

[104] Huang, J., Smola, A. J., Gretton, A., Borgwardt, K. M., and Schölkopf, B. Correcting sample selection bias by unlabeled data. In *Advances in Neural Information Processing Systems 19 (NIPS)* (2007), pp. 601–608.

[105] Hubel, D. H., and Wiesel, T. Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology 195* (1968), 215–243.

[106] Hull, J. J. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) 16*, 5 (1994), 550–554.

[107] Hüper, K., and Leite, F. S. On the geometry of rolling and interpolation curves on sn, son, grasmann manifolds. *Journal of Dynamical and Control Systems 13*, 4 (2007), 467–502.

[108] Hyvärinen, A., and Oja, E. Independent Component Analysis: Algorithms and Applications. *Neural Networks 13*, 4–5 (2000), 411–430.

[109] Jarrett, K., Kavukcuoglu, K., Ranzato, M., and LeCun, Y. What is the Best Multi-Stage Architecture for Object Recognition? In *Proceedings of International Conference on Computer Vision (ICCV)* (2009), pp. 2146–2153.

[110] Jiang, J. A Literature Survey on Domain Adaptation of Statistical Classifiers. Available at `http://sifaka.cs.uiuc.edu/jiang4/domain_adaptation/survey`, 2008.

[111] Jiang, W., Nie, F., lai Korris Chung, F., and Huang, H. Algorithm and Theoretical Analysis for Domain Adaptation Feature Learning with Linear Classifiers. *CoRR arXiv:1509.01710v1* (2015).

[112] Jordan, M. I. *Learning in Graphical Models.* The MIT Press, 1998.

[113] Khosla, A., Zhou, T., Malisiewicz, T., Efros, A., and Torralba, A. Undoing the Damage of Dataset Bias. In *European Conference on Computer Vision (ECCV)* (2012), vol. I, pp. 158–171.

[114] Kifer, D., Ben-David, S., and Gehrke, J. Detecting Change in Data Streams. In *Proceedings of VLDB Conference* (2004), pp. 180–191.

[115] Kim, M., and Pavlovic, V. Central Subspace Dimensionality Reduction Using Covariance Operators. *IEEE TPAMI 33*, 4 (2011), 657–670.

[116] Kreyszig, E. *Introductory Functional Analysis with Applications.* John Wiley & Sons, 1978.

[117] Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images. Master's thesis, Department of Computer Science, University of Toronto, Apr. 2009.

[118] Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)* (2012), vol. 25, pp. 1106–1114l.

[119] Kulis, B., Saenko, K., and Darrell, T. What You Saw is Not What You Get: Domain Adaptation Using Asymmetric Kernel Transforms. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2011), pp. 1785–1792.

[120] Kuss, M., and Graepel, T. The Geometry of Kernel Canonical Correlation Analysis. Tech. Rep. 108, Max Planck Institute for Biological Cybernetics, Tuebingen, 2003.

[121] Landau, L. D., and Mikhailovich, E. *Statistical Physics. Course of Theoretical Physics*, 3 ed., vol. 5. Oxford: Pergamon Press, 1980.

[122] Le, Q. V., Ngiam, J., Lahiri, A. C. A., Prochnow, B., and Ng, A. Y. On Optimization Methods for Deep Learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML)* (2011), pp. 265–272.

[123] LeCun, Y., Bengio, Y., and Hinton, G. E. Deep learning. *Nature 521* (2015), 436–444.

[124] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural Computation 1*, 4 (1989), 541–551.

[125] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE* (1998), vol. 86, pp. 2278–2324.

[126] LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. Efficient backprop. In *Neural Networks: Tricks of the Trade*. Springer Berlin Heidelberg, 1998, ch. 2, p. 546.

[127] Ledoux, M., and Talagrand, M. *Probability in Banach Space*. Springer-Verlag, New York, 1991.

[128] Lee, H., Ekanadham, C., and Ng, A. Y. Sparse Deep Belief Net Model for Visual Area V2. In *Advances in Neural Information Processing Systems (NIPS)* (2007), vol. 20, pp. 873–880.

[129] Leibe, B., and Schiele, B. Analyzing appearnce and contour based methods for object categorizatio. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2003), pp. 409–415.

[130] Li, K.-C. Sliced inverse regression for dimension reduction. *J. American Statistical Association 86*, 414 (1991), 316–327.

[131] Li, M., and Vitányi, P. *An Introduction to Kolmogorov Complexity and Its Applications (2nd Ed.)*. Springer-Verlag New York, Inc., 1997.

[132] Long, M., Cao, Y., Wang, J., and Jordan, M. I. Learning transferable features with deep adaptation networks. In *Proceedings of the International Conference on Machine Learning (ICML)* (2015).

[133] Long, M., Ding, G., Wang, J., Sun, J., Guo, Y., and Yu, P. S. Transfer Sparse Coding for Robust Image Representation. In *CVPR* (2013), pp. 404–414.

[134] Long, M., Wang, J., Ding, G., Shen, D., and Yang, Q. Transfer Learning with Graph Co-Regularization. *IEEE Transactions on Knowledge and Data Engineering 26*, 7 (2014), 1805–1818.

[135] Long, M., Wang, J., Ding, G., Sun, J., and Yu, P. S. Transfer Joint Matching for Unsupervised Domain Adaptation. In *CVPR* (2014), pp. 1410–1417.

[136] Lowe, D. G. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision (ICCV)* (1999), pp. 1150–1157.

[137] Mairal, J., Bach, F., Ponce, J., Sapiro, G., and Zisserman, A. Supervised dictionary learning. In *Advances in Neural Information Processing Systems (NIPS)* (2008), pp. 1033–1040.

[138] Mairal, J., Elad, M., and Sapiro, G. Sparse representation for color image restoration. In *IEEE Transactions on Image Processing* (2008), vol. 17, pp. 53–69.

[139] Malisiewicz, T., Gupta, A., and Efros, A. A. Ensemble of Exemplar-SVMs for Object Detection and Beyond. In *IEEE International Conference on Computer Vision (ICCV)* (2011), pp. 89–96.

[140] Mansour, Y., Mohri, M., and Rostamizadeh, A. Domain adaptation: Learning bounds and algorithms. In *Proceedings of the 22nd Annual Conference on Learning Theory (COLT)* (2009).

[141] Masci, J., Meier, U., Ciresan, D., and Schmidhuber, J.-e. Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction. In *Proceedings of the 21st International Conference on Artificial Neural Networks* (2011), pp. 52–59.

[142] McCulloch, W. S., and Pitts, W. A Logical Calculus of The Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics 5* (1943).

[143] McDiarmid, C. On the Method of Bounded Differences. *Surveys in Combinatorics 141*, 148-188 (1989).

[144] Mercer, J. Functions of positive and negative type and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society A 209*, 441–458 (1909), 415–446.

[145] Micchelli, C. A., Xu, Y., and Zhang, H. Universal Kernels. *Journal of Machine Learning Research (JMLR) 7* (2006), 2651–2667.

[146] Mika, S., Rätsch, G., Weston, J., Schölkopf, B., and Müller, K.-R. Fisher discriminant analysis with kernels. In *Neural Networks for Signal Processing IX* (1999), pp. 41–48.

[147] Mikami, A., Newsome, W. T., and Wurtz, R. H. Motion selectivity in macaque visual ccortex. II. Spatiotemporal range of directional interactions in MT and V1. *Journal of Neurophysiology 55*, 6 (1986), 1328–1339.

[148] Minsky, M., and Papert, S. *Perceptrons: An Introduction to Computational Geometry.* The MIT Press, 1969.

[149] Mitchell, T. M. *Machine Learning.* McGraw-Hill, Inc, New York, NY, USA, 1997.

[150] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature* (2015), 529–533.

[151] Mohri, M., Rostamizadeh, A., and Talwalkar, A. *Foundations of Machine Learning.* The MIT Press, 2012.

[152] Muandet, K., Balduzzi, D., and Schölkopf, B. Domain Generalization via Invariant Feature Representation. In *Proceedings of the International Conference on Machine Learning (ICML)* (2013), pp. 10–18.

[153] Nair, V., and Hinton, G. E. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the International Conference on Machine Learning (ICML)* (2010).

[154] Natarajan, B. K. On learning sets and functions. *Machine Learning 4* (1989), 67–97.

[155] Natarajan, B. K. Sparse approximate solutions to linear systems. In *SIAM J. Comput.* (1995), vol. 24, pp. 227–234.

[156] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., and Ng, A. Y. Reading Digits in Natural Images with Unsupervised Feature Learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning* (2011).

[157] Ng, A. Sparse autoencoder. CS294A Lecture notes, 2011.

[158] Nguyen, H. V., Ho, H. T., Patel, V. M., and Chellapa, R. DASH-N: Joint Hierarchical Domain Adaptation and Feature Learning. *IEEE Transactions on Image Processing 24*, 12 (2015), 5479–5491.

[159] Niu, L., Li, W., and Xu, D. Visual Recognition by Learning from Web Data: A Weakly Supervised Domain Generalization Approach. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 2774–2783.

[160] Nocedal, J., and Wright, S. J. *Numerical Optimization.* Springer, 1999.

[161] Ojala, T., Pietikäinen, M., and Harwood, D. A comparative study of texture measures with classification based on feature distributions. *Pattern Recognition 29*, 1 (1996), 51–59.

[162] Olshausen, B., and Field, D. Emergence of simple-cell receptive field properties by learning sparse code for natural images. In *Nature*. 1996, pp. 607–609.

[163] Olshausen, B. A., and Field, D. J. Sparse coding with an overcomplete basis set: a strategy employed by V1. *Vision Research 37* (1997), 3311–3325.

[164] O'Regan, J. K., and Noë, A. A sensorimotor account of vision and visual consciousness. *Behavioral and Brain Sciences 24* (2001), 939–1031.

[165] Pan, S. J., Tsang, I. W., Kwok, J. T., and Yang, Q. Domain adaptation via transfer component analysis. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (2009), pp. 1187–1192.

[166] Pan, S. J., Tsang, I. W.-H., Kwok, J. T., and Yang, Q. Domain adaptation via transfer component analysis. *IEEE Trans. Neural Networks 22*, 2 (2011), 199–210.

[167] Pan, S. J., and Yang, Q. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering 22*, 10 (2010), 1345–1359.

[168] Patel, V. M., Gopalan, R., Li, R., and Chellapa, R. Visual domain adaptation: A survey of recent advances. *IEEE Signal Processing Magazine 32*, 3 (2015), 53–69.

[169] Pazzani, M. J., and Sarrett, W. A framework for average case analysis of conjuctive learning algorithms. In *Machine Learning* (1992), vol. 9, pp. 349–372.

[170] Pearson, K. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine 2*, 6 (1901), 559–572.

[171] Ponce, J., Berg, T. L., Everingham, M., Forsyth, D. A., Hebert, M., Lazebnik, S., Marsza?ek, M., Schmid, C., Russell, B. C., Torralba, A., Williams, C. K. I., Zhang, J., and Zisserman, A. Dataset issues in object recognition. In *Toward Category-Level Object Recognition*, vol. 4170. 2006, pp. 29–48.

[172] Pratt, L. Y. Discriminability-Based Transfer between Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)* (1993), pp. 204–211.

[173] Rahimi, A., and Recht, B. Random Features for Large-Scale Kernel Machines. In *Advances in Neural Information Processing Systems (NIPS)* (2007), pp. 1177–1184.

[174] Ranzato, M., Boureau, Y.-L., and LeCun, Y. Sparse Feature Learning for Deep Belief Networks. In *Advances in Neural Information Processing Systems (NIPS)* (2008), vol. 20, pp. 1185–1192.

[175] Ranzato, M., Huang, F.-J., Boureau, Y.-L., and LeCun, Y. Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2007).

[176] Reed, M., and Simon, B. *Functional Analysis, Methods of Modern Mathematical Physics.* Academic Press, 1980.

[177] Riesz, F. Sur les opérations fonctionnelles linéaires. *Comptes rendus de l'Académie des sciences 149* (1909), 974–977.

[178] Rifai, S., Vincent, P., Muller, X., Glorot, X., and Bengio, Y. Contractive Auto-Encoders : Explicit Invariance During Feature Extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML)* (2011), pp. 833–840.

[179] Rosasco, L. Reproducing kernel hilbert spaces, February 2007.

[180] Rosenblatt, F. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review 65* (1958), 386–408.

[181] Roweis, S. T., and Saul, L. K. Nonlinear dimensionality reduction by locally linear embedding. *SCIENCE 290* (2000), 2323–2326.

[182] Rumelhart, D., Hinton, G., and Williams, R. *Parallel Distributed Processing. I: Foundations.* MIT Press, 1986.

[183] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning internal representations by error propagation. In *Parallel Distributed Pro-*

*cessing: Explorations in the Microstructure of Cognition, Vol. 1.* MIT Press, Cambridge, MA, USA, 1986, pp. 318–362.

[184] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Nature 323* (1986), 533–536.

[185] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV) 115* (2015), 211–252.

[186] Russell, B. C., Torralba, A., Murphy, K. P., and Freeman, W. T. LabelMe: A database and web-based tool for image annotation. In *International Journal of Computer Vision (IJCV)*, vol. 77. 2008, pp. 157–173.

[187] Saenko, K., Kulis, B., Fritz, M., and Darrell, T. Adapting Visual Category Models to New Domains. In *European Conference on Computer Vision (ECCV)* (2010), pp. 213–226.

[188] Salakhutdinov, R., and Hinton, G. E. Deep {B}oltzmann Machines. In *the 12th International Conference on Artificial Intelligence and Statistics (AISTATS)* (2009), pp. 448–455.

[189] Samuel, A. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development 3*, 3 (1959), 210–229.

[190] Scherer, D., Müller, A., and Behnke, S. Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition. In *International Conference on Artificial Neural Networks (ICANN)* (2010), pp. 92–101.

[191] Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks 61* (2015), 85–117.

[192] Schölkopf, B., Smola, A., and Müller, K.-R. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation 10*, 5 (1998), 1299–1319.

[193] Schölkopf, B., and Smola, A. J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, Cambridge, MA, USA, 2001.

[194] Sermanet, P., Chintala, S., and LeCun, Y. Convolutional neural networks applied to house number digit classification. In *Proceedings of the International Conference on Pattern Recognition (ICPR)* (2012), pp. 3288–3291.

[195] Shekhar, S., Patel, V. M., Nguyen, H. V., and Chellapa, R. Generalized domain-adaptive dictionaries. In *IEEE Conference on Computer Vision and Pattern Recognition* (2013), pp. 361–368.

[196] Shimodaira, H. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference 90*, 2 (2000), 227–244.

[197] Simard, P. Y., Steinkraus, D., and Platt, J. C. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR* (2003), vol. 2, pp. 958–962.

[198] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representation (ICLR)* (2015).

[199] Smola, A., Gretton, A., Song, L., and Schölkopf, B. A Hilbert Space Embedding for Distributions. In *Algorithmic Learning Theory (ALT)* (2007), pp. 13–31.

[200] Smolensky, P. Information processing in dynamical systems: Foundations of harmony theory. In *Parallel distributed processing: explorations in the microstructure of cognition*, vol. 1. MIT Press, 1986, pp. 194–281.

[201] Sriperumbudur, B. K., Gretton, A., Fukumizu, K., Schölkopf, B., and Lanckriet, G. R. G. Hilbert space embeddings and metrics on probability measures. *Journal of Machine Learning Research (JMLR) 11* (2010), 1517–1561.

[202] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research (JMLR) 15*, 1 (2014), 1929–1958.

[203] Steinwart, I. On the influence of the kernel on the consistency of support vector machines. *Journal of Machine Learning Research 2* (2002), 67–93.

[204] Stracuzzi, D. J. Memory Organization and Knowledge Transfer. In *ICML Workshop on Structural Knowledge Transfer for Machine Learning* (2006).

[205] Sun, B., Feng, J., and Saenko, K. Return of Frustratingly Easy Domain Adaptation. In *AAAI* (2016), pp. 2058–2065.

[206] Sun, B., and Saenko, K. From Virtual to Reality: Fast Adaptation of Virtual Object Detectors to Real Domains. In *BMVC* (2014).

[207] Sutskever, I., Vinyals, O., and Le, Q. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)* (2014), pp. 3104–3112.

[208] Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2014), pp. 1701–1708.

[209] Tang, Y., and Eliasmith, C. Deep networks for robust visual recognition. In *Proceedings of the 27th International Conference on Machine Learning* (2010), pp. 1055–1062.

[210] Tang, Y., Salakhutdinov, R., and Hinton, G. Deep Lambertian Networks. In *Proceedings of the International Conference on Machine Learning (ICML)* (2012).

[211] Thrun, S. Is learning the n-th thing any easier than learning the first? In *Advances in Neural Information Processing Systems (NIPS)* (1996), pp. 640–646.

[212] Thrun, S., and Pratt, L., Eds. *Learning to Learn.* Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[213] Tieleman, T., and Hinton, G. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[214] Tommasi, T., and Caputo, B. Frustratingly Easy NBNN Domain Adaptation. In *IEEE International Conference on Computer Vision (ICCV)* (2013), pp. 897–904.

[215] Torralba, A., and Efros, A. A. Unbiased Look at Dataset Bias. In *IEEE Conferenceon Computer Vision and Pattern Recognition (CVPR)* (2011), pp. 1521–1528.

[216] Tzeng, E., Hoffman, J., Darrell, T., and Saenko, K. Simultaneous deep transfer across domains and tasks. In *ICCV* (2015).

[217] Tzeng, E., Hoffman, J., Zhang, N., Saenko, K., and Darrell, T. Deep domain confusion: Maximizing for domain invariance. *CoRR abs/1412.3474* (2014).

[218] Valiant, L. G. A theory of the learnable. *Communications of the ACM 27*, 11 (1984), 1134–1142.

[219] van der Maaten, L., and Hinton, G. Visualizing High-Dimensional Data Using t-SNE. *Journal of Machine Learning Research (JMLR) 9* (2008), 2579–2605.

[220] van Gestel, T., Suykens, J. A. K., Baesens, B., Viaene, S., Vanthienen, J., Dedene, G., de Moor, B., and Vandewalle, J. Benchmarking least squares support vector machine classifiers. *Machine Learning 54*, 1 (2004), 5–32.

[221] Vapnik, V. *The Nature of Statistical Learning Theory.* Springer, New York, 1995.

[222] Vapnik, V. N. *Estimation of Dependences Based on Empirical Data.* Springer Series in Statistics. Springer-Verlag New York, 1982.

[223] Vedaldi, A., and Fulkerson, B. VLFeat: An Open and Portable Library of Computer Vision Algorithms. `http://www.vlfeat.org`, 2008.

[224] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. Extracting and Composing Robust Features with Denoising Autoencoders. In *Proceedings of the 25th International Conference on Machine Learning (ICML)* (2008), pp. 1096–1103.

[225] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. *Journal of Machine Learning Research 11* (2010), 3371–3408.

[226] Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. Show and Tell: A Neural Image Caption Generator. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 3156–3164.

[227] Wang, C., and Mahadevan, S. Heterogeneous Domain Adaptation using Manifold Alignment. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (2011), pp. 1541–1546.

[228] Wang, H., Kläser, A., Schmid, C., and Liu, C.-L. Dense trajectories and motion boundary descriptors for action recognition. *International Journal of Computer Vision (IJCV) 103*, 1 (2013), 60–79.

[229] Weinland, D., Ronfard, R., and Boyer, E. Free viewpview action recognition using motion history volumes. *Computer Vision and Image Understanding (CVIU) 104*, 2 (2006), 249–257.

[230] Welch, B. L. The generalization of "Student's" problem when several different population variances are involved. *Biometrika 34*, 1–2 (1947), 28–35.

[231] Werbos, P. J. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, 1974.

[232] Werbos, P. J. Applications of advances in nonlinear sensitivity analysis. In *Proceedings of the 10th IFIP Conference* (1981), pp. 762–770.

[233] White, H. Maximum likelihood estimation of misspecified models. *Econometrica 50*, 1 (1982), 1–25.

[234] Williams, C. K. I., and Seeger, M. Using the Nystr–m Method to Speed Up Kernel Machines. In *Advances in Neural Information Processing Systems (NIPS)* (2001), pp. 682–688.

[235] Winn, J., Criminisi, A., and Minka, T. Object categorization by learned universal visual dictionary. In *Proceedings of IEEE International Conference on Computer Vision (ICCV)* (2005), pp. 1800–1807.

[236] Xu, Z., Li, W., Niu, L., and Xu, D. Exploiting Low-Rank Structure from Latent Domains for Domain Generalization. In *European Conference on Computer Vision (ECCV)* (2014), pp. 628–643.

[237] Yang, J., Yan, R., and Hauptmann, A. G. Cross-domain video concept detection using adaptive SVMs. In *Proc of 15th Int Conf on Multimedia* (2007), pp. 188–197.

[238] Yang, J., Yu, K., Gong, Y., and Huang, T. Linear Spatial Pyramid Matching Using Sparse Coding for Image Classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2009), pp. 1794–1801.

[239] You, Q., Luo, J., Jin, H., and Yang, J. Robust image sentiment analysis using progressively trained and domain transferred deep networks. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence* (2015), pp. 381–388.

[240] Zaremba, S. L'équation biharmonique et une class remarquable de fonctions fondamentals harmoniques. *Bulletin International de l'Acaémie des Sciences de Cracovie* (1907), 147–196.

[241] Zeiler, M. D., Krishnan, D., Taylor, G. W., and Fergus, R. Deconvolutional Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2010), pp. 2528–2535.

[242] Zhong, E., Fan, W., Yang, Q., Verscheure, O., and Ren, J. Cross validation framework to choose amongst models and datasets for transfer

learning. In *European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)* (2010), pp. 547–562.

[243] Zou, W. Y., Ng, A. Y., and Yu, K. Unsupervised learning of visual invariance with temporal coherence. In *NIPS 2011 Workshop on Deep Learning and Unsupervised Feature Learning* (2011).