FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Contributions on Deep Transfer Learning

Chetak Kandaswamy

MAP-tele Doctoral Program in Telecommunications

Supervisor: Jaime dos Santos Cardoso (PhD)

Co-supervisor: Luís Miguel Almeida da Silva (PhD)

January 20, 2016

Contributions on Deep Transfer Learning

Chetak Kandaswamy

MAP-tele Doctoral Program in Telecommunications

January 20, 2016

Abstract

Machine Learning (ML) is reshaping our world by building machines that can see, read, listen, talk, and write (Marr, 2015). For example, imagine you want to eat your favorite food and ask a mobile app what are the nearby restaurants that serve your favorite food. The app uses a ML algorithm to analyze the information of your location and time to provide better service. Imagine a startup that wants to use ML algorithm optimize their resources and requirements. Envision a scenario where a breast cancer patient and a surgeon want to plan a surgery. Generally, it takes a highly trained surgeon's subjective evaluation of the breast cancer surgery to make treatment decisions. Employing ML algorithms allows an objective evaluation of the surgery with a sense of femininity (aesthetic metric by (Beadle et al., 1984)), using 3D imaging that helps the patient and the doctor to visualize and collectively decide what they are seeing, and then make a informed decision about the surgery (Oliveira, 2013).

The ability of ML algorithms to automatically understand the contents of images opened new ways to tackle prominent computer vision challenges like view point variation, illumination, occlusion, scale, deformation, background clutter, and intra-class variation. Especially, Deep learning algorithms like Stacked Denoising Autoencoders, Convolutional Neural Networks, and Deep Belief Nets have addressed these challenges with state-of-the-art results and have surpassed human performance. The salient attribute of Deep learning algorithms is that they automatically learn features from a large number of data samples without overfitting the model. Deep learning provides a partial solution to the research statement *how to get computer programs to self-learn patterns from data*. We consider machine learning as a broadening field which integrates interdisciplinary knowledge of learning processes from other fields such as psychology, biology, neuroscience, and economy. From this we attempt to answer questions such as: what kind of process can lead to learning, under what conditions, and for what kind of data? (Mitchell, 2006) In this thesis we integrated knowledge of deep learning with transfer learning where the learning is inspired from human-like learning processes to build our Deep Transfer Learning framework:

> "*how to get computer programs to self-learn patterns in data-efficient and domain-general way?*"

Our survey on the state-of-the-art machine learning methods that use self-learn patterns lead to deep learning models. The subsequent question is about how learning algorithms can be inspired from human-like learning processes. For the sake of simplicity, we narrowed our scope down to human like knowledge transfer from one scenario to another.

It is interesting to study the various possible cases of transfer learning settings based on the distributions, posterior probabilities, learning function and the classification tasks. We have designed three main mechanisms that harness the advantages of our proposed Deep Transfer Learning (DTL) framework: 1) Layerwise Transfer Learning (LTL), 2) Source-Target-Source (STS) and 3) Deep Transfer Learning Ensemble. First, we developed two approaches based on LTL mechanism : a) Transfer Learning unsupervised (TLu) and b) Transfer Learning supervised (TLs). On

the other hand, STS mechanisms were tested for both single and multi-source problems. Finally, we developed DTLE as an ensemble of various LTL approaches. To analyze the effectiveness we assess the designed DTL framework for practical applications like drug discovery and cross-sensor biometric identification. We also implemented a DTL software based on an interactive interface for GPU based machine learning algorithms. This software provides an interface to baseline and various transfer learning methods.

Resumo

Aprendizagem Computacional (AC) consiste na criação de máquinas capazes de ver, ler, ouvir, falar e escrever (Marr, 2015). Por exemplo, imaginando que desejamos comer o nosso prato favorito e nos aconselhamos com uma aplicação móvel relativamente a restaurante próximos que sirvam tal prato, a aplicação utilizará um algoritmo de AC para analisar a informação relativa à localização do utilizador de modo a proporcionar o melhor serviço possível. No caso alternativo da criação de uma startup tentando cimentar o seu negócio, um algoritmo de AC facilitará o processo de como economizar recursos suficientes. Ainda num terceiro caso, num cenário em que um paciente de cancro da mama e um cirurgião querem planear uma cirurgia, as decisões são geralmente tomadas através de uma avaliação subjetiva levada a cabo por um cirurgião altamente especializado. A utilização de algoritmos de AC nesta aplicação específica introduziu um maior grau de objetividade (Beadle et al., 1984) através de informação de imagiologia 3D, ajudando tanto o médico como o paciente a atingir coletivamente uma decisão bem informada relativamente à cirurgia (Oliveira, 2013).

A capacidade de algoritmos de AC de automaticamente compreender o conteúdo de imagens abriu novas possibilidades de encarar desafios proeminentes de visão computacional, como variações de ponto de vista, iluminação, oclusão, escala, deformação, variabilidade de background ou variação intra-classe. Em especial, algorithmos de aprendizagem profunda, como deep algorithims: Stacked Denoising Autoencoders, Convolutional Neural Networks ou Deep Belief Nets mostraram-se capazes de abordar estes desafios e apresentar performance ao nível do estado da arte, ultrapassando largamente o potencial humano para realizar tais tarefas. O atributo mais saliente relativo a algoritmos de aprendizagem profunda foca-se na sua capacidade de automaticamente aprenderem características a partir de uma grande quantidade de dados, sem causar overfitting dos modelos treinados. Aprendizagem profunda permite uma solução parcial à questão comum nesta área de investigação relativa a *como conseguir que computadores sejam capazes de entender de forma autónoma padrões em grandes quantidades de dados*. De modo a atingir este último desafio, a área de aprendizagem computacional tem alargado o seu espectro interdisciplinar através da integração de conhecimentos de especialistas de outras áreas, como a psicologia, a biologia, as neurociências ou a economia, de modo a entender que processos humanos conduzem à aprendizagem e integração de conhecimentos dada uma série de dados sob uma série de condições (Mitchell, 2006). Nesta tese, integrámos o conhecimento de aprendizagem computacional profunda com transfer learning, em que a aprendizagem é inspirada no processo humano de aprendizagem, para construir a nossa framework de Deep Transfer Learning:

> "*como conseguir que computadores sejam capazes de aprender, de forma autónoma, padrões de maneira eficiente em termos de dados, e para domínio generalizado?*"

Na presente tese o foco de investigação recai sobre métodos de AC capazes de auto-aprenderem padrões através da utilização de métodos de aprendizagem profunda. A questão subsequente prende-se com como desenvolver tais metodologias de forma inspirada no processo de aquisição

de conhecimento do ser-humano. Como simplificação, reduzimos o espectro de interesse na transferência de conhecimento entre duas tarefas de modo a utilizar conhecimento adquirido numa para resolver outra.

É interessante estudar uma série de possíveis variações no processo de transferência de aprendizagem, quer ao nível das distribuições, probabilidades posteriores, funções de aprendizagem e também tarefas de classificação. Foram criados três mecanismos base da framework de deep transfer learning: 1) Layerwise Transfer Learning (LTL), 2) Source-Target-Source (STS) e 3) Deep Transfer Learning Ensemble. Relativamente à primeira metodologia, duas abordagens foram tomadas: a) Transferência de conhecimento não-supervisionada (TLu) e b) Transferência de conhecimento supervisionada (TLs). Por outro lado, os mecanismos de STS desenvolvidos foram testados tanto em problemas com uma fonte de informação como em problemas de múltiplas fontes. Finalmente, foram também desenvolvidas e testadas metodologias ensemble de várias abordagens LTL. De modo a analisar a eficácia dos métodos desenvolvidos, testes práticos foram levados a cabo ao nível de descoberta de fármacos e também de biometria inter-sensor. Um software de DTL baseado em interface interativa para algoritmos de AC baseados em GPU foi também implementado. Este software fornece uma interface para problemas baseline e outras metodologias de transferência de conhecimento.

To my mother and father, to whom I owe everything.
To my sister and my brother, for being driving force in my life.

Acknowledgments

I would like to express my sincere gratitude to my advisor Professor Jaime dos Santos Cardoso for the continuous support of my PhD study and research, for his patience, motivation, enthusiasm and immense knowledge. The joy and enthusiasm he has for his research was contagious and motivational for me, even during tough times in the PhD pursuit. I am also thankful for the excellent example he has provided me as a successful researcher and Professor. I could not imagine having a better advisor and mentor for my PhD study.

Furthermore, I am very grateful to my co-advisor Professor Luís Miguel Almeida da Silva, for the opportunity to develop my PhD study in the area of deep learning, for the insightful comments both in my work and in this thesis, for her support and for many motivating discussions. I appreciate all his contributions of time, ideas, and funding to make my PhD experience productive and stimulating. His guidance helped me in all the time of research and writing of this thesis.

I extend my gratitude to NNIG (Neural Network Interest Group), INEB for all the conditions provided. I am very glad to have had the possibility of being embraced by this group. I am especially gra teful to: Luís. A. Alexandre, Jorge. M. Santos and Joaquim Marques de Sá. A very special word of gratitude is due to Ricardo Sousa, Tiago Esteves and Telmo Amaral, for their constant support and encouragement.

I extend my sincere thank to INESC Porto, for all the work conditions, infrastructure and technical support given to my PhD thesis and for the other challenges I worked in. I am especially grateful to all the members of the Visual Computing and Machine Intelligence (VCMI) group have contributed immensely to my personal and professional time. I am especially grateful to: Ana Rebelo, Ines Domingues, A. Filipa Sequeira, Samaneh Khoshrou, Eduardo Marques, João. C. Monteiro, João. P. Monteiro, Hooshiar Zolfagharnasab, Kelwin Fernandes and Helder Olivera. The group has been a source of friendship as well as motivation and collaboration. I thank FEUP for providing the support and conducive environment for my PhD. I would like to than Jonathan Barber for his support for building tools for the research work.

I shall also dedicate a few words to someone whose influence in my life spanned out the time I was his student from the degree to become a researcher. The way I reason about Mathematics, Science, Knowledge and Life itself was deeply influenced by the many things he taught me about. Thank you Professor Girish M Chandra.

I thank Professor Manuel Alberto Pereira Ricardo and Professor Rui Campos who are source of inspiration and provided me the guidance for building my research capabilities. I also like to thank Professor Adriano Moreira for believing in my professional ability and provided me the opportunity to pursue my PhD.

I thank Professor Helmut Prendinger for the opportunity to work in his lab on drones at Tokyo. I thank Andrew Holliday, Danielle Delatte, Johannes Laurmaa, Kim Samba, Pierre Ecarlat, Ragevendra Jain, Ruben Geraldas and Naoki Tada for their support during the research at National Institute of Information, Tokyo.

A special word of thanks to my friends without whose encouragement and unconditional support it would be a challenge: Saravanan Kandasamy, Kumaresa Vanji, Abhishek Chatterjee, Balamurugan Varadharajan, Herlina Jayadianti and Merle Planten. I would like to thank my Portuguese Professor Magarida Mouta who helped me to get along with culture shock and enjoy the interaction with beautiful land of Portugal. Thank you for your support when I have needed it the most.

For the ancestors who paved the path before me upon whose shoulders I stand. Kandaswamy family who have been my constant source inspiration, competition and encouragement. To my wife Jivitha Anand who became my source of encouragement and to my six months old baby Adya, you mean the world to me.

Thank you with all my heart!

Chetak Kandaswamy

"Machine Learning is a core,
transformative way by which we (academicians &
industrial specialist) are rethinking how we're doing everything"

Sundar Pichai

Contents

List of Figures

List of Tables

Acronym

BL	Baseline
cBL	Combined Baseline
CNN	Convolutional Neural Network
conv	Convolutional layer
CPA	CellProfiler Analyst
CPU	Central Processing Unit
CSIP	Cross-sensor Iris and Periocular
dA	denoising Autoencoder
DBN	Deep Belief Nets
DNA	Deoxyribonucleic acid
DNN	Deep Neural Network
DTL	Deep Transfer Learning
DTLE	Deep Transfer Learning Ensemble
DTLEr	Retrain Specific Deep Transfer Learning Ensemble
DTLEt	Transfer Specific Deep Transfer Learning Ensemble
FC	Fully-connected layer
FCN	Fully Convolutional Network
FT	Finetune
GMM	Gaussian Mixture Models
GPU	Graphical Processing Unit
HCA	High Content Analysis
HT	Hard Transfer
IDSM	individual specific models
JS	Jensen-Shannon
JSD	Jensen-Shannon Divergence

KL	Kullback-Leibler
LBP	Local Binary Pattern
LOOCV	Leave-One-Compound-Out Cross Validation
LR	Logistic regression
LTL	Layerwise Transfer Learning
MAP	Maximum a Posterior
MCF7-wt	Breast Cancer Expressing wild-type p53
MFCC	Mel-frequency Cepstral Coefficients
ML	Machine Learning
MOA	Mechanisms of Action
MSCOCO	Microsoft Common Object in Context
MS-STS	Multi-source Source-Target-Source
NICE	Noisy Iris Challenge Evaluation
NN	Neural Network
pool	Pooling layer
PT	Pre-train
RBF	Radial Basis Function
RBM	Restricted Boltzmann Machines
RT	Reverse Transfer
SAA	Stacked Autoencoder
SDA	Stacked Denoising Autoencoder
SGD	Stochastic Gradient Descent
STS	Source-Target-Source
SV	Support Vector
SVM	Support Vector Machines
TL	Transfer Learning
TLs	Transfer Learning supervised
TLu	Transfer Learning unsupervised
UAV	Unmanned Aerial Vehicle
UBM	Universal Background Model
WiFi	Wireless Fidelity

Part I

Introduction and Related work

Chapter 1

Introduction

Machines have become an essential part of our everyday life. Towards the end of the last century, smart machines outperformed humans in mundane or highly specific tasks. These machines use algorithms that are trained to automatically learn general laws from specific training data. Algorithms such as deep neural networks, support vector machines, Bayesian methods and many more have contributed to a wide range of applications including biomedical applications for drug discovery, data mining applications for detection of traffic signs, self-driving cars, biometric sensor interpretations, location identification of a person based on his wireless fidelity (WiFi) data, and aerial surveillance using swarm of unmanned aerial vehicles (UAVs). While these algorithms demonstrate the practical importance of machine learning methods, researchers are actively pursuing more effective algorithms. Some of the interesting application of machine learning methods are now briefly discussed.

Computer Vision: Object recognition. The early success in hand-written digit recognition by convolutional (or time-delay) neural networks laid stepping stone for many computer vision applications. Semantic segmentation, object recognition, image recognition and 3D objects recognition in natural images are among the main examples.

Telecommunications: WiFi-Based Indoor Localization. An interesting problem faced by ubiquitous computing and social networking community is locating the smartphone user position in an indoor environment with WiFi data. This indoor WiFi localization problem is a challenge as it is very expensive to calibrate WiFi data for building localization models in a large-scale environment. Moreover, it is known that the WiFi signal-strength values are function of time, device, and other dynamic factors. Machine learning algorithms are used to reduce the recalibration efforts by adapting to the dynamic changes in time and devices.

Biomedical Applications: Breast cancer drug discovery. Machine learning algorithms has paved a new way to better utilize the vast patients' data for better and faster drug discovery and diagnosis of patients. Areas like early detection of Breast Cancer using Mammogramy Images and Magnetic Resonance Imaging of breast have benefited from such methods.

While these applications demonstrate the practical importance of machine learning methods, researchers are actively pursuing more effective algorithms.

1.1 Motivation

The world we live in requires knowledge of many things. We learn these things by continuously interacting with the external environment and developing skills accordingly.

We learn to play football for the fun of kicking the ball. Yet for playing football we require many other basic skills. To be a successful player, we use our previous knowledge of walking, running, jumping, kicking, etc. Our brain continuously learns to interact with the external environment and learns these specialized behavior patterns which may lead to winning. Inspired by this natural and continuous human learning, we attempt to train machines to mimic such human-like learning structures. The machine continuously learns and reuses its knowledge to solve different and specialised tasks, and this enables it to develop a wide knowledge base.

The world we live in presents both good and bad opportunities to learn. Football is fun to play, as long as we curb our habits which may prove to be counter productive. To be effective we should avoid playing without properly warming up first and look to leverage our strengths to positively affect the overall performance. The same is true even for machines. Training an algorithm with adverse knowledge produces negative performance on the intended task, and the resulting solution itself may falls into a local minima. It will be beneficial to utilize the adverse knowledge as a means and not as an end result.

1.2 Thesis Statement

The machine learning community in general has addressed challenges focused on the narrow view of the research question *how to get computer programs to learn some class functions from examples?* To illustrate the limitation of this narrow view *we do not have computer programs that have the ability to think like people*. To break this view the community focused on Alan Turing's ambitious research question: *Can machines think?* (Turing, 1950) which faced severe challenges on the definitions of machine and thinking. The question was later softened with *Can machines do what we (as thinking entities) do?* (Kurzweil, 2005). As a pragmatic option towards the goal of the General Intelligence paradigm, Tom Mitchell proposes a broader interdisciplinary view of machine learning involving computer programmers and statisticians who have already contributed to statistical-computational theories of learning processes. Together with other field experts like psychologist, economists, biologists and neuroscientists, they collectively questioned *What kind of process can lead to learning under what conditions for what kind of data?* (Mitchell, 2006).

This document attempts to address the fundamental question of *how to get computer programs to self-learn patterns from data*. To this inquiry, we integrate interdisciplinary knowledge of human learning processes from other fields, for example: 1) psychological studies of the human ability to easily adapt the learning from one situation to suit or adjust to another situation with minimal or no deviation (Perkins and Salomon, 1992), 2) neurological studies like the human ability to perceive images with the hierarchical working structure of the neocortex (Hubel and Wiesel, 1959), and 3) other psychological studies of the human ability to continuously learn

new processes (London and Sessa, 2007). In this thesis, we are interested in designing machine learning algorithms that are motivated by the above research question inspired from human-like learning processes.

1.3 Objectives

The main objective of this research is to develop a machine learning framework that attempts to self-learn the patterns and reuse extracted information from the data, enabling it to express the information as understandable by humans while making it possible to compete with state-of-the-art technology. In other words, the research aims to create an automated feature extractor that can be used to solve various tasks in spite of the tasks being different from each other. The core idea is to reuse the experience gained in learning to perform one or more tasks to help improve the learning performance of other tasks. Although, sharing or reusing the knowledge may lead to either an improved (positive) or degraded (negative) performance. In this work we intend to curtail negative learning or at least maintain the same performance as in the case of no sharing of knowledge.

In this sense, the self-learning feature extractor should produce generic and reusable features for multiple tasks from different domains. The performance of the designed framework is to be evaluated on various computer vision benchmark data as well as in two problem specific applications: a biomedical application for drug discovery in breast cancer cells and sensor applications for person identification from multiple sensory data.

1.4 Contributions and Related Publications

A summary of the contributions of the thesis is as follows:

1. We have designed a Deep Transfer Learning (DTL) framework by combining the advantage of the hierarchical feature representation property of deep networks with the feature reuse property of Transfer Learning. This synergy led to the development of a self-learning feature extractor that produces generic and reusable features for solving multiple tasks. The DTL framework produced three mechanisms inspired by the human learning process that help to solve major challenges of machine learning problems:

   (a) A layer-wise feature transference mechanism to reuse extracted features initially trained on a source domain and tested on a target domain with little modification of the model; this mechanism indeed enhanced the performance for many challenging computer vision datasets, but is limited to reuse only features of source problems that lead to positive feature transference;

   (b) A Source-Target-Source mechanism, where the layer-wise feature transference is optimized by switching between multiple domains (both source and target) and thus expanding the optimal solution search space;

(c) A Deep Transfer Learning Ensemble mechanism where the layer-wise feature transference mechanism is combined with the traditional ensemble learning.

2. We have investigated our designed layer-wise feature transference mechanism for application specific scenarios, such as the analysis of breast cancer cell images for drug discovery.

3. We extended our Source-Target-Source mechanism with a multi-source version for cross-sensor biometric classification applied to the identification of human anatomical structure in the periocular region.

Finally, we created a user interface for our DTL framework utilizing GPU parallel processing capabilities, which can be used by machine learning researchers to compare their methodologies and/or help in solving real problems in the field.

List of Publications arising from this thesis

Journal papers:

- Kandaswamy, C., Silva, L.M., Alexandre, L.A., and Santos, J.M. "High-content Analysis of Breast Cancer using Single-Cell Deep Transfer Learning", Journal of Biomolecular Screening, SAGE, January 8, 2016, doi: 10.1177/1087057115623451

- Kandaswamy, C., Monteiro, J.C., Silva, L.M and Cardoso, J.S. "Multi-source Deep Transfer Learning for Cross-sensor Biometrics." Neural Computing and Applications, 1-15, 2016, doi: 10.1007/s00521-016-2325-5

Conference papers:

- Kandaswamy, C., Silva, L. M., Alexandre, L. A., and Santos, J. M. Deep transfer learning ensemble for classification. In *Advances in Computational Intelligence*, pages 335–348. Springer, 2015a. doi: 10.1007/978-3-319-19258-1_29

- Kandaswamy, C., Silva, L. M., and Cardoso, J. S. Source-target-source classification using stacked denoising autoencoders. In *Pattern Recognition and Image Analysis*, pages 39–47. Springer, 2015b

- Kandaswamy, C., Silva, L. M., Alexandre, L. A., Sousa, R., Santos, J. M., de Sá, J. M., et al. Improving transfer learning accuracy by reusing stacked denoising autoencoders. In *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 1380–1387. IEEE, 2014b. doi: 10.1109/SMC.2014.6974107

- Kandaswamy, C., Silva, L. M., Alexandre, L. A., Santos, J. M., and de Sá, J. M. Improving deep neural network performance by reusing features trained with transductive transference. In *Artificial Neural Networks and Machine Learning–ICANN 2014*, pages 265–272. Springer, 2014a. doi: 10.1007/978-3-319-11179-7_34

Workshop paper:

- Kandaswamy, C., Silva, L.M., Cardoso, J.S. "Improving Classification Accuracy of Deep Neural Networks by Transferring Features from a Different Distribution," 20th edition of the Portuguese Conference on Pattern Recognition, University of Beira Interior, Covilhã, 2014.

Conference papers in collaboration:

- Amaral, T., Kandaswamy, C., Silva, L. M., Alexandre, L., Marques de Sá, J., and Santos, J. M. Improving performance on problems with few labelled data by reusing stacked auto-encoders. In *International conference on Machine Learning and Applications (ICMLA)*, pages 367–372. IEEE, 2014a. doi: 10.1109/ICMLA.2014.65

- Amaral, T., Silva, L. M., Alexandre, L. A., Kandaswamy, C., de Sá, J. M., and Santos, J. M. Transfer learning using rotated image data to improve deep neural network performance. In *Image Analysis and Recognition*, pages 290–300. Springer, 2014b. doi: 10.1007/978-3-319-11758-4_32

- Amaral, T., Silva, L. M., Alexandre, L. A., Kandaswamy, C., Santos, J. M., and de Sá, J. M. Using different cost functions to train stacked auto-encoders. In *Mexican international conference on artificial intelligence (MICAI)*, pages 114–120. IEEE, 2013. doi: 10.1109/MICAI.2013.20

1.5 Structure of the Thesis

This thesis is organized into three parts. The first part includes a research introduction and a literature review. The second part discusses the theoretical modeling of the designed DTL framework (in Chapter 3, 4, 5, and 6). The third part discusses the application specific design of the DTL framework, thesis conclusions, and ideas for future work (in Chapter 7, 8 and 9). Finally appendix A discusses on one of the future work application.

Chapter 2

Related work

Computer algorithms that improve automatically through experience without being explicitly programmed have been the key research question of the machine learning community for the past fifty years. This technological need gave rise to an abundant variety of learning algorithms that are used in speech recognition, computer vision, data mining, and many other applications (Bishop, 2006). In this chapter we discuss only the most relevant state-of-the-art Machine Learning (ML) and Transfer Learning (TL) algorithms along with their applications and limitations, in the perspective of our defined objectives discussed in the previous chapter. A background knowledge on artificial neural networks, probability theory, and optimization are not essential.

In Section 2.1, we analyze the pros and cons of feature extraction since its inception as handcrafted features to the present day automated processes. In Section 2.2, we examine the knowledge transfer in machines[1] model, approaches, and limitations. In Section 2.3 we consider research methods of the established feature transference methods with state-of-the-art feature extraction processes, common practices, and pitfalls.

2.1 Trends in feature extraction methods in ML

In this section we briefly discuss the major trends of feature extraction methods in the Machine Learning field starting from the early 1960's to the present day. From a literature survey and keyword usage search, we identified major trend changes in the perspective of the machine learning community and categorized these trends into three evolutionary stages of feature extraction methods. A timeline depiction of these trends along with their evolution is shown in Figure 2.1.

From the early 1960's till 2006, the community answered queries on how to build methods which transform the collected raw data into a form that a computer can handle. These are first generation feature extraction methods appearing at a time when feature extraction was considered as a field only for specialists who generally used carefully handcrafted features for each learning problem.

[1] Commonly referred to as Transfer Learning.

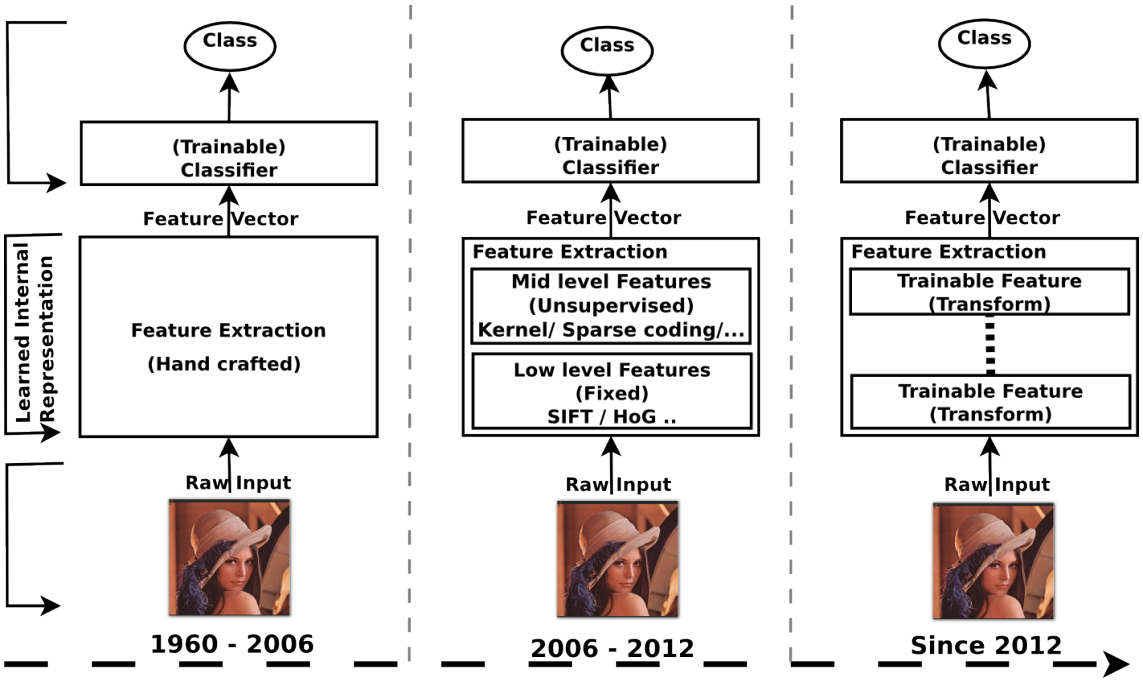


Figure 2.1: Evolution of feature extraction methods from 1960's to 2015.

Around 2006 the community devised low level feature extractors like Scale-Invariant Feature Transform (SIFT) (Lowe, 1999), Histogram of Oriented Gradients (HoG) (Dalal and Triggs, 2005) for object recognition, and Mel-Frequency Cepstral Coefficients (MFCC) (Davis and Mermelstein, 1980) for speech recognition. These methods were used in a wide variety of applications, thus heralding the second generation of feature extraction methods. SIFT is called a local feature descriptor and allows a point inside an RGB[2] image to be represented robustly by a low dimensional vector. When you take multiple images of the same physical object while rotating the camera, the SIFT descriptors of corresponding points are very similar in their 128-D space. After the community shifted towards more ambitious object recognition problems and away from geometry recovery problems, we had a flurry of research in Bag of Words, Spatial Pyramids, Vector Quantization and machine learning tools used in any and all stages of the computer vision pipeline. HoG came at a time when everybody was applying spatial binning to bags of words, using multiple layers of learning and making their systems overly complicated. HoG was quite simple and well understood since it was a linear Support Vector Machine (Tomasz, 2015).

In 2012, the community at large began asking how machines can self-learn representations from data. For example, represented data must be in the space of the learner such that it can be classified. This led to a third generation of feature extraction methods in the form of Trainable Feature Transform, which substitutes traditional handcrafted feature extraction with automated feature extraction. To illustrate, when Trainable Feature Transform understands a scene of a man standing, it first learns by distinguishing the pixels of the man with the background, then the lines or edges and finally the object of the man (Bengio, 2009). This has become a new model

[2]RGB is a color model in which the red, green, and blue are added in different ways to reproduce broad array colors

for representing data, which is generally based on deep neural network architectures and more popularly known as Deep Learning.

What about other widely used algorithms? Machine learning algorithms such as decision trees, nearest neighbor, logistic regression, Bayesian network, multi-layer neural networks, support vector machines (SVM), and random forest may indeed produce reasonably effective methods for a vast array of applications but are limited by feature extraction methods that are mostly handcrafted or have low-level features. To understand the state-of-the-art trends in the machine learning field, we conducted a survey using *Google keyword search* on worldwide data for the last 10 years with respect to most popular algorithms such as support vector machines, deep learning and random forest. We observed that around the year 2012 the field of Machine Learning gained popularity along with deep learning and random forest algorithms (See Figure 2.2, full report on Google Trends) (Google, 2015). To understand these trends in detail, we further studied various applications and competitions held in the field of Machine Learning. We observed that deep learning was not only used in a wide variety of applications, but also revolutionized the Machine Learning field in the past decade.

Convolutional Neural Network (CNN) (LeCun et al., 1998) is a machine learning algorithm belonging to the family of deep neural networks whose architecture of alternating convolutional layers and subsampling layers was inspired by the alternating structure of simple and complex cells in the primary visual cortex (Hubel and Wiesel, 1959). Below is a list of applications in particular to competitions won by CNN among all other algorithms like Decision Trees, Nearest Neighbor, Support Vector Machines, Random Forest and Bayesian Networks. For more on current state-of-the-art results in object classification visit Rodrigo Benenson's website `http://rodrigob.github.io/` (Benenson, accessed January 12, 2016).

Application: [Year] competition (Group won)

- Handwriting recognition:[Many] MNIST & Arabic (IDSIA)
- Volumetric brain image segmentation: [2009] connectomics (IDSIA, MIT)
- OCR in the Wild [2011]: StreetView House Numbers (NYU and others)
- Traffic sign recognition: [2011] GTSRB competition (IDSIA, NYU)
- Breast cancer cell mitosis detection: [2011] MITOS (IDSIA)
- Human Action Recognition: [2011] Hollywood II dataset (Stanford)
- Scene Parsing: [2012] Stanford bgd, SiftFlow, Barcelona (NYU)
- Speech Recognition: [2012] Acoustic modeling (IBM and Google)
- Pedestrian Detection: [2013] INRIA datasets and others (NYU)
- Large Scale Visual Recognition: [2013] ImageNet dataset (NYU)
- Large Scale Visual Recognition: [2014] ImageNet dataset (GoogLeNet)
- Large Scale Visual Recognition: [2015] ImageNet dataset (MSRA, AMAX)

These achievements were made possible due to a breakthrough in training neural nets to self-learn the representation one layer of neurons at a time from a large number of training samples (labeled and unlabeled) without overfitting. This made deep learning appear more efficient when

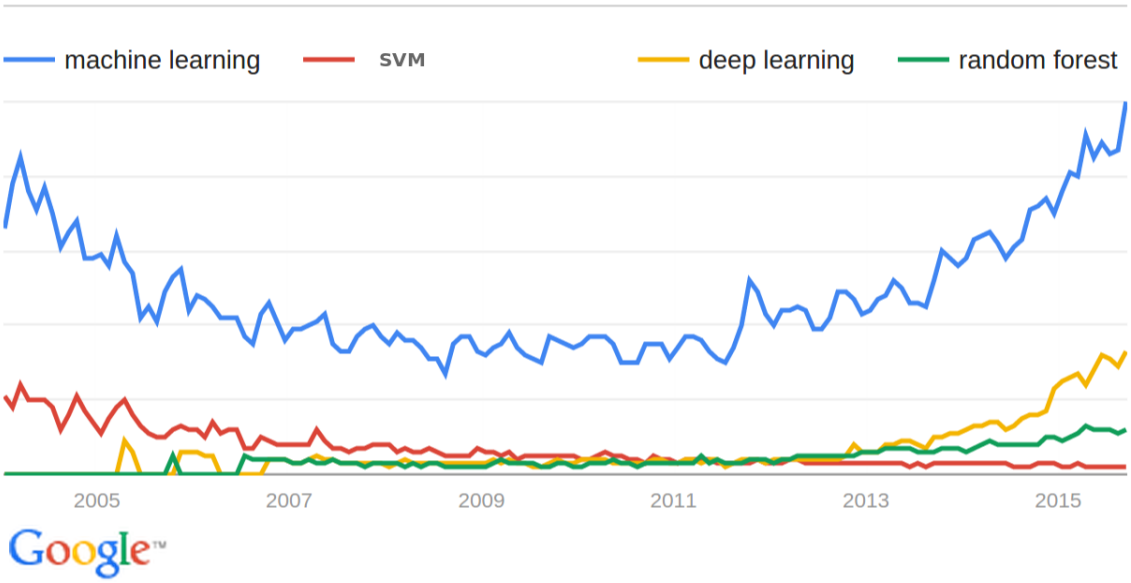


Figure 2.2: Worldwide interest over time in the field of Machine Learning with Deep learning, Support Vector Machines and Random Forest. Keyword usage trends from 2004 to present.

compared to other algorithms. This ability along with the availability of low cost parallel computational capabilities developed wide acceptance among varied machine learning groups all over the world.

Let's begin by understanding what is deep learning. To learn the skill of running one has to know the basics of balancing and walking. Similarly, deep learning intends to first learn the basic representation structures and then reuse these structures to develop more specific and abstract feature representations of the data. Deep Learning allows computational models that are composed of multiple processing layers to learn representations of data with *multiple levels of abstraction* (LeCun et al., 1998).

In the next subsections we discuss some of the popular deep learning methods: Convolutional Neural Networks (LeCun et al., 1998), Stacked Denoising Autoencoders (Vincent et al., 2010) and Deep Belief Nets (Hinton et al., 2006).

2.1.1 Convolutional Neural Networks (CNN)

The research of Neocognitron by Fukushima et.al, introduced CNN as a self-organizing neural network which is unaffected by shift in position for pattern recognition (Fukushima, 1980). This work was later improved by Yann Lecunn et.al, by training a multi-layer neural network with the back-propagation algorithm for gradient based learning (LeCun et al., 1998).

Convolutional Neural Networks take advantage of the fact that the input consists of images by constraining the architecture in a more sensible way. In particular, and unlike a regular Neural Network, the layers of a CNN have neurons arranged in three dimensions: width, height, and depth. (Note that the word depth here refers to the third dimension of an activation volume, not

to the depth of a full Neural Network, which can refer to the total number of layers in a network.) The CNN architecture transforms the full image into a single vector of class scores, arranged along the depth dimension. This process is illustrated in Fig 2.3.

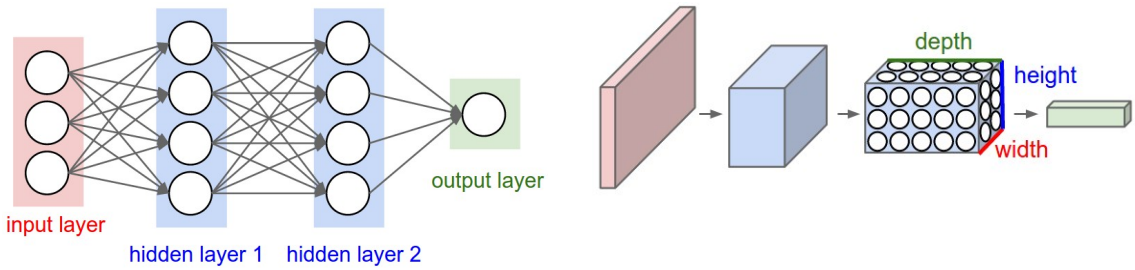


Figure 2.3: **Left:** A regular 3-layer Neural Network. **Right:** A CNN arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a CNN transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be three (Red, Green, Blue channels) (Li and Karpathy, 2015).

CNNs exploit *spatially-local correlation* by enforcing a local connectivity pattern between neurons of adjacent layers. The architecture thus ensures that the learned filters produce the strongest response to a spatially local input pattern. Also, *sharing weights* increases the invariance of learned filters by replicating each filter across the entire visual field. These replicated filters share the same parametrization (weight vector and bias) and form a feature map. Replicating units in this way allows for features to be detected regardless of their position in the visual field. Additionally, weight sharing increases learning efficiency by greatly reducing the number of free parameters being learned. The constraints on the model enable CNN to achieve better generalization on vision problems.

2.1.2 Stacked Denoising Autoencoders (SDA)

An autoencoder is a simple neural network with one hidden layer designed to reconstruct its own input. For that reason, it hs an equal number of input and output neurons. The learning accuracy is obtained by minimizing the average reconstruction error between the original and the reconstructed instances. The encoding and decoding feature sets (input-hidden and hidden-output weights, respectively) may optionally be constrained as transposes of each other. In this case the autoencoder is said to have tied weights. A denoising Autoencoder (dA) (Vincent et al., 2008) is a variant of the autoencoder where now a corrupted version of the input is used to reconstruct the original instances. Moreover, the dA makes an excellent building block for deep networks (Bengio, 2012, Section 5.4). Stacking multiple dA's one on top of each other gives the model the advantage of hierarchical features with low-layer features represented at lower layers and higher-layer features represented at upper layers (Bengio, 2012, Section 3).

2.1.3 Deep Belief Nets (DBN)

Deep Belief Nets are a specific type of energy-based model that attempts to learn low-energy state for a desired variable (Salakhutdinov and Hinton, 2009). The DBNs are a fully general Boltzmann machines (Hinton et al., 1984), in which the connections between the hidden units are restricted in such a way that the hidden units form multiple layers. Restricting the fully connected network in Boltzmann machines improves the speed and accuracy, latter coined as restricted boltzmann machines (RBM) (Salakhutdinov et al., 2007). Using a greedy layer-wise pre-training (Hinton et al., 2006) for training stacked RBM paved the way for Deep architectures leading to Deep Belief Nets (Hinton et al., 2006) and Deep Boltzmann Machines (Salakhutdinov and Hinton, 2009). DBNs perform better than the Stacked Autoencoders in cases where they have access to prior probability distribution are available (Holst et al., 2015).

2.2 Model for Transfer Learning in ML

The initial works on transfer learning began with the 1995 NIPS workshop. The Defense Advanced Research Projects Agency (DARPA) funded in 2005 a Transfer Learning (TL) program to increase the interest in TL challenges and potential contributions (Gasser et al., 2005). The Multitask learning by (Caruana, 1997) explores the simultaneously learning of multiple tasks by sharing the weights of the network. The core idea of sharing weights increases learning of model that solves many tasks with good generalization. The Multitask Learning approach is different from multiclass learning, the model learns c different output variables $\{Y_1, Y_2, ..., Y_c\}$ corresponding to n different tasks. The Multitask Learning learns multiple source tasks simultaneously by sharing the knowledge (features) between the tasks or conditional models.

In Lifelong Learning, multiple related source tasks are learned one after another by the model in an incremental approach; to solve for nth (target) task the model needs to learn serially all source model tasks up to the $(n-1)$ task. The previous learning helps to solve the new target task. However, if the source task(s) are not related to the target task causes degraded performance. It is also limited by the order in which the source tasks are presented. Learning to Learn (Thrun and Pratt, 2012) is a variation of Lifelong Learning (Thrun, 1998). The main intuition of this model is based on learning many tasks serially, one after another, under the assumption that learning the n-th task may be easier than the $(n-1)$-th task.

A large number of works have been produced with the name Domain Adaptation like (Blitzer et al., 2006), (Jiang, 2008), (Patricia and Caputo, 2014), (Ben-David et al., 2010) and (Bruzzone and Marconcini, 2010). Domain adaptation is a transfer learning framework which adapts the learner such that tasks between correlated domains[3] perform better than uncorrelated domains.

[3]The correlation between probability distributions (which allows estimating quantitatively how similar they are) can be empirically evaluated according to some similarity metrics. Hence, two domains are considered correlated if the distance between the corresponding underlying distributions is relatively small according to proper metrics, see (Bruzzone and Marconcini, 2010).

Domain adaptation expects that the closer the distributions of the problem are, the better the features trained on the source problem will perform on the target problem, thus limiting transfer learning problems to those for which the distributions are closely related.

Some traditional machine learning models can be used under different conditions for TL problems, like Semi-Supervised Learning, Ensemble methods, Bayesian priors, and meta-learning. The Semi-Supervised Learning explores how to learn from both labeled and unlabeled data, thus transferring the knowledge from source labeled data to the target unlabeled data (Zhu, 2006). The traditional Ensemble method combines a set of models to construct a complex classifier for a classification problem. Ensemble methods can be directly used in TL models by building the set of models from different distributions or tasks or problems. The various ensemble approaches and its solutions are discussed in (Jiang, 2008). Traditionally, in Bayesian priors models, we use a maximum a posterior (MAP) estimation approach for supervised learning to get Bayesian priors distributions. If the Bayesian priors are computed from the source domain labeled instances then this approach can be used easily for many TL problems. Even classical machine learning techniques such as rule induction may be easily leveraged to assist with TL applications. In meta-learning the reuse of meta-features (knowledge) or properties of the model is utilized to solve a new task (Vilalta and Drissi, 2002). Listed below are the most common methods for TL problems with unlabeled data (Zhu, 2006):

1. Use a trade-off parameter between the labeled and the unlabeled data. The trade-off parameter is calculated with Kullback-Leibler divergence[4] between the domains.

2. Use of a instance weighting method to factorize in both source labeled and target unlabeled instances during training.

3. Use of label propagation for unlabeled target data on a nearest neighbor graph.

Here we focus on the knowledge transfer based on the several survey works in the past decade discussing on overview and applications of transfer learning. The survey by Pan et.al, discusses several transfer learning frameworks including classification, regression, and clustering approaches in inductive, transductive, and unsupervised transfer settings (Pan and Yang, 2010). The difference between inductive and transductive learning is that the inductive learners can naturally handle *unseen data*, whereas the transductive learning will be used to contrast inductive learning. A learner is said to be transductive if it works only on the labeled and unlabeled training data, and cannot handle *unseen data*. Survey works of Van otterlo et.al, on TL for reinforcement learning are specific for relational domains (Van Otterlo, 2005) and another survey on reinforcement learning specify about the relation between between domains with only limited environmental feedback rather than correctly labeled examples (Taylor and Stone, 2009). Many specific transfer learning applications have been studied in detail: in activity recognition by (Cook et al.,

[4]Kullback-Leibler divergence measures the similarity of some distribution P to another distribution Q. It is not symmetric in P and Q.

2013), in bioinformatics by (Xu and Yang, 2011) and in Cross-domain collaborative filtering by (Li, 2011).

Studying the various surveys on TL we summaries the main goal of TL is to transfer the knowledge (learning) obtained from a *source* domain to one or more *target* domains in order to efficiently develop an effective hypothesis for a new task, domain or distribution (Ben-David et al., 2010). Brute-forcing knowledge from the source domain into the target domain, irrespective of their divergence, may cause a certain performance degradation or, in even worse cases, break the original data consistency in the target domain called as negative transference (Shao et al., 2015) or the performance the may exceed than the no transfer network (baseline approach) called as positive transference. This ambiguity in performances raises general issues regarding the transfer process: when to transfer?, what to transfer?, and how to transfer?

The answer to *when to transfer* includes the issues whether transfer learning is necessary for specific learning tasks and whether the source domain data is related to the target domain data. In scenarios where the training instances are sufficient, impressive performance can be achieved without any type of knowledge transference. However, we need to build models which adapt the gained knowledge to these scenarios in order to improve the overall performance of the new tasks.

The answer to *what to transfer* can be summed up in three approaches: the inductive transfer learning, using all the source domain instances and their corresponding labels for knowledge transfer; the instance transfer learning, relying mostly in the use of source domain labeled instances and also sometimes in target domain unlabeled instances; and the parameter transfer learning, using model parameters or hyper-parameters of the source domain for faster and accurate modelling of the target domain.

The answer to *how to transfer* includes all the specific transfer learning techniques. Knowledge transfer is based on the non-negative matrix trifactorization framework. The transfer learning phase is performed via dimensionality reduction (Shao et al., 2015). The most widely used methods transfer not only features but also parameters and hyper-parameters to the target domain.

In the literature, the term TL is used by different groups under different names and/or definitions. To have all these definitions under a single framework is challenging. In here, we use the most common underlying phenomena of TL that is very simple and generic among all the different definitions. We reuse the knowledge learned from a problem, or a set of problems, in a way that the knowledge gained helps to solve the new problem(s) more effectively. Inside this broad definition of TL, in the next section we discuss various methodologies for solving the above general issues that have been previously explored in the context of deep neural networks.

2.3 Challenges in Deep architectures using Transfer Learning

In recent years, there has been a growing interest in deep architectures for TL applications. One such interesting study inquired on how transferable are the lower layers' (generic) versus higher

layers' (specific) features in the hierarchical deep network (Yosinski et al., 2014). Another important work studied unsupervised TL using stacked autoencoders on a deep architecture model (Glorot et al., 2011). Concept drift is a study that incorporate the change in environment for big data (Gama et al., 2014). The feature transference approach for transferring top level CNN features for various object recognition problems have performed better in most of the cases (Razavian et al., 2014).

Even with good performance with most of the problems we foresee many shortcomings of these types of approaches: 1) the tasks have to be related, 2) the feature space and the learning space of the problem must be the same. Despite the vast body of literature on the subject (see (Glorot et al., 2011), (Bengio et al., 2013), (Bengio, 2012), (Ciresan et al., 2012)), there are still many contentious issues regarding TL problems from different distributions. TL methods, especially domain adaptation (Glorot et al., 2011) and multi-task learning (Bengio et al., 2013), are based on the assumption that both source and target problems are drawn from a related distribution. Even self-taught learning, which uses unlabeled data to train, needs both the source and the target datasets to be from the same modality (the input must be either image, audio or text only) (Raina et al., 2007).

Here we summarize the main challenges that are faced by the deep learning community using transfer learning approaches:

1. How transferable are the supervised and/or unsupervised features for correlated or uncorrelated domains?

2. How do we avoid or reduce the feature transferability that may lead to negative transference?

3. Is it possible to have transference across different network architectures? e.g. the number of layers and/or number of neurons.

4. Is it possible to have transference between different modalities? e.g. image to text or text to sound, etc.

5. Is it still possible to have effective TL in the case of large number of a target data training samples?

6. How do we select the target model if there are multiple models from different source domain problems?

2.4 Conclusions

It can be seen throughout the history of machine learning that some algorithms do better than others, but what makes the difference? Easily, the most important factor is the ability of machines to interpret data like humans. Then, how to enable machines to efficiently represent data without human interaction? Also, how to reuse the knowledge gained by learning problems to solve a new problem? The key to answer such questions may be in deep learning using TL approaches. Our

analysis of feature extraction methods and knowledge transference in terms of their capabilities and goals leads to this conclusion.

As guidance for future directions of this research work, we intend to utilize generic feature representations obtained by deep learning methods for multiple problems, tasks, or domains using the TL approaches. These computationally expensive hierarchical models trained on very large datasets will be harnessed. First we using parallel processing hardware that has scalable computational power like GPUs that provide higher performance of watt per bit. Second we improve the algorithm to use faster optimization techniques and TL approaches. In the subsequently chapters we discuss our proposed approaches and original contributions that would mitigate previously discussed challenges.

Part II

Deep Transfer Learning Mechanisms

Chapter 3

Deep Transfer Learning Framework

In this chapter our objective is to present an introduction of the basic Deep Transfer Learning concepts and techniques for classification of image based problems. We begin by introducing some of the necessary terminology and by describing fundamental concepts and operations associated with the process of transferring source problem knowledge (features) to target form suitable for training machines. In this chapter we discuss the deep transfer learning methodology that overcomes the multi-layer perceptrons limitation of overfitting and utilize a transfer learning technique for training on large unlabeled data samples. Some parts of this chapter are used in (Kandaswamy et al., 2014b) and (Kandaswamy et al., 2014a).

3.1 Fundamental concepts of deep learning

Deep Neural Networks (DNN) are very similar to ordinary Neural Networks (NN). NNs receive an input (a single vector) and transform it through a series of hidden layers. Each hidden layer is made up of a set of neurons, where each neuron is typically fully connected to all neurons in the previous layer, and where neurons in a single layer function completely independently and do not share any connections. The last fully-connected layer is called the "output layer" and in classification settings it represents the class scores. NNs do not scale well to full images: the full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting. Similarly, DNN's are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product with a weight vector and optionally applies a non-linear function. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. Softmax) on the last (fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply:

1. **Learning method:** Let us define a dataset by a set of tuples $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, $\boldsymbol{x}_i \in X_N$, $y_i \in Y_N$. The set $X_N = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ contains N instances of a d-dimensional random vector $X \subseteq \mathbb{R}^d$. Similarly, the set $Y_N = \{y_1, \ldots, y_N\}$ contains N instances of a one-dimensional random variable Y. Essentially, Y is a coding set for the labels using some one-to-one mapping (e.g.,

$\Omega =$ "*equilateral*","*circle*","*square*"$\} \rightarrow Y = \{0,1,2\}$ with number of labels $c = 3$). We assume that N instances are drawn by an i.i.d. sampling process from the input space X with a certain probability distribution $P(X)$.

Deep learning design has flexibility to utilize two main types of learning methods for training a classifier: supervised learning (when labeled training samples are given) or unsupervised learning (when labeled training samples are not available). In this context for supervised learning $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$ while for unsupervised learning $D = \{\boldsymbol{x}_i\}_{i=1}^{N}$.

2. **Logistic regression model** Logistic regression is a probabilistic, linear classifier. The score function $f : \mathbb{R}^d \mapsto \mathbb{R}^c$ that maps the raw image pixels to c number of class scores is given by a linear mapping equation as shown below:

$$f(\boldsymbol{x}_i, \boldsymbol{w}, \boldsymbol{b}) = \boldsymbol{w}^T \boldsymbol{x}_i + \boldsymbol{b} \tag{3.1}$$

where $\boldsymbol{w}$ are often called the weights (features) matrices mapping each input layer neuron to every output layer neuron, and $\boldsymbol{b}$ is the bias vector because it influences the output scores. The model's prediction y_{pred} is the class whose probability is maximal, specifically:

$$y_{pred} = \text{argmax}_y P(Y = y|x, \boldsymbol{w}, \boldsymbol{b}) \tag{3.2}$$

3. **Deep network model** Deep learning has multiple hidden layer and plus one linear classifier layer, thus a K layered deep network has $K-1$ number of hidden layers plus one classification layer. Given a dataset D a neural network attempts to learn features, represented as a vector $\boldsymbol{w}^j$ of optimal weights and biases. For a neural network with K number of layers, the features w^j are represented as a set of matrix of each layer, i.e., the set $W = \boldsymbol{w}^1_{M_1 \times M_2}, \ldots, \boldsymbol{w}^k_{M_k \times M_{k+1}} {}_{k=1}^{K}$

contains $M_k \times M_{k+1}$ weight matrix for each layer. Here M_k is the number of neurons in the k-th layer and M_{k+1} is the number of neurons in the $(k+1)$-th layer. The mapping function for deep network for k-th layer is as follows:

$$f\left(\boldsymbol{x}_i, \boldsymbol{w}^k, \boldsymbol{b}^k\right) = \{\boldsymbol{w}^k\}^T \boldsymbol{x}_i + \boldsymbol{b}^k \tag{3.3}$$

Here we use the negative log-likelihood as the loss function to learn optimal model parameters θ. This is equivalent to maximizing the log-likelihood $\mathscr{L}$ of the data set D as given below:

$$\mathscr{L}(\theta = \{W, \boldsymbol{b}\}, D) = \sum_{i=1}^{|D|} \log\left(P(Y = y|x, W, \boldsymbol{b})\right) \tag{3.4}$$

We then compute the class-membership probabilities for each sample y_{pred} as given in the below equation:

$$y_{pred} = \text{argmax}_y P(Y = y|x, W, \boldsymbol{b}) \tag{3.5}$$

4. **Random Initialization:** To reduce the variance of the back-propagated gradient, Glorot and Bengio (Glorot et al., 2011) proposed to randomly initialize the weights of the neurons in the k-th layer of the network using the uniform distribution as given in eq. (3.6)

$$\boldsymbol{w}^k = \mathscr{U}\left[\frac{-\sqrt{6}}{\sqrt{M_k + M_{k+1}}}, \frac{\sqrt{6}}{\sqrt{M_k + M_{k+1}}}\right] \tag{3.6}$$

5. **Stochastic Gradient Descent:** Gradient Descent is a process that optimizes the Neural Network loss function. At each iteration (also known as epoch) the gradient of the loss function is computed to perform a parameter update. In large-scale applications, the training data can have millions of examples, hence, it seems wasteful to compute the full loss function over the entire training set in order to perform only a single parameter update. A very common approach to address this challenge is to compute the gradient over batches of training data. Therefore a much faster convergence can be achieved in practice by evaluating mini-batch gradients to perform more frequent parameter updates. The extreme case of this is a setting where the mini-batch contains only a single example. This process is called Stochastic Gradient Descent (SGD) (or also sometimes on-line gradient descent).

6. **Baseline (BL):** Given a target dataset D_T, a BL classifier is any function $f(\boldsymbol{w})$ that is trained from a random combination of instances from $\boldsymbol{x}_i \in X_T$ to solve for target task Y_T.

7. **Validation sets for Hyperparameter tuning:** From the training set we extract a subset called validation set that is used not for training but as a "fake" test set to tune the model's hyperparameters.

3.1.1 Stacked Denoising Autoencoder (SDA)

A Stacked Denoising Autoencoder is a type deep neural network where greedy-layerwise training is done by stacking one denoising autoencoder on top of other. SDA training (Bengio, 2009, Section 6.2) comprises of two stages: an unsupervised pre-training stage followed by a supervised fine-tuning stage. In the unsupervised pre-training stage, the X_N is used alone without their corresponding label set. The pre-training of the first hidden layer denoted as $L1$ (k-th layer = 1) is performed by considering it as a regular dA as shown in Fig.3.1a. Its features $\boldsymbol{w}^1$ are trained for several epochs until the cost function hopefully reaching a global minimum. After the first layer is completely pre-trained, we keep only the encoding features $\boldsymbol{w}^1$ of the dA and discard the decoding features. Then we begin pre-training the second hidden layer $L2$ in a similar way, except that in this case, we reconstruct the output values of the 1st hidden layer instead of the input data. Then we repeat the pre-training until the $K-1^{\text{th}}$ hidden layer is completely pre-trained to obtain $\boldsymbol{w}^K - 1$

as shown in Fig.3.1b. We represent each layer training of this multi-layered network as a function $U(W)$.

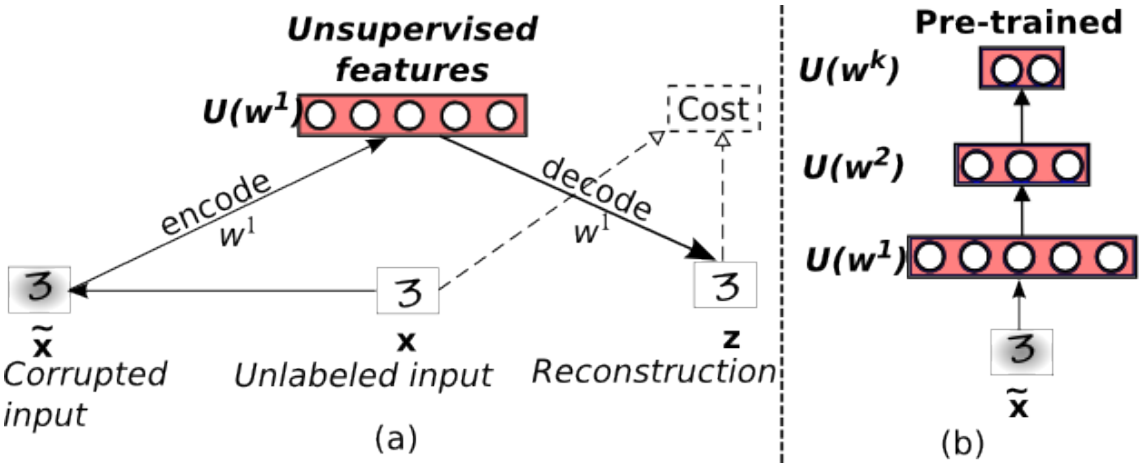


Figure 3.1: (a) Pre-training the first layer feature set, (b) Pre-trained $K-1$ layers

In the supervised fine-tuning stage, a logistic regression layer with c neurons is added to the top of the pre-trained machine, where c is the number of labels in D. Then, the entire classifier is trained (fine-tuned) using both X_N and Y_N in order to minimize a cross-entropy loss function measuring the error between the classifier's predictions and the correct labels. We represent this supervised fine-tuning process as a function $S(W, \boldsymbol{c})$.

3.1.2 Convolutional Neural Network (CNN)

We use three main types of layers to build CNN architectures: Convolutional Layer (conv), Pooling Layer (pool), and Fully-Connected Layer (FC). We will stack these layers to form a full CNN architecture with a logistic regression classifier (LR). CNN architecture is better explained in three stages. The image processing stage, the alternating convolutional and subsampling stage and finally the classification stage (LeCun et al., 1998). The image processing stage is a pre-processing stage of predefined filters that are kept fixed during training. The convolutional and subsampling are architectural ideas to ensure some degree of shift and distortion invariance. The convolution layer convolutes the input with a set of filters like Gabor filters or trained filters producing feature maps (Simple cells). These feature maps are further reduced by subsampling (Complex cells). Finally, feature or kernel size of convolution filters and subsampling are chosen such that the output maps of the last convolutional layer are downsampled to 1 pixel per map or fully connected layer and fed to classification stage. The depth of the CNN is a function of the number of alternating convolutional and subsampling stage.

Another important concept of CNNs is that of max-pooling. Max-pooling (Scherer et al., 2010) is a form of non-linear downsampling and can be used instead of subsampling layers. Max-pooling partitions the input image into a set of non-overlapping rectangles and, for each such

sub-region, outputs the maximum value. The max-pooling can lead to faster convergence, select superior invariant features and improve generalization.

3.1.3 Baseline for Stacked Denoising Autoencoders

SDA are multiple layer networks where each one is trained as a denoising autoencoder (dA). SDA training comprises of two stages: an unsupervised pre-training stage followed by a supervised fine-tuning stage. During pre-training (PT), the network is generated by stacking multiple dA one on top of each other thus learning *unsupervised features*, represented as a vector $U(\boldsymbol{w}^1,\ldots,\boldsymbol{w}^{K-1})$ of optimal weights and biases and simplified notation as $U(W)$. Then, a logistic regression layer is added on top and the whole network and fine-tuned $S(U(W))$in a supervised way, thus learning *supervised features* as discussed in Section 3.1.1.

Algorithm 1 Baseline approach for either SDA or CNN.

Given a training D_{train}, validation D_{valid} and test set D_{test},

1. Randomly initialize a classifier network;
2. IF: “Model == Convolution Neural Network";

   Train (Fine-tune) the network using D_{train} and D_{valid};

   ELSE IF: “Model == Stacked Denoising Autoencoder";

   Pre-train the network using D_{train} and D_{valid};
   Fine-tune the network using D_{train} and D_{valid};
3. Test the network using D_{test}, obtaining baseline classification error ε.

3.1.4 Baseline for Convolutional Neural Network

The training of CNN consists of alternating convolutional and max pooling layers. The posterior probabilities are then calculated for the fully connected layer. The CNN network is trained with random initialization. The BL approach for both SDA and CNN are elaborated in Algorithm 1.

3.2 Problem Formulation for Deep Transfer Learning

The study of transfer learning was inspired by the ability of humans to reuse prior experience under different environments. Naturally, the transfer learning paradigm implies reusing learning machines previously trained for a given source problem S in order to solve, with minor modifications, a different target problem T. An ideal transfer learning method should improve the reused classifier over the one trained from scratch called baseline.

Given an dataset $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$ drawn from input space X and a set of labels Y, a classifier is any function $f(\boldsymbol{x}) : X \to Y$ that maps instances $\boldsymbol{x}_i \in X$ to labels. The classifier performance,

like error rate ε to predict and computation time, is measured on a test set X_{test} with v unlabeled instances drawn from the same distribution $P(X)$.

Traditionally, the goal of transfer learning is to transfer the learning (knowledge) from a source-problem input space X_S to one or more problems, or distributions, to efficiently develop an effective hypothesis for a new task, problem, or distribution (Bruzzone and Marconcini, 2010). In this framework of transfer learning, the source and target problems may come from equal or different distributions. In supervised learning, the source Y_S and target Y_T labels may be equal or different. Four possible cases of transfer learning problems can be identified:

1. The distributions are equal $P_S(X) = P_T(X)$ and the labels are also equal $Y_S = Y_T$.
2. The distributions are equal $P_S(X) = P_T(X)$ and the labels are not equal $Y_S \neq Y_T$.
3. The distributions are different $P_S(X) \neq P_T(X)$ and the labels are equal $Y_S = Y_T$.
4. The distributions are different $P_S(X) \neq P_T(X)$ and the labels are not equal $Y_S \neq Y_T$.

Under such hypothesis, our goal is to obtain an accurate classification for target-domain instances by exploiting labeled training instances from the source-domain.

Throughout the thesis when formalizing a theory we represent different dataset from the same classification problem as D_A, D_B or D_C. In case of dataset from real world applications we use P_A, P_B or P_C. To emphasize the distinction that real world dataset P_A is an estimate of true domain X from which the i.i.d samples are drawn. The source dataset is represented as D_S and the target dataset as D_T. The usage of this representation will be clear through the context.

Combined Baseline (cBL): Given a source dataset D_S and a target dataset D_T, a cBL classifier is any function $f(\boldsymbol{w})$ that is trained from a random combination of instances from both $\boldsymbol{x}_i \in X_S$ and $\boldsymbol{x}_i \in X_T$ to solve for target task Y_T.

Transfer learning (TL): We first train the source network with the source data X_S and Y_S and then copy its hidden layers to the target network. In case $Y_S \neq Y_T$, we add a classifier layer randomly initialized. The network is trained towards the target task Y_T. If the performance of the newly trained target network exceeds the performance of the baseline approach we have positive transference; otherwise we have negative transference.

Transferred layers: We may select a particular layer or set of layers of the whole source network to transfer. For example we may select to transfer only the first layer features to the target network. The rest of the target network layer features are randomly initialized.

3.2.1 Comparing distributions

Traditionally, the Kullback-Leibler (KL) divergence has been used to estimate distribution differences between two datasets (Lin, 1991). Given two probability functions $p(x)$ and $q(x)$, KL divergence is defined as:

$$\mathscr{D}_{KL}(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)} \tag{3.7}$$

Besides the theoretical and practical limitations of this measure (undefined when $q(x) = 0$) and having no upper bound, one drawback of this measure is that it cannot be defined as a distance metric since it does not obey the symmetry and the triangular inequality properties. An alternative to this is the well known Jensen-Shannon (JS) divergence (Lin, 1991) given by:

$$\mathscr{D}_{JS}(p||q) = \alpha\mathscr{D}_{KL}(p||r) + \beta\mathscr{D}_{KL}(q||r), \text{ with } r = \alpha p + \beta q, \tag{3.8}$$

where $\mathscr{D}_{KL}$ is the Kullback-Leibler divergence as defined in eq. (3.7).

When $\alpha = \beta = 1/2$ in eq.3.8 we are dealing with the *specific* Jensen-Shannon divergence and $\mathscr{D}_{JS}$ is lower- and upper-bounded by 0 and 1, respectively, when using logarithm base 2 (Lin, 1991). This means that when $\mathscr{D}_{JS}(p||q) = 0$ we can consider that p and q are identical and when $\mathscr{D}_{JS}(p||q) = 1$, the distributions are different. We use Jensen-Shannon divergence as a measure to compute the difference between the distributions of two datasets.

We measure the improvement using the transferred features over random initialization using the relative improvement, i_r, to the baseline methodology as follows:

$$i_r = \frac{\varepsilon_{BL} - \varepsilon_{method}}{\varepsilon_{BL}} \tag{3.9}$$

where ε_{method} represents the average error rate observed for a given methodology.

3.3 Datasets

We evaluate DTL on three types of tasks - character and object recognition - using five original datasets[1]. These datasets are either distinct in the number of labels or distributions. To evaluate all possible TL cases, we modified the five original datasets into fourteen different datasets as listed in Table 3.1. All the datasets are re-sized to 28×28 pixels from the original size.

3.3.1 Character recognition

We evaluated the framework in two different settings for recognizing characters. We used the *MNIST* dataset and called it as *Latin* P_L which had 60,000 training and 10,000 testing instances with labeled hand-written latin digits from 0 to 9. Then we used the *MADbase* dataset and called it as *Arabic* P_{Ar} which had 60,000 training and 10,000 testing instances with labeled hand-written arabic digits from 0 to 9. Additionally, the Chars74k dataset was modified to obtain the *Lowercase* dataset P_{LC} with lowercase letters from a-to-z, the *Uppercase* dataset P_{UC} with uppercase letters from A-to-Z, and the *Digits* P_D dataset with digits from 0-to-9. The Latin-2 dataset is a modified version of MNIST to match the number of training and validation instances of the Lowercase dataset.

[1]Here we would like to acknowledge the following research centers for making available their datasets: Center for Neural Science, New York University for MNIST; Microsoft Research India for Chars74k; Electronics Engineering Dept., The American University in Cairo for MADbase; LISA labs, University of Montreal, Canada for BabyAI shapes.

Table 3.1: Number of instances available for each dataset.

Data set		**Labels**			**Instances**		
		Ω		classes	**Train**	**Valid**	**Test**
Latin	P_L	0-to-9	Ω_{09}	10	50,000	10,000	10,000
Latin-2	P_{L2}	0-to-9	Ω_{09}	10	13,208	6,604	6,604
Arabic	P_{Ar}	0-to-9	Ω_{09}	10	50,000	10,000	10,000
Chars74k	P_{Ch}	a-to-z, A-to-Z, 0-to-9	Ω_{aZ09}	62	39,624	19,812	19,812
Lowercase	P_{LC}	a-to-z	Ω_{az}	26	13,208	6,604	6,604
Uppercase	P_{UC}	A-to-Z	Ω_{AZ}	26	13,208	6,604	6,604
Digits	P_D	0-to-9	Ω_{09}	10	13,208	6,604	6,604
Canonical	P_{Sh1}	eqt,cir,sqr	Ω_{sh1}	3	14,000	1,000	5,000
Non-Canonical	P_{Sh2}	tri,ell,rec	Ω_{sh2}	3	14,000	1,000	5,000
Curve & corner	P_{Sh3}	rou,cor	Ω_{sh3}	2	14,000	1,000	5,000
Shape1	P_A	eqt,cir,sqr	Ω_A	3	10,000	5,000	5,000
Shape2	P_B	tri,ell,rec	Ω_B	3	10,000	5,000	5,000

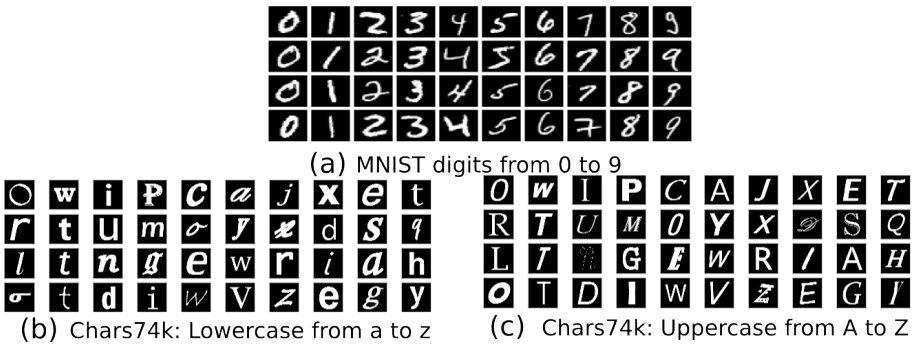


Figure 3.2: Samples from character recognition tasks

3.3.2 Object recognition

We generated two sets of object recognition problems based on the level of complexity of the classification tasks. The *basic object recognition problems* contains three datasets: canonical, non-canonical and curve Vs. corner as shown in Fig 3.3. The (more) *complex object recognition problems* contains two datasets: Shape1 and Shape2.

The first set of basic object recognition problems are as follows:

- the *canonical* dataset P_{Sh1} is composed of images of equilateral triangle, circle and square;
- the *non-canonical* dataset P_{Sh2} is composed of images of triangle, ellipse and rectangle;
- the *curve Vs. corner* dataset P_{Sh3} is composed of images of curved or cornered surfaces.

The second set has more complex object recognition problems made of variations from the basic set; such as variation of colors (ranging from of 0 to 7), variation in position of objects inside the image frame (from left extreme to right extreme) and variations in the angle of the object alignment (from 0 to 360^o), which are categorized into two tasks listed as follows:

- *Shape1* having canonical patterns namely images of equilateral triangles, circles and squares;
- *Shape2* having more complex non-canonical patterns, namely images of ellipsis, rectangles and triangles;

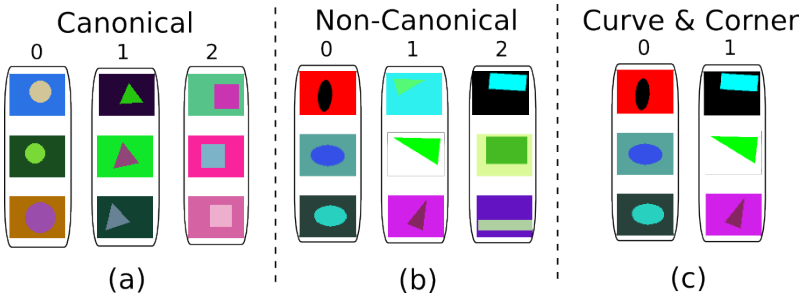


Figure 3.3: Samples from various shape recognition tasks

3.4 Conclusion

The fundamental concepts for Deep Transfer Learning framework have been studied based on the current state-of-the-art methods. We have various object recognition and character recognition problems to study the mechanism that will be developed for the DTL framework. We combine the advantage of the deep models and the transfer learning model for the three mechanisms developed. In Chapter 4, we discuss the layerwise transfer learning, where we specifically study the unsupervised versus supervised training of the models. In Chapter 5, we investigate by continuously switching source and target dataset such that the solution space reaches to common minima. In Chapter 6, we implement an ensemble method for various layerwise transfer learning mechanisms.

Chapter 4

Layerwise Transfer Learning

We propose a layerwise transfer learning approach[1], which enables deep neural networks to transfer features of hidden layer(s) for a classifier trained in either an unsupervised or a supervised way. Our approach is inspired by the 1959 biological model proposed by Nobel laureates David H. Hubel and Torsten Wiesel, who found two types of cells in the visual primary cortex: simple cells and complex cells. The visual cortex is the part of the brain that is responsible for processing the visual information. Deep architectures try to mimic the human primary visual cortex (see (Vincent et al., 2010), (LeCun et al., 1998), (Hinton et al., 2006) and (Bengio, 2009, Section 11.3)).

4.1 Layerwise Transfer Learning mechanism

We analyze two types of DTL approaches for Layerwise Transfer Learning mechanism: 1) Transfer Learning unsupervised (TLu), and 2) Transfer Learning supervised (TLs). We train a classifier on a harder problem and then reuse it on a simpler problem with a completely *different task* drawn from a different distribution. For example we pick the features of a machine built to classify images of digits from 0-to-9 and reuse them to classify images of letters from a-to-z. Similar experiments are conducted by reversing the role played by each problem (simpler to harder). In addition, we also explore transfer learning between *same task problems* drawn from different distributions of geometrical shapes. Processing large data as we did, on millions of neural connections, would take several weeks using traditional CPUs. Instead, we used a GPU for faster processing of these large networks and to allow repetitions of each experiment several times for statistical significance.

A deep model is trained on a source problem, and its features are transferred to help in solving a target problem. We represent this feature transference by $\boldsymbol{w}_S^k \Rightarrow \boldsymbol{w}_T^k$ for k-th layer. Therefore we explore feature transference in deep models either at the pre-training stage $U(W)$, coined Transfer Learning unsupervised (TLu), or at the fine-tuning stage $S(W)$, coined Transfer Learning supervised (TLs).

[1] Some parts of this chapter are used in (Kandaswamy et al., 2014a) and (Kandaswamy et al., 2014b)

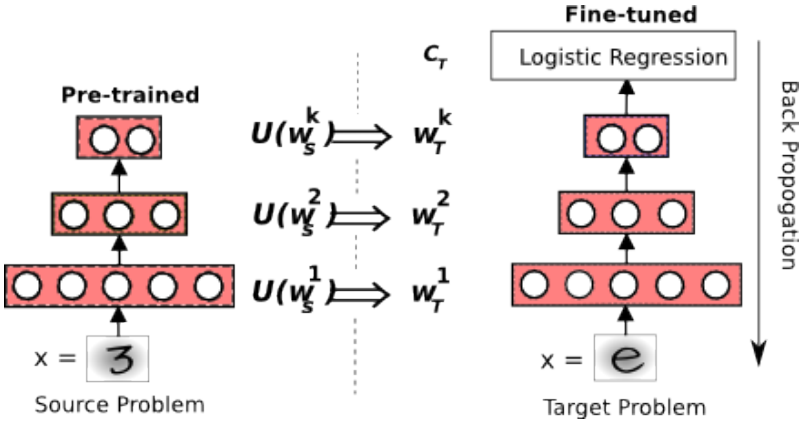


Figure 4.1: Transfer learning unsupervised (TLu)

4.1.1 Transfer Learning unsupervised (TLu)

In the TLu approach the unsupervised features are transferred from the source to the target network as depicted in Fig 4.1, TLu. As a first step we randomly initialize each layer of the network using uniform distribution (Glorot et al., 2011):

$$\boldsymbol{w}^k = \mathscr{U}\left[\frac{-\sqrt{6}}{\sqrt{M_k + M_{k+1}}}, \frac{\sqrt{6}}{\sqrt{M_k + M_{k+1}}}\right] \tag{4.1}$$

In second step we apply greedy layerwise pretraining to the network until $K-1^{\text{th}}$ hidden layer $U(\boldsymbol{w}_S^1, \ldots, \boldsymbol{w}_S^{K-1})$ using source data unlabelled samples as explained in Section 3.1.1.

$$U(\boldsymbol{w}_S^1, \ldots, \boldsymbol{w}_S^{K-1}) \tag{4.2}$$

In the third step we apply feature transference by mapping the pretrained source network features to the target problem features as shown in eq. (4.3) and as depicted in Fig.4.1.

$$U(\boldsymbol{w}_S^1, \ldots, \boldsymbol{w}_S^{K-1}) \Rightarrow \{\boldsymbol{w}_T^1, \ldots, \boldsymbol{w}_T^{K-1}\} \tag{4.3}$$

Once the features are transferred to the target network we apply fine-tuning as a regular deep network as explained in Section 3.1.1. We add a logistic regression layer with c_T number of neurons, where c_T is number of classes in the target dataset. The K-th layer weight matrix connects the networks $K-1$-th hidden layer to the logistic regression layer and the matrix dimension is given by $M_{K-1} \times c_T$.

$$\boldsymbol{w}_T^K = \boldsymbol{w}_{\{M_{K-1} \times c_T\}}^K \tag{4.4}$$

Finally, we fine-tune this entire deep network as a multi-layer perceptron using back-propagation as given in eq.(4.5). The TLu approach is listed as approach number 9 in Table 4.1 and pseudocode is given in Algorithm 2.

$$S(W) = S(U(\boldsymbol{w}_T^1, \ldots, \boldsymbol{w}_T^{K-1}), \boldsymbol{w}_T^K) \tag{4.5}$$

4.1.2 Transfer Learning supervised (TLs)

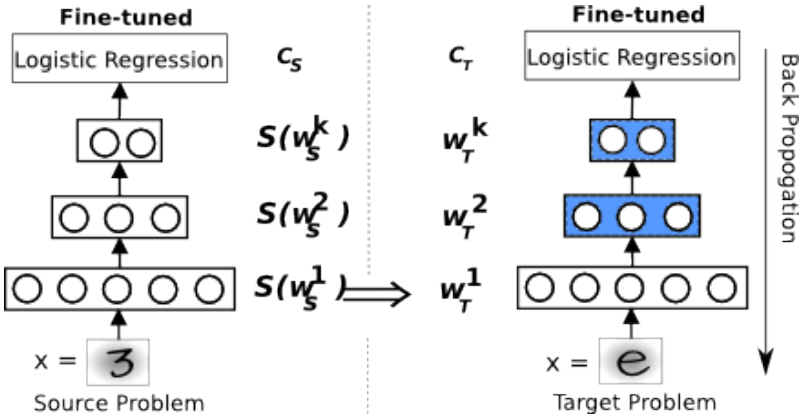


Figure 4.2: TLs for "L1" feature transference approach.

Supervised features are transferred from the source to the target network as illustrated in Fig 4.2. The TLs approach is similar to TLu approach. Here in TLs approach we transfer fine-tuned source problem features instead of pre-trained features.

First, we randomly initialized source feature set as in eq.(4.1) and then pre-train the network $U(\boldsymbol{w}_S^1,\ldots,\boldsymbol{w}_S^{K-1})$ until the $K-1^{\text{th}}$ hidden layer as in eq.(4.2). After pre-training, we fine-tuned these unsupervised features with the source problem labeled instances as given in eq.(4.6), where c_S is number of source problem classes.

$$S(U(\boldsymbol{w}_S^1,\ldots,\boldsymbol{w}_S^{K-1}),\boldsymbol{w}_{S.\{M_{K-1}\times c_S\}}^K) \tag{4.6}$$

Then we apply feature transference by mapping the fine-tuned source network features to the target problem features based on transfer setting $Y_S = Y_T$ or $Y_S \neq Y_T$.

In the case of $Y_S \neq Y_T$ transfer setting could not reuse the logistic regression layer, as the label set for the source problem Ω_S with c_S labels is not equal to the target problem label set Ω_T with c_T labels. Thus the logistic regression layer was randomly initialized for the target problem. Then we apply feature transference by mapping the fine-tuned source network features to the target problem features as shown in eq.(4.7):

$$S(U(\boldsymbol{w}_S^1,\ldots,\boldsymbol{w}_S^{K-1})) \Rightarrow \{\boldsymbol{w}_T^1,\ldots,\boldsymbol{w}_T^{K-1}\} \tag{4.7}$$

Once the features are transferred to the target network we apply fine-tuning as a regular deep network as explained in Section 3.1.1. We remove the previous logistic regression layer if $Y_S \neq Y_T$ and add a new logistic regression layer. Finally, we fine-tune this entire deep network as a multi-layer perceptron using back-propagation as given in eq.(4.8). This is listed as "FT" approach in Table 4.1

$$S(W_T) = S(S(U(\boldsymbol{w}_T^1,\ldots,\boldsymbol{w}_T^{K-1}),\boldsymbol{w}_T^K)) \tag{4.8}$$

Table 4.1: Lists TLs, TLu Transfer Learning and Baseline Approach. An illustration of TLs with all possible combinations for a 3 hidden layer network.

No.	Approaches	Feature Transference	$S(W_T) =$
1.	FT	$S(U(\boldsymbol{w}_S^1,\ldots,\boldsymbol{w}_S^{K-1})) \Rightarrow \{\boldsymbol{w}_T^1,\ldots,\boldsymbol{w}_T^{K-1}\}$	$S(S(U(W_T)))$
2.	L1+L2+L3	$S(U(\boldsymbol{w}_S^1,\boldsymbol{w}_S^2,\boldsymbol{w}_S^3)) \Rightarrow \boldsymbol{w}_T^1,\boldsymbol{w}_T^2,\boldsymbol{w}_T^3$	$S(S(U(W_T)))$
3.	L1+L3	$S(U(\boldsymbol{w}_S^1,\boldsymbol{w}_S^3)) \Rightarrow \boldsymbol{w}_T^1,\boldsymbol{w}_T^3$	$S(S(U(W_T)))$
4.	L2+L3	$S(U(\boldsymbol{w}_S^2,\boldsymbol{w}_S^3)) \Rightarrow \boldsymbol{w}_T^2,\boldsymbol{w}_T^3$	$S(S(U(W_T)))$
5.	L1+L2	$S(U(\boldsymbol{w}_S^1,\boldsymbol{w}_S^2)) \Rightarrow \boldsymbol{w}_T^1,\boldsymbol{w}_T^2$	$S(S(U(W_T)))$
6.	L3	$S(U(\boldsymbol{w}_S^3)) \Rightarrow \boldsymbol{w}_T^3$	$S(S(U(W_T)))$
7.	L2	$S(U(\boldsymbol{w}_S^2)) \Rightarrow \boldsymbol{w}_T^2$	$S(S(U(W_T)))$
8.	L1	$S(U(\boldsymbol{w}_S^1)) \Rightarrow \boldsymbol{w}_T^1$	$S(S(U(W_T)))$
9.	TLu	$U(\boldsymbol{w}_S^1,\ldots,\boldsymbol{w}_S^{K-1}) \Rightarrow \{\boldsymbol{w}_T^1,\ldots,\boldsymbol{w}_T^{K-1}\}$	$S(U(\boldsymbol{w}_T^1,\ldots,\boldsymbol{w}_T^{K-1}),\boldsymbol{w}_T^K)$
10.	Baseline	-	$S(U(\boldsymbol{w}_T^1,\ldots,\boldsymbol{w}_T^{K-1}),\boldsymbol{w}_T^K)$

In case of $Y_S = Y_T$ we retrain logistic regression layer as shown in eq.(4.9) and its simplified form is $S(S(U(W_T)))$:

$$S(U(\boldsymbol{w}_S^1,\ldots,\boldsymbol{w}_S^{K-1}),\boldsymbol{w}_S^K) \Rightarrow \{W\} \tag{4.9}$$

The TLs has 1 to 8 approaches as listed in Table 4.1 and pseudocode is given in Algorithm 2. For example in approach number 8 listed as "L1" approach in Table 4.1, we transfer the first layer, that is, $S(U(\boldsymbol{w}_S^1)) \Rightarrow \boldsymbol{w}_T^1$ as illustrated in Fig.4.2 (L1 stands for Layer 1). Then we fine-tuned for a second time the entire classifier like a regular multi-layer perceptron with back-propagation using both design and label sets of the target problem.

Similarly, we can transfer the first and second layer features, that is, $S(U(\boldsymbol{w}_S^1,\boldsymbol{w}_S^2)) \Rightarrow \boldsymbol{w}_T^1,\boldsymbol{w}_T^2$, listed as the "L1+L2" approach in Table 4.1. It was interesting to see that this opened up various new combinations of supervised features to reuse for the target problem.

4.2 Layerwise Transfer Learning for SDA and CNN

Usually, in the feature transference, we transfer the features of a model from the source to the target problem. In the Layerwise Transfer Learning (LTL) approach, we transfer either supervised or unsupervised or low-level layer or high-level layer or combination of these features using LTL as shown in Table 4.1 and pseudocode for baseline and LTL is shown in Algorithm 2.

Generic features or specific feature: The bottom-layer/ low-level features are referred to as *generic* and the top-layer/ high-level features as *specific*. The in-between layers between the bottom and top layer are the middle-level features. For example in a three hidden layer deep network, bottom or L1 layer features are termed as generic features, middle or L2 layer features are termed as middle-level features and top or L3 layer features are termed as specific features. The

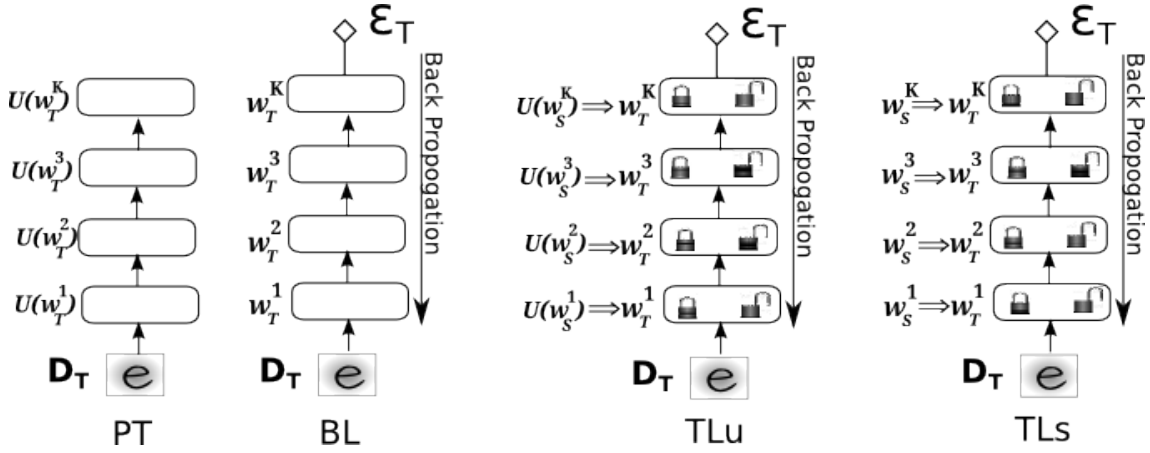


Figure 4.3: A pictorial representation of approaches: Pre-training (PT), Baseline (BL), Transfer Learning unsupervised (TLu) and Transfer Learning supervised (TLs) with the option of *lock* or *unlock* for each layer

choice of whether or not to fine-tune the L1 layer of the target network depends on the number of samples in the target dataset and also on network architecture (Kandaswamy et al., 2014b) and (Razavian et al., 2014).

Locking or unlocking layers: Once the features are transferred to the target network, we add a logistic regression layer for the target task Y_T. We have the choice to fine-tune this entire network W as a multi-layer perceptron using backpropagation or *lock* a given layer (Yosinski et al., 2014), (Kandaswamy et al., 2015b), meaning that the transferred feature from the source network $\boldsymbol{w}_S^k \Rightarrow \boldsymbol{w}_T^k$ do not change during the error propagation for the target task. This gives the choice of whether or not to fine-tune certain layers of the target network. Locking a layer dictates the splitting of inter-layer connections (co-adapted neurons) which leads to difficulty in optimization (Yosinski et al., 2014).

For example, locking a middle-level (L2) layer in a three hidden layer deep net, breaks the error backpropogation from the top (L3) to middle (L2) layer. The network weights between L2 and L3 has a matrix dimension is given by $M_2 \times M_3$ is as given in eq.(4.10).

$$\boldsymbol{w}_S^3 = \boldsymbol{w}_{\{M_2 \times M_3\}}^3 \tag{4.10}$$

The weight matrix was optimized when trained with the source problem dataset also influences the co-adaptation of neurons between L2 and L3. Locking the flow of gradient decent from L3 to L2 during the backpropogation of error breaks the fragile co-adaptation of neurons between the layers, this forces the network to shake the solution space from local minima to a new solution space adaptable to the target problem promoting domain generalization.

The TLu and TLs leads to several possible approaches to solve a problem as shown in Fig. 4.3: 1) we can choose particular layer(s) of the network to be locked or unlocked, 2) we can choose to transfer either generic, middle-level or specific features to the target network, 3) we can choose to transfer either supervised or unsupervised features to the target network

The LTL approach enables the flexibility to adapt the mechanism based on the application.

Table 4.2: Average classification test error in percentage ($\overline{\varepsilon}$) obtained with the baseline approach along with the corresponding average training times (seconds) with GTX 770.

Data set Distribution		Labels Ω		c	$\overline{\varepsilon}$	Avg. Training Time (s) Total	Pre-train(%)	Fine-tune(%)
Latin	P_L	0-to-9	Ω_{09}	10	1.61±0.19	10698	40.0	60.0
Arabic	P_{Ar}	•-to-9	$\Omega_{\bullet 9}$	10	1.37±0.07	8051	20.7	79.3
Latin-2	P_{L2}	0-to-9	Ω_{09}	10	2.92±0.10	2347	28.1	71.9
Digits	P_D	0-to-9	Ω_{09}	10	1.88±0.14	1010	34.6	65.4
Lowercase	P_{LC}	a-to-z	Ω_{az}	26	4.95±0.16	2997	43.6	56.4
Uppercase	P_{UC}	A-to-Z	Ω_{AZ}	26	5.01±0.27	2567	34.7	65.3
Shape1	P_A	'eqt','cir','sqr'	Ω_{sh1}	3	7.88±0.93	3564	54.9	45.1
Shape2	P_B	'tri','ell','rec'	Ω_{sh2}	3	15.51±6.31	4095	60.5	39.5

For example, if a application needs supervised features from generic layer, we choose the $L1$ approach from Table 4.1. We transfer only layer one supervised features from source to target network, $S(U(\boldsymbol{w}_S^1)) \Rightarrow \boldsymbol{w}_T^1$ and then randomly initialize and fine-tune other layer feature set of the target network, $S(S(U(W_T)))$. Similarly if we need both low-level and middle-level features to transfer both we choose to transfer both first and second layer parameters i.e.,we choose the $L1+L2$ approach $S(U(\boldsymbol{w}_S^1, \boldsymbol{w}_S^2)) \Rightarrow \boldsymbol{w}_T^1, \boldsymbol{w}_T^2$.

We use following simplified notations to ease the readability of the text. We denote deep network architecture as [L1,...,LK] with $K-1$ hidden layers and the K-th layer is the logistic regression layer (LR). We represent feature transference as '1' and not to transfer as '0' in the *transfer* network architecture, similarly '1' represent unlocked layer and '0' represent locked layer in the *retrain* network architecture. For example to denote network architecture for a three hidden layer network as [L1,L2,L3,LR]. Here we transfer all the hidden layers only and do not transfer LR layer of the source network to the target network, we denote the transfer network architecture as [1 1 1 0]. To retain only the top and LR layer we denote the retrain network architecture as [0 0 1 1].

4.2.1 Network Architecture

Tuning hyper-parameters such as learning rate or setting the appropriate network architecture for training the deep network is desirable but it is highly time consuming. For SDA model, we used pre-training and fine-tuning learning rates of 0.001 and 0.1, respectively, taken from our previously tuned models. The stopping criteria for pre-training was fixed to 40 epochs; stopping criteria for fine-tuning was set to a maximum of 1000 epochs with the validation dataset. Each of these experiments is repeated 10 times and Student t-test is performed to give some statistical significance.

We selected the network architecture inspired from the Convolutional Neural Network's pyramidal structure for classification of visual patterns (LeCun et al., 1998). This enabled us to exploit the *geometrical properties* of images. Given the number of inputs as $28 \times 28 = 784 = 16 \times 7^2$ pixels, the number of neurons at each hidden layer is selected as a decreasing geometrical progression. Thus, the number of neurons in the k^{th} layer is given by $M^k = 16\,(7-k)^2$. We represent the deep

Algorithm 2 Pseudocode for baseline and Transfer learning approach

Baseline: Initialize randomly:
Given a two datasets D_A and D_B, Select a dataset and train the network with input x
{**Stage 1:** Pretrain the Network}
build SDA by greedy layer
for K in number of hidden layers **do**
  randomly initialize: W
  {Build denoising autoencoder (dA)}
  for each epoch in Pretraining **do**
    Corrupt the input, $x = x + noise$
    $hidden\,layer = Sigmoid(w^k x + bias)$
    $reconstruct = Sigmoid((w^k)^T x + bias')$
    minimize cross-entropy loss and update weight vector
  end for
  stack the dA's
end for
{**Stage 2:** Fine-tune the Network}
add a logistic regression layer with Y labels
for each epoch in Fine-tuining **do**
  backpropogate the errors
  update the weights
  calculate validation error on validation set
  if best validation error $<$ validation error **then**
    update weights of the network
    best validation error = validation error
    calculate test error on test set
    best test error = current error
  end if
end for
error = best test error

Initialize with trained features D_A:
Given a two datasets D_A and D_B, with tasks Y_A and Y_B,
Select D_A dataset and train the network A as described on the left side.
{**Stage 1:** Transfer the features}
Select a reuse mode: TLu or and TLs
Select which hidden layers to transfer
if $Y_A \neq Y_B$ **then**
  chop of the logistic layer
end if
for K in number of layers **do**
  if layer = transfer **then**
    if mode = TLu **then**
      transfer unsupervised features
      $U(w_A^k) \Rightarrow w_B^k$
    else if mode = TLs **then**
      transfer supervised features
      $w_A^k \Rightarrow w_B^k$
    end if
  else if layer = no transfer **then**
    randomly initialize weights w_B^k
  end if
end for
{**Stage 2:** Fine-tune the Network}
if $Y_A = Y_B$ **then**
  add a logistic regression layer with Y_B labels
end if
for each epoch in Fine-tuining **do**
  backpropogate the errors
  if lock is TRUE in each Layer **then**
    no update of weights
  else
    update the weights
  end if
  calculate validation error on validation set
  if best validation error $<$ validation error **then**
    update weights of the network
    best validation error = validation error
    calculate test error on test set
    best test error = current error
  end if
end for
error = best test error

network as [M^1,M^2,..., M^k, c], where c is the number of output labels. In the following experiments the SDA network has three hidden layers and one output layer, or [16×6^2, 16×5^2, 16×4^2, c] amounting to 784,384 connections. Moreover, the induced random corruption levels for each of the three hidden layers inputs are [10%, 20%, 30%], respectively.

We used Theano (Bergstra et al., 2010), a GPU compatible machine learning library to perform all our experiments on a i7-377(3.50GHz), 16GB RAM and GTX 770 GPU processor. Table 4.2 presents average test error rates of the Baseline SDA for each dataset along with the computation time in seconds for the above defined network architecture. The GPU parallel processing allows training both CNN's and SDA's deep neural networks with millions of neural connection, for very small learning rate, for large number of epochs, for very large datasets within several days. Each of these experiments are repeated 10 times to increase the confidence level of the results. The hyperparameters for CNN used kernel filter size of [20, 50] and max training epochs of 200. The learning rate of 0.1 is set with batch training of 100.

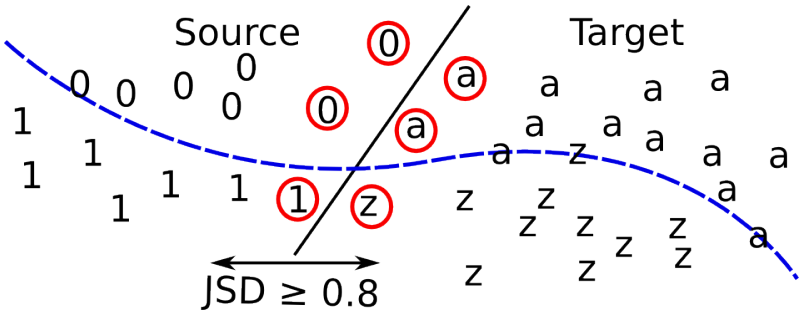


Figure 4.4: A pictorial representation of labels are different TL setting $Y_S \neq Y_T$ and $P_S(X) \neq P_T(X)$ as the Jensen-Shannon divergence (JSD) between the source and the target distribution is greater than 0.8

4.3 Results and Discussions

4.3.1 Problem Categorization

If a problem has higher classification error than another problem we categorize it as a *harder* problem. Moreover, if the source problem is harder than the target problem we categorize the transfer learning setting as *Hard Transfer* (HT). The reverse case, that is, when the roles of such source and target problems are interchanged the transfer learning setting is categorized as *Reverse Transfer* (RT). In the experiments we are interested solely in the case of different distributions of the source and target datasets, $P_S(X) \neq P_T(X)$.

4.3.2 TLu: Different label sets

In this section we study feature transference behavior of a machine trained on a harder problem with our TLu approach using SDA model. For that purpose we have carried out experiments by training a machine to classify images of handwritten digits (harder problem than synthetic digits) and reusing unsupervised features to classify images of synthetic letters, as shown in Fig. 4.4. We also performed experiments by reversing the problem roles: training a machine with simpler problems like synthetic letters and reusing the features to classify harder problems like handwritten digits. In both these studies, the label set 'digits' and the label set 'letters' are different, $Y_S \neq Y_T$.

4.3.2.1 Classifying letters reusing digits: HT

The goal is to classify images of synthetic letters by reusing unsupervised features of a machine trained on a harder problem like handwritten digits from 0-to-9.

The performance of classifying letters reusing a machine pre-trained with digits is listed in Table 4.3. The average error rate of recognizing uppercase letters, 4.31±0.61% by reusing a machine pre-trained with Latin digits is significantly *lower* than baseline, 5.01±0.27%. Similar results are obtained from recognizing the lowercase letters. In both cases the significance level (5%) allows rejecting the null hypothesis of equal error rates.

Table 4.3: Changing the set of labels $Y_S \neq Y_T$, $Y_S = Y_T$ for arbitrary distributions $P_S(X) \neq P_T(X)$. Average classification test error (%) ($\overline{\varepsilon}$) obtained for a target problem using TLu approach for different combinations of: target data distribution (P_T); target label set (Ω_T); source distribution (P_S); source label set (Ω_S) for Hard and Reverse Transfer problems using SDA; The difference between distributions is given by Kullback-Leibler (KL) and Jensen-Shannon (JS) divergence.

	Hard Transfer: Harder ⇒ Simpler							Reverse Transfer: Simpler ⇒ Harder						
		P_S	Ω_S	P_T	Ω_T	$\overline{\varepsilon}$	KL	JS		P_S	Ω_S	P_T	Ω_T	$\overline{\varepsilon}$	KL	JS
$Y_S \neq Y_T$	BL			P_{UC}	Ω_{AZ}	5.01±0.27			BL			P_{L2}	Ω_{09}	**2.92±0.10**		
	TL	P_{L2}	Ω_{09}	P_{UC}	Ω_{AZ}	4.65±0.19 ⇑	49.3	0.99	TL	P_{UC}	Ω_{AZ}	P_{L2}	Ω_{09}	3.34±0.09 ⇓	42.2	0.99
	TL	P_L	Ω_{09}	P_{UC}	Ω_{AZ}	**4.31±0.16** ⇑	32.8	0.79	TL	P_{LC}	Ω_{az}	P_{L2}	Ω_{09}	3.28±0.10 ⇓	42.2	0.99
	TL	P_{Ar}	$\Omega_{\bullet 9}$	P_{UC}	Ω_{AZ}	4.41±0.22 ⇑	48.6	0.99								
									BL			P_L	Ω_{09}	**1.61±0.19**		
	BL			P_{LC}	Ω_{az}	4.95±0.16			TL	P_{UC}	Ω_{AZ}	P_L	Ω_{09}	1.81±0.19 ⇓	4.3	0.79
	TL	P_{L2}	Ω_{09}	P_{LC}	Ω_{az}	4.67±0.38 ⇑	49.3	0.99	TL	P_{LC}	Ω_{az}	P_L	Ω_{09}	1.79±0.22 ⇓	4.5	0.80
	TL	P_L	Ω_{09}	P_{LC}	Ω_{az}	**4.37±0.13** ⇑	32.6	0.80								
	TL	P_{Ar}	$\Omega_{\bullet 9}$	P_{LC}	Ω_{az}	4.43±0.11 ⇑	48.6	0.99	BL			P_{Ar}	$\Omega_{\bullet 9}$	**1.37±0.07**		
									TL	P_{UC}	Ω_{AZ}	P_{Ar}	$\Omega_{\bullet 9}$	1.47±0.08 ⇓	22.8	0.99
									TL	P_{LC}	Ω_{az}	P_{Ar}	$\Omega_{\bullet 9}$	1.49±0.07 ⇓	22.9	0.99
$Y_S = Y_T$	BL			P_D	Ω_{09}	1.88±0.14			BL			P_{L2}	Ω_{09}	**2.92±0.10**		
	TL	P_{L2}	Ω_{09}	P_D	Ω_{09}	1.79±0.12 ○	44.5	0.99	TL	P_D	Ω_{09}	P_{L2}	Ω_{09}	3.27±0.16 ⇓	43.7	0.99
	TL	P_L	Ω_{09}	P_D	Ω_{09}	1.78±0.21 ○	31.9	0.88								
	TL	P_{Ar}	$\Omega_{\bullet 9}$	P_D	Ω_{09}	**1.75±0.21** ○	43.9	0.99	BL			P_L	Ω_{09}	**1.61±0.19**		
									TL	P_D	Ω_{09}	P_L	Ω_{09}	1.84±0.26 ⇓	43.7	0.99
	BL			P_A	Ω_{sh1}	**7.88±0.93**										
	TL	P_B	Ω_{sh2}	P_A	Ω_{sh1}	7.96±0.93 ○	39.4	0.99	BL			P_{Ar}	$\Omega_{\bullet 9}$	**1.37±0.07**		
									TL	P_D	Ω_{09}	P_{Ar}	$\Omega_{\bullet 9}$	1.52±0.07 ⇓	24.4	0.99
									BL			P_B	Ω_{sh2}	15.51±6.31		
									TL	P_A	Ω_{sh1}	P_B	Ω_{sh2}	**13.08±0.58** ○	34.2	0.99

⇑, ⇓, ○ statistically significant improvement or degradation or no change, respectively, than baseline at 5% level. The best $\overline{\varepsilon}$ obtained for a target dataset is marked in bold.

Table 4.4: Average Test Error (%) ($\overline{\varepsilon}$) of TLs approaches for Hard and Reverse Transfer problems using SDA

	Hard Transfer			Reverse Transfer		
Target:	P_{UC}	P_{LC}	P_A	P_{L2}	P_{L2}	P_B
Source:	P_{L2}	P_{L2}	P_B	P_{UC}	P_{LC}	P_A
Labels:	$Y_S \neq Y_T$	$Y_S \neq Y_T$	$Y_S = Y_T$	$Y_S \neq Y_T$	$Y_S \neq Y_T$	$Y_S = Y_T$
JS:	0.99	0.99	0.99	0.99	0.99	0.99
Approaches	$\overline{\varepsilon}$	$\overline{\varepsilon}$	$\overline{\varepsilon}$	$\overline{\varepsilon}$	$\overline{\varepsilon}$	$\overline{\varepsilon}$
FT	**4.58±0.19** ⇑	**4.57±0.08** ⇑	9.13±1.57 ⇓	3.49±0.19 ⇓	3.46±0.18 ⇓	25.52±14.71 ⇓
L1+L2+L3	10.93±0.5 ⇓	10.70±0.3 ⇓	11.10±2.0 ⇓	9.29±0.54 ⇓	8.68±0.39 ⇓	39.81±09.88 ⇓
L1+L3	5.28±0.16 ⇓	5.31±0.18 ⇓	**5.23±1.45** ⇑	4.14±0.24 ⇓	4.14±0.15 ⇓	20.34±16.42 ○
L2+L3	5.41±0.25 ⇓	5.61±0.11 ⇓	9.94±2.54 ⇓	4.40±0.13 ⇓	4.36±0.12 ⇓	26.27±15.47 ⇓
L1+L2	5.60±0.19 ⇓	5.68±0.10 ⇓	6.88±1.89 ⇑	4.22±0.13 ⇓	4.15±0.14 ⇓	22.69±15.43 ○
L3	4.81±0.30 ○	5.17±0.15 ⇓	10.34±0.9 ⇓	3.86±0.11 ⇓	3.82±0.17 ⇓	26.89±13.53 ⇓
L2	4.88±0.17 ○	4.95±0.13 ○	11.14±1.5 ⇓	3.78±0.13 ⇓	3.76±0.14 ⇓	29.71±13.79 ⇓
L1	4.72±0.18 ⇑	4.72±0.17 ⇑	7.29±1.42 ○	3.59±0.14 ⇓	3.59±0.19 ⇓	23.96±15.45 ○
Baseline	5.01±0.27	4.95±0.16	7.88±0.93	**2.92±0.10**	**2.92±0.10**	**15.51±6.31**

⇑, ⇓, ○ statistically significant improvement or degradation or no change than baseline (at 5% level). The best $\overline{\varepsilon}$ obtained for a target dataset are marked in bold.

Transference of unsupervised features of a machine trained on harder source problems like handwritten digits improves the overall performance of simpler target problems. It is interesting to note that the source trained problems are from totally different distributions.

4.3.2.2 Classifying digits reusing letters: RT

The goal is to study the transference behavior by reusing unsupervised features of a machine trained on simpler problem. We simply reversed the roles of the source and target problems as discussed in section 4.3.2.1. Here we consider a problem of classifying handwritten digits from 0-to-9 by reusing unsupervised features of a machine trained on simpler problem like synthetic letters. We observed, that this approach had worst performance than the baseline. The results are listed in Table 4.3 also with similar results in the case of Arabic digits. The study confirms that the degrading performance is due to transference of unsupervised features trained on simpler problems like synthetic letters.

4.3.3 TLu: Equal label sets

We have considered the problem of recognizing digits by reusing unsupervised features trained with digits. Similarly, the problem of recognizing geometrical shapes by reusing unsupervised features trained with canonical shapes is studied. In both cases the label sets are equal, $Y_S = Y_T$.

4.3.3.1 Canonical shapes as a subset of geometrical shapes: HT

Let's consider a classification task to determine geometrical shapes from the Shape2 dataset having triangles, ellipses or rectangles. This task has higher classification error than the Shape1 dataset which is made up of canonical shapes of equilateral triangles, circles or squares. The classification

performance of the TLu approach for Shape1 by reusing Shape2 features is $\overline{\varepsilon}$ = 7.88±0.93%, *lower* than the baseline 7.96±0.93% approach where there is not sufficient evidence to reject the null hypothesis. The results are listed in Table 4.3.

4.3.3.2 Synthetic digits as a subset of handwritten digits: HT

The performance of recognizing synthetic digits by reusing unsupervised features trained with either Latin-2, Latin or Arabic handwritten digits is listed in Table 4.3. We observe that the average error rate of recognizing synthetic digits by reusing Latin-2 digits is 1.79±0.12% which is *lower* than the baseline 1.88±0.14% approach. This result is supported by a similar result when reusing Latin or Arabic digits. However, the differences are not statistically significant.

4.3.3.3 Handwritten digits as a superset of synthetic digits: RT

The recognition of Latin-2, Latin and Arabic digits reusing unsupervised features trained with synthetic digits, is a reverse problem. We observe that the average error rate of recognizing Latin-2 digits is 3.27±0.16% which is *higher* than the baseline 2.92±0.10% approach. This result is supported by a similar result when reusing Latin or Arabic digits. In addition, we observe degrading performance in the case of recognizing geometrical shapes by reusing unsupervised features of canonical shapes. The degradation is statistically significant in all cases.

4.3.4 TLs

In this section we discuss the performance of the TLs approach both for *hard transfer* and *reverse transfer* problems using SDA models. Eight different layerwise transfer settings were studied as listed in the first column of Table 4.4 (see also Table 4.1) the values are marked bold when they performed significantly better than the baseline.

4.3.4.1 Reuse supervised features for HT: Different Label sets

Let us consider a HT problem of unequal label sets, $Y_S \neq Y_T$ drawn from different distributions. For example, classifying images of lowercase letters from a-to-z by reusing supervised features of handwritten digits.

In case of the "L1", the average error rate of uppercase letters, 4.72±0.18% was significantly *lower* than the baseline, 5.01±0.27%. Similar results are obtained for the lowercase letters. In both cases the significance level allows rejecting the null hypothesis of equal error rates. We observe a reduction in computation time with large standard deviation. That may be due to the fine-tuning stopping criteria.

When reusing a single layer L1, L2 or L3, we observe that the features of the low-level/generic layer lead to lower classification error. When reusing multiple layers L1+L2, L2+L3, L1+L3, we observe that reusing L1+L3 performs better than the reuse of L1+L2 for both uppercase and lowercase datasets. Reusing all three layers L1+L2+L3 has degraded performance as the supervised

features are well tuned for the source problem and fine-tuning only the logistic regression layer does not compensate for good features for the target problem. Thus reusing higher layer (L3 of the network) supervised features is not as good as reusing low-level/generic features layer (L1 layer of the network) supervised features.

In the case of "FT", the average error rate of uppercase letters is 4.58±0.19% and is significantly *lower* than the baseline 5.01±0.27%, with 54% speed up w.r.t the baseline a significant reduction in average computation time. Similar results are obtained for the lowercase letters. In both cases the significance level allows rejecting the null hypothesis of equal error rates.

4.3.4.2 Reuse supervised features for HT: Equal label sets

In the same way, let us consider a HT problem of equal label sets $Y_S = Y_T$ drawn from different distributions. For example, a problem of recognizing geometrical shapes by reusing supervised features trained with canonical shapes. We observe performance improvement namely in case of the "L1+L3", $\overline{\varepsilon}$= 5.23±1.45% was significantly *lower* than the baseline $\overline{\varepsilon}$=7.88±0.93%. The significance level allows rejecting the null hypothesis of equal error rates.

4.3.4.3 Reuse supervised features for RT

Let us now consider two problems: 1) Classifying digits by reusing supervised features of a machine trained with letters is a case of $Y_S \neq Y_T$, and 2) Classifying geometrical shapes reusing supervised features of a machine trained with canonical shapes is a case of $Y_S = Y_T$. In both cases, we observe degrading performance with respect to the baseline (negative feature transference) is shown in Table 4.4 reverse transfer column.

Table 4.5: Average Test Error (%) ($\overline{\varepsilon}$) by reusing harder problem Latin-2 for classifying either Lowercase or Uppercase letters.

	Approaches	P_{LC}		P_{UC}	
		$\overline{\varepsilon}$	Time(s)	$\overline{\varepsilon}$	Time(s)
BL	SDA	4.95±0.16	2997	5.01±0.27	2567
TL	TLs:L1	4.72±0.17	2261	4.72±0.18	2515
TL	TLu	4.67±0.38	1148	4.65±0.19	1498
TL	TLs:FT	**4.57±0.08**	**1020**	**4.58±0.19**	**1180**

4.3.5 Layerwise Transfer Learning for CNN

In the following experiments we compare baseline (BL) and transfer learning supervised (TLs) approaches classification error ε using test dataset, for different amounts of training samples per class, N/c. Given a training, validation and test sets we can design a classifier, apply transference using either CNN or SDA model to obtain test errors by following the procedure described in Algorithm 3.

Algorithm 3 Experimental procedure.

Select a source D_S and a target D_T dataset $\in$ (Latin or Arabic or Lowercase or Uppercase). Both D_S and D_T dataset contains a subset of training, validation and test samples, i.e., $(D_{S.train}, D_{S.valid}, D_{S.test})$ and $(D_{T.train}, D_{T.valid}, D_{T.test})$ respectively.

Repeat the below experiment for for different fractions of whole training sample such that $N \in [100, 250, 500, 1000, 1320, 2500, 5000]$:

1. Run the baseline approach;
2. Obtain a new $D_{T.train.frac}$ by randomly picking N samples from the complete $D_{T.train}$ samples;
3. For each *TL* approach such that $L \in [L1, L1+L2, ...]$ from Table 4.1;
   (a) Lock k-th layer of the network trained on D_S;
   (b) Retrain the network using $D_{T.train.frac}$ except the k-th layer;
   (c) Test the network using dataset D_T, to obtain the classification error ε.

First we select a source and target dataset from any of the four datasets: Latin, Arabic, Lowercase or Uppercase. We train the source dataset D_S for each N samples that were randomly picked from a set of different amounts of training samples $[100, 250, 500, 1000, 1320, 2500, 5000]$. In each of this iteration; step (1) we run the baseline approach, step (2) we build $D_{T.train.frac}$ by randomly picking N samples. Finally, in step (3), we apply the various layer based feature transference approaches as listed in Table 4.1.

4.3.5.1 Classifying digits reusing letters: RT

Let us consider a problem of classifying images of lowercase a-to-z by reusing supervised features of digits 0-to-9. We train a CNN to solve Latin handwritten digits and reuse it to solve a Latin synthetic characters without having to train it from scratch, is a case of reverse transfer learning problem. We perform TLs approach by applying Algorithm 3 for classifying images of both Lowercase and Uppercase Latin synthetic characters. The results are presented in Table 4.6.

From Table 4.6 we observe the classification results of single layer approach: L1, L2 or L3. Reusing the low-level layer supervised features of Layer one (L1) not only performs better than L2 and L3 but also performs better than the baseline approach. Except in case of classifying uppercase characters using $N/c = 1320$, which rises the question that does the source model needs more number of samples per class to classify, $N/c = 5000$. In that case the L1 approach performs better than the rest of the approaches.

Performing TLs by transferring multiple layers at a time like L1+L2, L2+L3 or L1+L3, as listed in Table 4.1 approach number 5, 4 and 3. We observe that reusing L1+L2 layers (lower-level layers) perform better than reusing L1+L3 and L2+L3 for both uppercase and lowercase datasets. This supports our previous conclusion that low-level layer features of L1 are better and it would be better to use low and middle layers at the same time. We can also observe that the

Table 4.6: Percent average classification test error (standard deviation) obtained for different approaches, dataset, and numbers N/c of design samples per class for layer based, supervised feature transference for CNN model.

Approaches:	P_{UC} reuse P_L		P_{LC} reuse P_L	
N/c:	1320	5000	1320	5000
$L1+L2+L3$	5.96(0.13)	5.32(0.18)	6.13(0.13)	5.63(0.15)
$L1+L3$	4.49(0.14)	4.24(0.10)	4.75(0.13)	4.57(0.09)
$L1+L2$	3.61(0.12)	**3.39(0.12)**	**3.83(0.06)**	**3.63(0.13)**
$L3$	4.30(0.13)	4.20(0.16)	4.62(0.18)	4.61(0.14)
$L2$	3.54(0.14)	3.43(0.06)	3.72(0.11)	3.58(0.15)
$L1$	3.43(0.11)	**3.35(0.09)**	**3.64(0.06)**	**3.56(0.11)**
BL	**3.42(0.10)**	3.42(0.10)	3.65(0.12)	3.65(0.12)

reusing multiple layers:L1+L2 features are better than reusing only middle (L2) or only top (L3) layer features.

Finally, reusing all three layers: L1+L2+L3 has degraded performance as the complete supervised features are well tuned for the source problem and training only the logistic regression layer has no improvement.

We observe from Table 4.6 and Table 4.7 that increasing the number of samples per class improves the classification of uppercase characters with L1 features using CNN. Also Table 4.7 provides the comparison of classification results between both CNN and SDA (Kandaswamy et al., 2014c) models.

4.3.6 Analysis of TLu and TLs for SDA model

Analyzing the results of the TLu and TLs approach, we conclude that the unsupervised feature transference improves performance in the case of hard transfer problems. The TLs approach performs better for hard transfer problems, when either using "L1" or "FT" approach. The "FT" approach has the least classification error among all approaches.

To summarize, let us consider the problem of classifying images of either from a-to-z or from A-to-Z (lowercase or uppercase) letters, using unlabeled images of the Latin digits from 0-to-9 of the Latin-2 dataset. To have a fair comparison we use the Latin-2 dataset, which has the same number of instances as the lowercase or uppercase letters dataset. The Table 4.5 gives the summary average error rates for the TLu and TLs approaches.

Analyzing the performance difference of transferring either Arabic or Latin dataset we conclude that even though both Arabic and Latin datasets are both handwritten digits with equal number of instances, the average classification error rate of Latin is higher than Arabic dataset. Thus, Latin is a harder problem than the Arabic dataset. We observe that the unsupervised features trained with the Latin dataset and reused to classify lowercase dataset had lower $\overline{\varepsilon}$ error than reusing features trained with the Arabic dataset. Similar results were observed in the case of uppercase dataset, as listed in Table 4.3. We studied TLs approach between Arabic and Latin

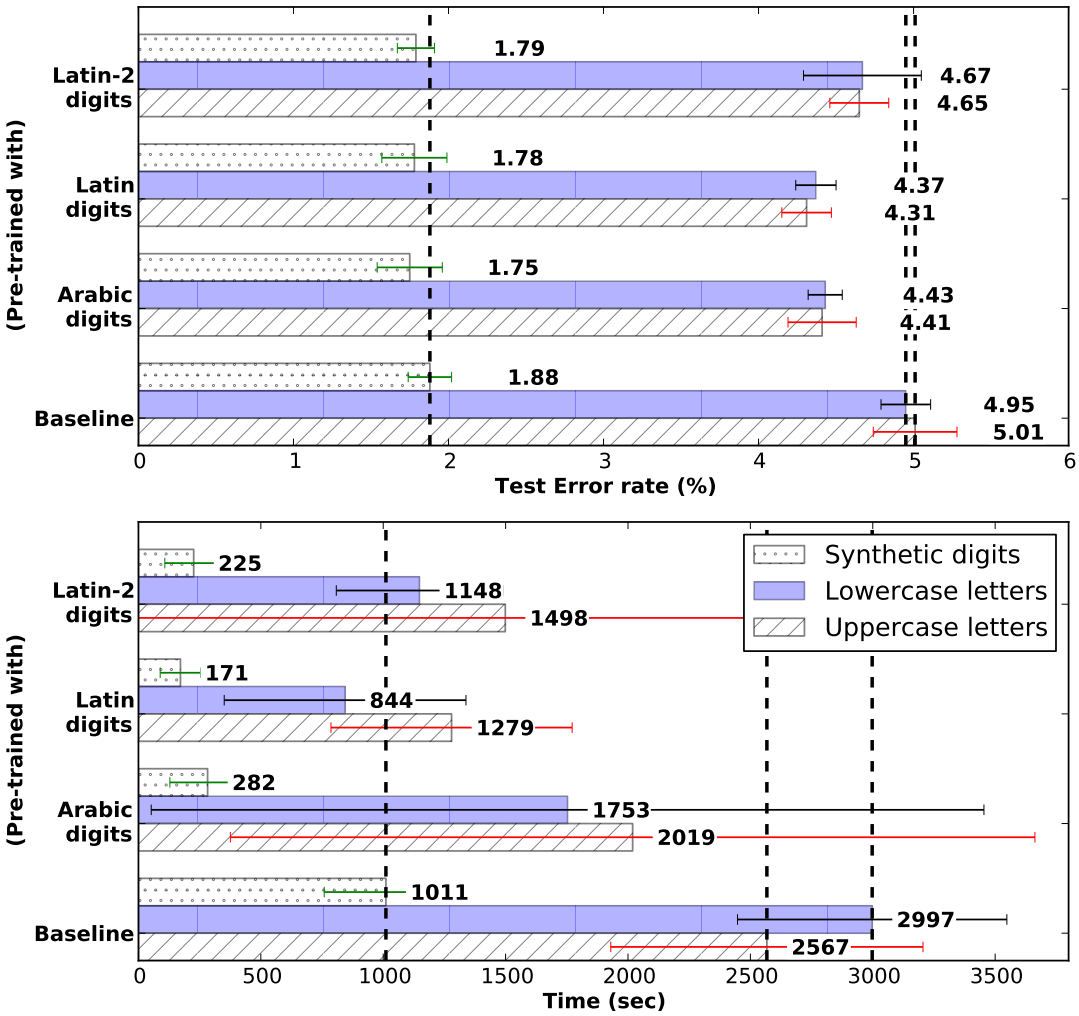


Figure 4.5: Comparison between TLu and baseline (dotted vertical line) for hard transfer problems. Top: Average test error rate (%) ($\overline{\epsilon}$) on Synthetic digits, Lowercase and Uppercase letters datasets by reusing unsupervised features either from *Arabic* or *Latin* or *Latin-2* dataset. Bottom: Computational time for the same experiments, in seconds. Box whiskers are standard deviations.

datasets using CNN model (Kandaswamy et al., 2014a). The result supports our conclusion of hard transfer performs better than reverse transfer.

On the other hand, both TLu and TLs show negative transference (degrading performance with respect to the baseline) in the case of reverse transfer problems. It seems that the features transferred from simpler problems to harder problems (from different distributions) are not well suited for the target problem.

A graphical illustration of the performance of TLu and baseline approaches is shown in Fig.4.5 (*hard transfer*) and Fig.4.6 (*reverse transfer*). To highlight the differences, the baseline averages are plotted as a dotted vertical line for each target problem. To summarize, the TLu approach shows positive transference when the machine is trained on hard transfer problems but negative transference when the machine is trained on reverse transfer problems.

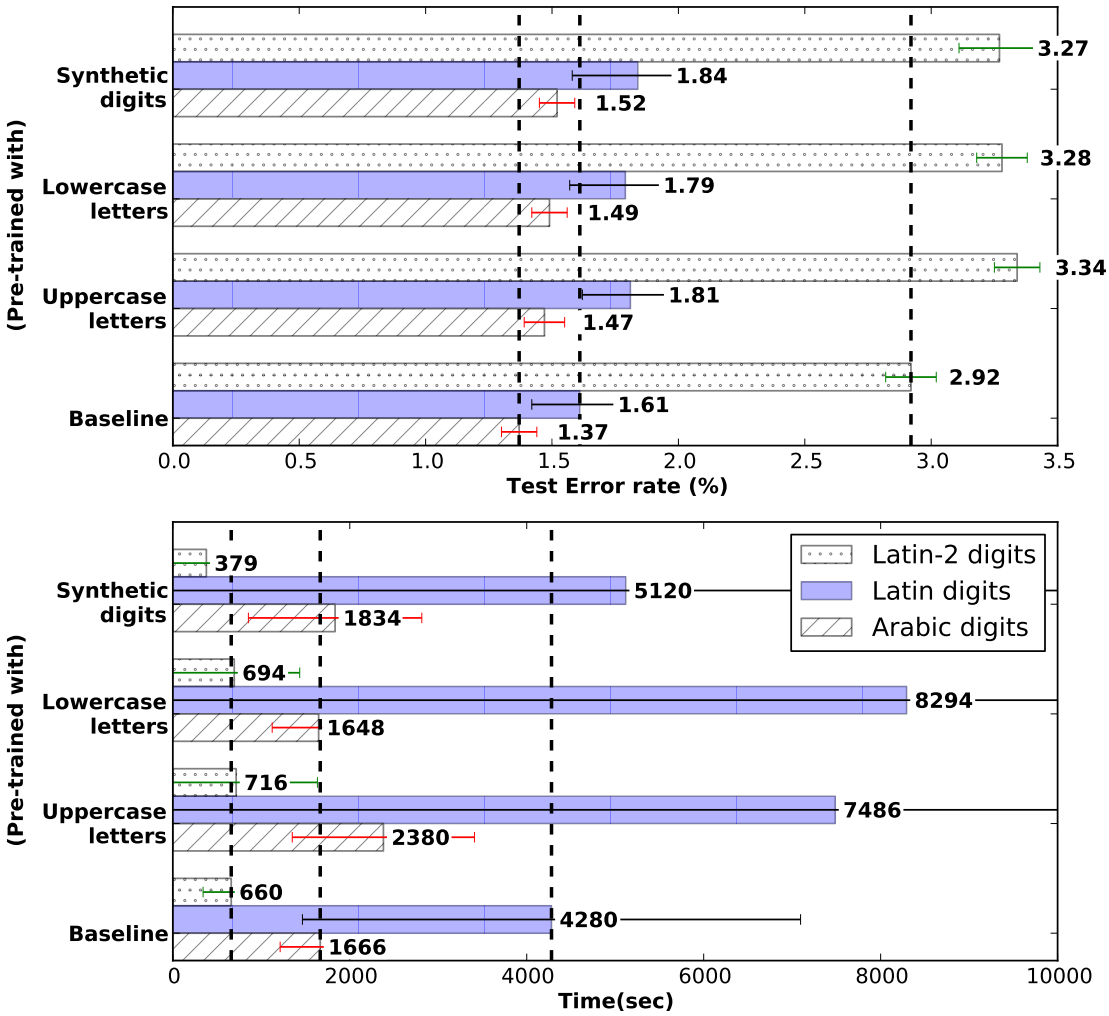


Figure 4.6: Comparison between TLu and baseline (dotted vertical line) for reverse transfer problems. Top: Average test error rate (%) ($\overline{\epsilon}$) on Arabic, Latin and Latin-2 datasets by reusing unsupervised features either from *Synthetic digits* or *Lowercase* or *Uppercase letters* dataset. Bottom: Computational time for the same experiments, in seconds. Box whiskers are standard deviations.

4.3.6.1 Transference From Arabic digits to Latin digits and Vice-versa

We conclude that reusing L1 and L1+L2 perform better than other approaches for both uppercase and lowercase datasets. To increase the confidence of the above conclusions we perform experiment with different datasets by varying the number of samples per class on TL by applying Algorithm 3.

We performed experiments to train a CNN with Latin digits and reuse it to classify Arabic digits and reverse the role of source and target datasets. We observe that reusing the features trained using Latin digits have better performance than reusing Arabic digits, as shown in Fig 4.7 and Fig 4.8.

Table 4.7: Percentage Average Error by reusing Latin at $N/c = 1320$

Approaches		Lowercase Test Error %	Uppercase Test Error %
SDA	*BL*	4.95±0.16	5.01±0.27
SDA	*L*1	4.72±0.17	4.72±0.18
SDA	*PT*	4.67±0.38	4.65±0.19
SDA	*FT*	4.57±0.08	4.58±0.19
CNN	$L1+L2$	3.83±0.06	3.61±0.12
CNN	*BL*	3.65±0.12	**3.42±0.10**
CNN	*L*1	**3.64±0.06**	3.43±0.11

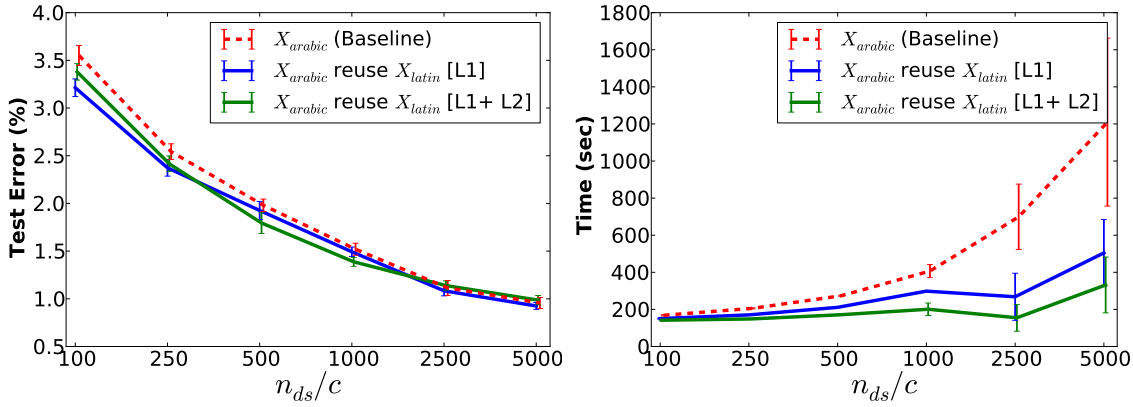


Figure 4.7: Classification results on MAHDBase dataset (Arabic digits) for feature transference approach by reusing various layers, for different numbers N/c of training samples per class. **Left:** Average classification test error rate. **Right:** Average time taken for classification.

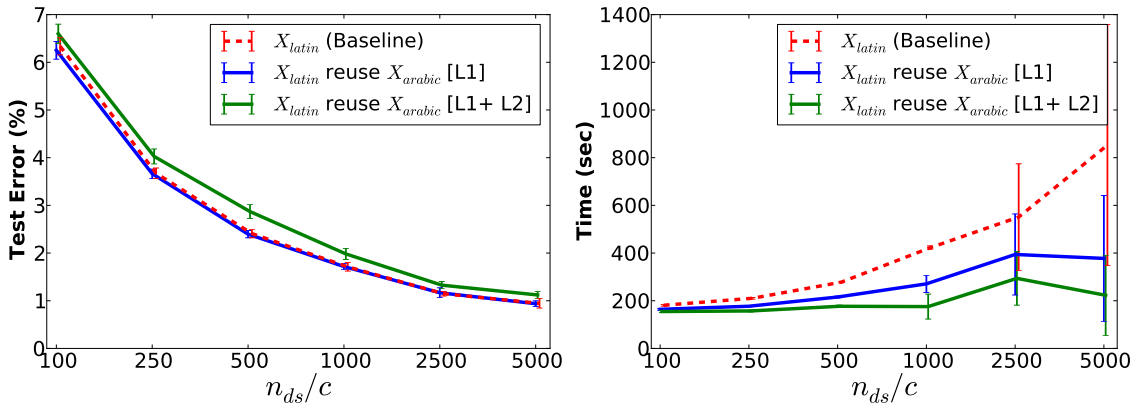


Figure 4.8: Classification results on MNIST dataset (Latin digits) for feature transference approach by reusing various layers, for different numbers N/c of training samples per class. **Left:** Average classification test error rate. **Right:** Average time taken for classification.

4.4 Conclusions

We studied the performance of both unsupervised (TLu) and supervised (TLs) feature transference approaches where the source and target instances were drawn from different distributions. The results showed significant reduction in average error rate and computation time from the baseline for hard transfer problems. In the TLu approach, we achieved a 7% improvement on accuracy and 41% reduction on computation time for uppercase datasets. Similar results were observed in lowercase datasets. Subsequently, in TLs approach we achieved lower average error rates than the baseline. The best result was obtained for TLs:FT approach in which we reused the three supervised hidden layers of the source problem for solving the target problem and it resulted in a 54% speed up w.r.t the baseline. We observed that features trained on harder problems are generic and are able to adapt better to the target problem than ones trained on simpler problems. In addition, by transferring features from geometrical shapes to canonical shapes, we achieved a 7.4% relative improvement on the average error rate in the TLs approach. We also observed negative transfer learning (performance degradation) in both approaches for reverse transfer cases drawn from different distributions.

We proposed a layer based feature transference approach that supports standard neural networks like CNN and SDA for solving hard transfer and reverse transfer learning problems. By transferring either low-level or high-level layer features on machines trained either in unsupervised or supervised fashion. Using this approach we achieved performance improvement with significant reduction in computation times and also decreased the classification error rate. We achieved significant performance by transferring learning from source to target problem, by using low-level layer features trained in supervised fashion in the case of CNN's and high-level layer features trained in a unsupervised fashion in the case of SDA's.

In some cases a feature transference of large labeled data collection is cheaper than collecting and labeling data for the new problem. The lower cost may be due to the fact that the unlabeled and/or labeled data from a related problem are in abundance. As a consequence of these advantages of LTL, we have applied it in practical application covering a broad range of problems. In Chapter 7, LTL is applied to the analysis of breast cancer cell for drug discovery and in Chapter 8, LTL is applied for the identification of persons using the periocular region.

As already indicated, however, LTL has its limitations. One practical limitation is that the source problem need to be harder than the target problem especially for character and object recognition tasks. The solution to this challenge is beyond the state-of-the-art of deep transfer learning methods. It would be interesting for future research to examine ways to avoid negative transference of features for improving classifier performance. Consequently, in Chapter 5 and Chapter 6 we investigate the above discussed challenges.

Chapter 5

Source-Target-Source [1]

5.1 Source-Target-Source mechanism

We propose a STS approach [2]. The main idea of transfer learning is that the knowledge (features) learnt in a source domain provide a good initialization for the learning task in a target problem, better than starting the learning in the target domain at random (likely to get stuck in a poor local optimum). In here we propose to iterate the learning between both domains. The intuition is that, like in typical metaheuristics in optimization (i.e. tabu search and simulated annealing), moving the learning from one domain to the other will 'shake' the current local optimal solution, allowing us to keep exploring the space of solutions (ideally, allowing us to reach a better solution in the process). Likewise the metaheuristics in optimization, we keep track of the solutions reached in each iteration, and the outputted solution is the best of all. The pseudo-code for the STS process is listed in Algorithm 4.

5.2 Multi-source Source-Target-Source mechanism

We extend the STS methodologies by reusing knowledge learnt by a model from training on multiple sources. In recent years, two different approaches have attempted to account for the reuse of multiple sources for TL: a) Lifelong Learning (Thrun, 1998) and b) Multitask Learning (Caruana, 1997). Both approaches are based on specific TL scenarios and assume that the data and the tasks are related.

Both Lifelong Learning and Multitask Learning approaches suffer from some limitations. For example, if two tasks are negatively correlated, the learning process will cause degradation of the generalization performance of both tasks. In order to avoid such issues, a strong task selection is required in order to restrain the application of such methods to a limited set of positively correlated problems. The Multi-Source Source-Target-Source (MS-STS) approach improves generalization performance over multiple problems, with no need for prior task selection.

[1] Some parts of this chapter are used in (Kandaswamy et al., 2015b)

[2] The naming 'source' and 'target' is some what misleading in our learning framework.

Algorithm 4 Pseudocode for STS

Initialize with trained features D_T:
Two datasets D_S and D_T, with tasks Y_S and Y_T are drawn from P_S and P_T distributions.
Select D_S dataset to train
baseline: train network A as shown in the baseline approach
Set value to max cycles
list of max cycles errors to zero
for R in max cycles **do**
    transfer: transfer features from network A to new network B as shown in the transfer learning approach
    update errors list with best test error
    if cycle = odd number **then**
        STS R = test error for Dataset D_S
    else
        STS R = test error for Dataset D_T
    end if
    if error $<$ avg(errors list) **then**
        BREAK
    end if
    Switch between dataset D_S and D_T
end for

The MS-STS approach is briefly explained along with the pseudocode in Algorithm 5. Given a pool set containing multiple datasets from a similar problem for solving a particular application, $Pool = \{D_A, D_B, \ldots, D_Z\}$ drawn from $P_A(X), P_B(X), \ldots, P_Z(X)$ distributions respectively, where Z is number of datasets. We select a deep neural network architecture and initialize the weights of the each layer of the network using uniform distribution under the limits as shown in Step 1 of Algorithm 5. Initializing the weights through this method narrows down the gradient search parameter thus speeding up the training of the network Glorot et al. (2011). Heuristically, we set max number of cycles $R = 10$ and can vary depending on the nature of the problem. In step 2, we select a desired target dataset D_{dT} from the *Pool* for which we intend to have the best overall accuracy. During each of R cycles a target dataset D_T is selected among the pool of datasets for which we apply deep transfer learning approach as discussed in Section 4.1.2 and Source-Target-Source approach as discussed in Kandaswamy et al. (2015b) by selecting layers to transfer and/or to lock in the new network.

The new network is trained and tested as regular deep network for the selected D_T. A list of best accuracies for each cycle is maintained for every dataset in the pool. If the current cycle test accuracy for the desired target dataset D_{dT} is greater than the average of top 3 best test accuracy for the desired target dataset D_{dT}. We break the cycle and store the final weights of the network. The training and testing serially on multiple datasets improves the domain generalization property on the approach and focusing on desired target dataset helps improve the domain specialization property also, with higher focus and performance on the first of such properties.

It is necessary here to clarify exactly what is meant by Multi-source Source-Target-Source

Algorithm 5 Pseudocode for MS-STS

Randomly initialize the weights of every layer of the network using uniform distribution: $\mathbf{w}^k \sim \mathscr{U}\left[\frac{-\sqrt{6}}{\sqrt{N_k+N_{k+1}}}, \frac{\sqrt{6}}{\sqrt{N_k+N_{k+1}}}\right]$
select a desired target dataset D_{dT} from the $Pool = \{D_A, D_B, \ldots, D_Z\}$.
for r in R cycles **do**
  for p in *Pool* **do**
    $D_T = p$ {set p as target dataset}
    {transfer features to new network as discussed in the transfer learning approach Kandaswamy et al. (2015b) and select which of the layers to transfer and which of the layers to lock, out of K layers}
    for k in K **do**
      $\mathbf{w}_S^k \Rightarrow \mathbf{w}_T^k$ {transfer selected layers}
      $\mathbf{w}_S^k \Updownarrow \mathbf{w}_T^k$ {lock selected layers}
    end for
    train and test the new network with D_T
    update the test accuracy list
    if accuracy of D_{dT} > avg(top 3 best test accuracy in D_{dT} accuracy list) **then**
      BREAK
    end if{Continue MS-STS step 3 till global optima is reached for the D_{dT}.}
  end for
end for

(MS-STS) as all of the above mentioned methods also use multiple sources to train the network. MS-STS proposes to extend the established STS methodology with multiple sources instead of only single source. The intuition is that, providing multiple initialization points for exploring the space for optimal solution may allow us to reach a better solution. The search may increase the computational cost.

We explore the MS-STS to improve generalization over multiple problems, not just one problem. This approach has both the ability of domain generalization and domain specialization properties, where in it has higher performance for task solved in perspective of domain generalization properties.

5.3 Experimental Setup and Results

Training Deep Neural Network: The network we used in character recognition experiments had three hidden layers with [576, 400, 256] units and the networks used in object recognition experiments also had three hidden layers with [100, 200, 300] units. Both networks have an output layer appropriate to the number of classes being considered. All hidden layers were pre-trained as denoising autoencoders via gradient descent, using the cross-entropy cost and a learning rate of 0.001. Pre-training ran for a minimum of 50 epochs in the case of character recognition tasks, and for a minimum of 60 epoch when using object recognition tasks. The complete networks were fine-tuned via gradient descent, using the cross-entropy cost and a learning rate of 0.1. The

fine-tuning ran until the validation error did not decrease below 0.1% or until 1000 epochs for all tasks. Our code for experiments was based on the Theano library 6 and ran with the help of a GTX 770 GPU.

5.3.1 Transferring specific features Vs. generic features for STS approach

In this experiment, we intentionally set adverse configurations for feature transference, to study the two main causes of negative feature transference. First, by transferring specific features on tasks that are different, $Y_S \neq Y_T$ we focus on feature specialization in tasks 1 to 4 as listed in Table 5.1. Second, by transferring generic features on distributions that are similar, we focus on splitting of co-adapted neurons between layers in tasks 5 & 6 in Table 5.1. Here we study the effects of negative feature transference problems with few training samples.

First, we study the effects of transferring *specific features* on character recognition problem. In Table 5.1 for TLu and TLs approach, tasks 1 & 2 have shown negative transference for classifying handwritten digits P_L by reusing source network P_{LC} and P_{UC}. Tasks 3 & 4 show positive transference for classifying either P_{LC} and P_{UC} by reusing source network P_L training on complete data. We observe for tasks 1 to 4 that STS outperforms other approaches for few target samples. In tasks 1 & 2, STS outperforms BL with a relative improvement of ≈59% and in tasks 3 & 4 STS shows ≈30% improvement for 0.05% of target data. Fig 5.1. illustrates the relative improvement performance of BL, TLu, TLs and STS approaches for tasks 1 & 2.

Second, tasks 5 & 6 analyse the effects of transferring *generic features* on object recognition problems as shown in Table 5.1. Intuitively canonical objects are a subset of non-canonical objects (equilateral triangles are a subset of triangles), thus $P_{Sh1} \subseteq P_{Sh2}$. The number of categories to classify in source and target tasks are equal $Y_S = Y_T$, thus the only change is due to splitting of co-adapted neurons between the layers while fine-tuning, as we have forced to *lock* the bottom layer, making the optimization harder. cBL, TLu and TLs approaches show negative transference as intended. As solving non-canonical objects is more difficult than solving canonical objects (Kandaswamy et al., 2014c), with STS we observe a relative improvement of ≈81% for the same task using 0.05% of total training data the baseline approach performs better when using complete training data. Fig 5.2. shows the non-canonical task features for BL, cBL, TL and STS approach using 0.05% of total training data.

To solve for the complete target data using STS, we implement repeating several cycles of STS (see Algorithm 4) till a certain stopping criteria is reached. We observe significant improvements using STS over both positive and negative transferred features using TLs as listed in Table 5.2.

5.4 Conclusions and discussion

Our experiments with the character and object recognition tasks show that a deep neural network learns a new task more quickly and accurately using transfer learning. Unfortunately, they are unreliable for different source and target distributions, because sometimes they lead to negative

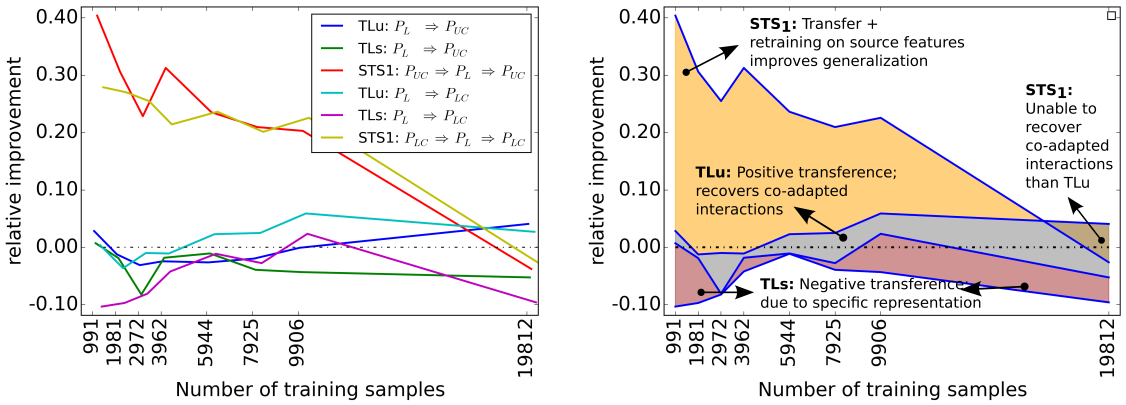


Figure 5.1: (**Left:**) Relative improvement over baseline approach for character recognition tasks 3 & 4 as listed in Table 5.1; (**Right:**) Relative improvement for the tasks on the left, the regions are enclosed to observe relative improvement between two different approaches. We observe negative transference for **TLs** (supervised) approach as it gets stuck at local solution space of specialized features. **TLu** (unsupervised) approach easily recovers the fragile co-adapted neurons as the unsupervised features are not target specific. Also TLu improves over the baseline for complete training data. **STS** approach as intended shake the current local optimal solution, thus overcoming the specialized features of source network unlike TLs approach. The STS shows performance improvement, but unable to recover the fragile co-adapted neurons thus using complete target data, had lower performance than TLu and baseline.

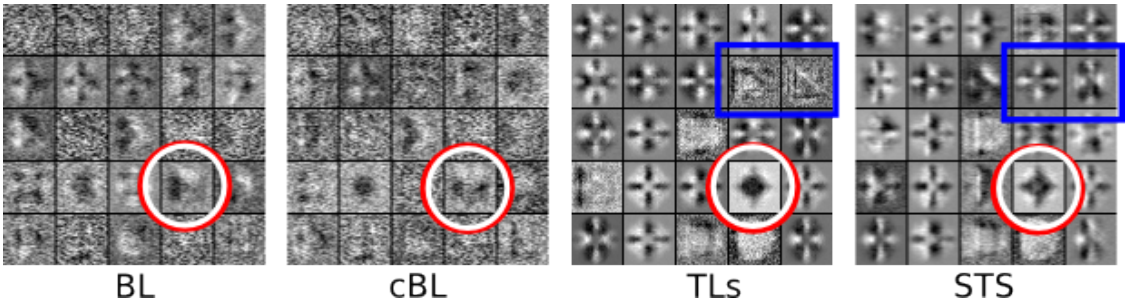


Figure 5.2: Feature samples from first layer of non-canonical object recognition task. We observe the transition of same features becoming more distinct, from BL towards STS approach are marked in red circle and from TLs towards STS marked in blue box.

Table 5.1: Comparison of percentage average error rate ($\overline{\varepsilon}$) for BL, cBL, TLu, TLs and STS approach for different ratios of target data (P_T) reusing source (P_S) distribution. Tasks 1 to 4 study *specific* feature transfer on character recognition problem and tasks 5 & 6 study *generic* feature transfer on object recognition problem.

		Approach			**Ratio of total number of training samples**							
			P_T	P_S	0.05	0.1	0.2	0.3	0.4	0.5	1	#
Characters	Tasks are different	BL	P_L		6.4 (0.1)	4.7 (0.1)	3.3 (0.1)	2.7 (0.2)	2.3 (0.0)	2.3 (0.4)	1.5 (0.1)	
		TLu	P_L	P_{LC}	7.4 (0.2)	5.3 (0.1)	3.8 (0.2)	3.5 (0.7)	2.7 (0.2)	2.5 (0.2)	2.3 (0.0)	(1)
		TLs	P_L	P_{LC}	7.4 (0.1)	5.8 (0.2)	4.6 (0.2)	3.7 (0.0)	3.2 (0.2)	2.9 (0.1)	2.1 (0.1)	
		STS	P_L	P_{LC}	**2.6 (0.1)**	**2.1 (0.0)**	**2.0 (0.1)**	**1.9 (0.1)**	**1.8 (0.0)**	**1.7 (0.1)**	**1.5 (0.0)**	
		BL	P_L		6.4 (0.1)	4.7 (0.1)	3.3 (0.1)	2.7 (0.2)	2.3 (0.0)	2.3 (0.4)	1.5 (0.1)	
		TLu	P_L	P_{UC}	7.4 (0.3)	5.6 (0.6)	4.0 (0.2)	3.1 (0.1)	2.8 (0.2)	3.0 (0.5)	2.1 (0.3)	(2)
		TLs	P_L	P_{UC}	7.6 (0.3)	5.8 (0.2)	4.4 (0.2)	3.5 (0.0)	3.1 (0.0)	2.7 (0.0)	2.0 (0.1)	
		STS	P_L	P_{UC}	**2.4 (0.0)**	**2.2 (0.2)**	**1.9 (0.0)**	**1.7 (0.1)**	**1.7 (0.1)**	**1.6 (0.1)**	**1.5 (0.0)**	
		BL	P_{LC}		17.1 (0.1)	13.3 (0.2)	10.8 (0.1)	9.5 (0.1)	8.4 (0.1)	7.7 (0.6)	4.8 (0.1)	
		TLu	P_{LC}	P_L	17.1 (0.6)	13.8 (0.6)	10.9 (0.2)	9.2 (0.4)	8.2 (0.4)	7.2 (0.2)	**4.7 (0.2)**	(3)
		TLs	P_{LC}	P_L	18.9 (0.2)	14.6 (0.8)	11.3 (0.2)	9.6 (0.2)	8.7 (0.4)	7.5 (0.2)	5.3 (0.3)	
		STS	P_{LC}	P_L	**12.3 (0.3)**	**9.7 (0.0)**	**8.5 (0.6)**	**7.2 (0.4)**	**6.7 (0.2)**	**6.0 (0.1)**	5.0 (0.2)	
		BL	P_{UC}		16.2 (0.2)	12.9 (0.2)	10.4 (0.2)	9.1 (0.1)	8.5 (0.7)	7.3 (0.5)	4.9 (0.2)	
		TLu	P_{UC}	P_L	15.9 (0.3)	13.2 (0.4)	10.8 (0.3)	9.1 (0.3)	8.0 (0.1)	7.4 (0.3)	**4.6 (0.1)**	(4)
		TLs	P_{UC}	P_L	16.5 (0.3)	13.6 (0.5)	10.8 (0.2)	9.2 (0.2)	8.5 (0.2)	7.4 (0.1)	5.0 (0.2)	
		STS	P_{UC}	P_L	**10.8 (0.4)**	**9.1 (0.1)**	**7.8 (0.2)**	**6.8 (0.1)**	**6.6 (0.1)**	**6.1 (0.1)**	4.7 (0.1)	
Objects	Tasks are similar	BL	P_{Sh2}		37.9 (10.2)	36.6 (4.8)	25.1 (3.6)	16.9 (9.6)	14.7 (7.8)	11.9 (7.1)	**4.2 (2.3)**	
		cBL	P_{Sh2}	P_{Sh1}	28.7 (6.3)	13.6 (2.2)	12.6 (10.4)	9.9 (8.0)	6.6 (3.0)	13.0 (8.4)	10.6 (6.7)	(5)
		TLs	P_{Sh2}	P_{Sh1}	32.3 (2.3)	32.0 (3.3)	30.7 (4.1)	26.9 (1.7)	26.4 (1.9)	27.0 (1.3)	24.0 (0.3)	
		STS	P_{Sh2}	P_{Sh1}	**7.7 (2.6)**	**6.2 (2.4)**	**5.9 (3.5)**	**5.4 (3.1)**	**5.3 (2.6)**	**5.0 (3.0)**	5.2 (2.2)	
		BL	P_{Sh2}		37.9 (10.2)	36.6 (4.8)	25.1 (3.6)	16.9 (9.6)	14.7 (7.8)	11.9 (7.1)	**4.2 (2.3)**	
		cBL	P_{Sh2}	P_{Sh3}	31.0 (1.8)	30.5 (8.8)	18.4 (11.3)	20.0 (11.2)	5.6 (1.7)	12.4 (7.6)	8.9 (6.6)	(6)
		TLs	P_{Sh2}	P_{Sh3}	25.0 (3.3)	20.7 (1.8)	18.4 (2.0)	18.4 (1.1)	16.8 (1.8)	17.2 (1.7)	15.5 (2.5)	
		STS	P_{Sh2}	P_{Sh3}	**6.1 (2.3)**	**5.9 (2.7)**	**5.8 (2.6)**	**4.9 (2.1)**	**5.0 (2.2)**	**5.7 (3.0)**	5.6 (2.6)	

feature transference. The STS algorithm was designed to avoid negative transfer, since by recovering fragile co-adapted interactions of neurons between the layers.

We make several contributions as listed:

1. The STS approach outperforms both baseline and other transfer learning approaches.

2. We studied TLu and TLs approach for both transferring generic features on distributions that are similar and transferring specific features on tasks that are different. The study demonstrated the flexibility of these two approaches for solving applications that had certain constrains.

3. We studied the impact of splitting of co-adapted neurons and it efficiently improved the domain generalization capability of the network.

4. Finally, using the cyclic STS approach reduced the transferability gap(the ratio of source error rate over target error rate) between the source and the target tasks. We summarize that the STS outperforms both the baseline and the transfer learning approaches.

Even though the cyclic STS reduced the transferability gap between the source and the target tasks, a pattern is observed when the initial transference was negative. In the negative transference case of cyclic STS, we observe the odd cycles performing better than the even cycles. Iteratively

Table 5.2: Comparison of positive vs. negative transference using *complete target data* and retraining all layers; Performance is measured using percent average test error ($\overline{\varepsilon}$) with 10 repetitions; TLs shows **positive** transference for classifying MNIST P_L reusing Lowercase P_{LC} same as Task 1. And **negative** transference for classifying P_{LC} reusing P_L, same as Task 3. In both cases iteratively repeating **STS** outperforms both BL and TLs approaches.

	Iterative STS	**-ve transference**			**+ve transference**		
		D_B	D_A	$\overline{\varepsilon}$	D_B	D_A	$\overline{\varepsilon}$
BL	D_A	P_{LC}	P_L	1.7 (0.3)	P_L	P_{LC}	4.9 (0.2)
TLs	$D_B \Rightarrow D_A$	P_{LC}	P_L	1.9 (0.2)	P_L	P_{LC}	4.5 (0.2)
STS_1	$D_A \Rightarrow D_B \Rightarrow D_A$	P_{LC}	P_L	1.6 (0.1)	P_L	P_{LC}	4.9 (0.1)
STS_2	$D_B \Rightarrow D_A \Rightarrow D_B \Rightarrow D_A$	P_{LC}	P_L	1.9 (0.1)	P_L	P_{LC}	**4.4 (0.2)**
STS_3	$D_A \Rightarrow D_B \Rightarrow D_A \Rightarrow D_B \Rightarrow D_A$	P_{LC}	P_L	1.6 (0.1)	P_L	P_{LC}	4.9 (0.1)
STS_4	$D_B \Rightarrow D_A \Rightarrow D_B \Rightarrow D_A \Rightarrow D_B \Rightarrow D_A$	P_{LC}	P_L	1.9 (0.1)	P_L	P_{LC}	4.5 (0.2)
STS_5	$D_A \Rightarrow D_B \Rightarrow D_A \Rightarrow D_B \Rightarrow D_A \Rightarrow D_B \Rightarrow D_A$	P_{LC}	P_L	**1.5 (0.1)**	P_L	P_{LC}	5.0 (0.1)

switching the training between the source and the target did not sufficiently perturb the solution out of local minima to a new solution space.

Chapter 6

Deep Transfer Learning Ensemble [1]

We propose a Deep Transfer Learning Ensemble (DTLE) where we combine the main advantage of deep transfer learning with traditional ensemble learning. DTL offers generic source domain features as a good initialization for the target problem, which is better than random initialization. Like in a traditional ensemble, the various transfer and/or retrain conditions of DTL combine to provide a *committee* of decision makers for the ensemble, where in the DTL model may transfer features from both the source domains. Numerous empirical and theoretical studies have demonstrated that ensemble (committee) models often obtain higher accuracy than single models (Kuncheva, 2004).

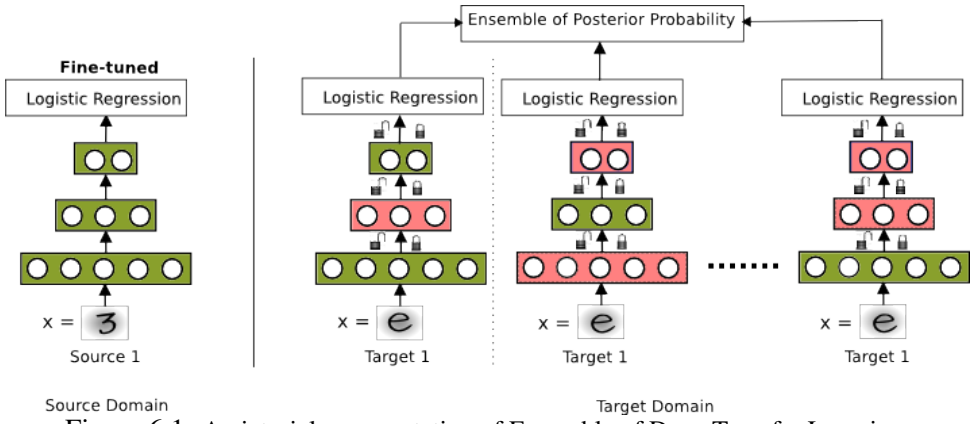


Figure 6.1: A pictorial representation of Ensemble of Deep Transfer Learning.

The overall framework of DTLE depicted in Fig. 6.1, employs a deep model learnt on the source domain and applies DTL with various conditions, like transfer hidden layers (transfer or randomly initialize), and then retrains (locks or unlocks) the network on the target domain. The DTLE also computes posterior probabilities $P_T(y|x)$ for each of the DTL models for the target task. The class probabilities are obtained using the average of all posterior probabilities $P_T(y|x)$ of every model. The models are trained on baseline method (BL) using the standard deep learning and deep transfer learning approaches.

The bottom-layer features are referred to as *general* and the top-layer features as *specific*. The pseudo-code for the DTLE process is listed in Algorithm 6, as a study of the two following

[1] Some parts of this chapter are used in (Kandaswamy et al., 2015a)

conditions of feature transference: 1) transfer specific features, and 2) retrain specific features. In retrain specific features the condition dictates the splitting of inter-layer connections (co-adapted neurons) which leads to difficulty in optimization (Yosinski et al., 2014)).

Notations: We use following simplified notations to ease the readability of the text. We denote deep network architecture as [L1,...,LK] with $K-1$ hidden layers and the K-th layer is the logistic regression layer (LR). We represent feature transference as '1' and not to transfer as '0' in the *transfer* network architecture, similarly '1' represent unlocked layer and '0' represent locked layer in the *retrain* network architecture. For example to denote network architecture for a three hidden layer network as [L1,L2,L3,LR]. Here we transfer all the hidden layers only and do not transfer LR layer of the source network to the target network, we denote the transfer network architecture as [1 1 1 0]. To retain only the top and LR layer we denote the retrain network architecture as [0 0 1 1].

Algorithm 6 Pseudocode for DTLE

Initialize with trained features D_S:
Given two datasets D_S and D_T, with tasks Y_S and Y_T, drawn from P_S and P_T distributions.
Let the total number of models in the ensemble be R
{Select type of TL interaction to evaluate}
if evaluate == co-adapted interactions **then**
    R = possible combination of retrained layers
else if evaluate == generic vs. specific **then**
    R = possible combination of transferred layers
end if
baseline: Train network A using source dataset, D_S as shown in the baseline approach.
for each model R in the ensemble of TL **do**
    transfer: transfer features from network A to new network B as shown in the transfer learning approach
    Compute posterior probabilities $P_T(y|x)$ for target dataset, D_T.
end for
{Combine all the posterior probabilities $P_T(y|x)$ of each model, R}
Compute $y = argmax \sum_{R_i \in R} P_T(y|x)$

6.1 Experimental setup and Results

Training Stacked denoising autoencoders: The network we used in character recognition experiments had three hidden layers with [576, 400, 256] units, batch size of 100 and pre-training ran for a minimum of 25 epochs. The networks used in object recognition experiments also had three hidden layers with [100, 200, 300] units, batch size of 300 and pre-training ran for a minimum of 10 epochs. All hidden layers were pre-trained as denoising autoencoders via gradient descent, using the cross-entropy cost and a learning rate of 0.001. The complete networks were fine-tuned via gradient descent, using the cross-entropy cost and a learning rate of 0.1. The fine-tuning ran until the validation error did not decrease below 0.1% or until 1000 epochs for all tasks. Our code was

based on the Theano library 6 and ran with the help of a GTX 770 GPU. To determine if a result is statistically significant over ten repetition of each experiment, we used paired student t-test to calculate a p-value, which is the probability of observing an effect given that the null hypothesis is true. We marked each result in Table 6.1, with '*' when the result was statistically significant, i.e., if an observed p-value is lower than 0.01 (1%).

6.1.1 Retrain specific DTL

In this section, we study retrain specific DTL (**DTL**$_r$). In this condition of DTL, we transfer all the hidden layers of the source network to the target network, i.e., transfer [1 1 1 1] and retrain only unlocked layers marked as '1', for example retrain [0 0 1 1]. We study the fragile splitting of the co-adapted neurons caused due to locking of the layer, thus stopping learning in that selected hidden layer of the target network. This avoids overfitting of the network for the target task.

Generally the features of the lower layers of the network are generic therefore they can be used to solve a broader spectrum of problem. The higher layer features are specific to the task the network was trained. We would like to re-utilize the generic features of the source network and retrain the transferred network for target specific task. In this section, we study suitable conditions such that we obtain positive transference retraining only specific layers of the target network.

We observe a consistent improvement in **DTL**$_r$ across all the cases of transfer learning for the condition: transfer = [1111] & retrain = [1111]. We conclude that this is due to two main reasons: 1) the transferred layer weights are better than random initialization and, 2) retraining the network without locking any layer improves the chances of better generalization.

We observe statistically significant result for all conditions of **DTL**$_r$ except for transfer = [1111] & retrain = [0001]. This still offer good generalization compared to random initialization, but is lower than in other transfer conditions.

Ensemble of 4 **DTL**$_r$ models gives retrain specific DTLE (**DTLE**$_r$). We observe better average accuracy than BL and **DTL**$_r$ conditions and results are shown in Table 6.1. We perform paired student t-test comparing the accuracy results **DTLE**$_r$ with accuracy results of **DTL**$_r$.

6.1.2 Transfer specific DTL

In this section, we study Transfer specific DTL (**DTL**$_t$). In this condition of DTL, we transfer only specific layers of the source network to the target network, for example transfer [0 0 1 1] and retrain all the layers, i.e., retrain [1 1 1 1]. We study the generic versus specific feature transference due to transferring of the layer, thus reusing the features for the target task. This not only speeds up the training but also improves the accuracy of the network.

We observe that **DTL**$_t$ performs better than BL, even for condition when only the logistic regression layer is transferred and retaining the whole target network with backpropagation algorithm as shown in Table 6.1.

Ensemble of 4 **DTL**$_t$ models gives transfer specific DTLE (**DTLE**$_t$). We observe better average accuracy than BL and **DTL**$_t$ conditions as shown in Table 6.1.

Table 6.1: Percent average classification accuracy obtained for all three possible transfer learning cases; 6 different experiments are performed on three different types of tasks i.e., character, object and biomedical image recognition; We compare established frameworks i.e., Baseline (BL), retrain specific DTL ($\mathbf{DTL}_r$), and transfer specific DTL ($\mathbf{DTL}_t$) with our approach, retrain specific DTLE ($\mathbf{DTLE}_r$), transfer specific DTLE ($\mathbf{DTLE}_t$), and Ensemble of DTL (DTLE); the difference between two datasets distribution and is given by Jensen-Shannon divergence (JSD)

	Marginal	$P_S(X) \neq P_T(X)$		$P_S(X) = P_T(X)$		$P_S(X) \neq P_T(X)$	
	Labels	$Y_S = Y_T$		$Y_S \neq Y_T$		$Y_S \neq Y_T$	
	TL case	I		II		III	
	Experiment	(1)	(2)	(3)	(4)	(5)	(6)
	Source	Non-Canonical	MNIST	Non-Canonical	COMP	MNIST	MNIST
	Target	Canonical	Digit	Curve & corner	MOA	Lower	Upper
	JSD	0.99	0.99	0	0	0.80	0.79
Approaches		**Avg Acc**	**Avg Acc**	**Avg Acc**	**Avg Acc**	**Avg Acc**	**Avg Acc**
BL		99.49(0.32)	97.74(0.09)	98.35(0.27)	96.38(0.5)	94.34(0.13)	94.93(0.13)
Retrain Specific DTL							
	transfer	**retrain**					
$\mathbf{DTL}_r$	[1111]	[1111]	99.51(0.17)	97.92(0.27) *	99.00(0.46)	97.54(0.40) *	94.71(0.22) *	94.94(0.23) *
	[1111]	[0111]	96.92(1.72) *	98.06(0.17) *	97.06(1.47) *	98.07(0.18)	94.52(0.24) *	94.74(0.15) *
	[1111]	[0011]	96.60(1.64) *	98.14(0.18) *	96.79(1.53) *	98.23(0.17)	93.92(0.29) *	94.10(0.15) *
	[1111]	[0001]	95.78(1.91) *	97.36(0.51) *	96.46(1.68) *	98.16(0.22)	89.36(0.83) *	89.57(0.99) *
$\mathbf{DTLE}_r$			**99.57**(0.13)	**98.62**(0.14)	**99.27**(0.31)	**98.24**(0.27)	**95.11**(0.18)	**95.38**(0.18)
Transfer Specific DTL							
	transfer	**retrain**					
$\mathbf{DTL}_t$	[1111]	[1111]	99.51(0.17) *	97.93(0.27) *	99.00(0.46) *	97.54(0.40)	94.71(0.22)	94.94(0.23) *
	[0111]	[1111]	99.73(0.12) *	97.09(0.34) *	**99.60**(0.16) *	96.76(1.22)	92.80(0.26) *	93.96(0.15) *
	[0011]	[1111]	86.83(17.26)	97.29(0.30) *	68.81(5.71)	96.71(0.46) *	93.28(0.28) *	93.97(0.33) *
	[0001]	[1111]	99.84(0.08)	97.46(0.17) *	98.86(0.38) *	96.81(0.65)	93.40(0.17) *	94.38(0.16) *
$\mathbf{DTLE}_t$			**99.91**(0.03)	**98.18**(0.16)	99.58(0.16)	**97.54**(0.53)	**94.56**(0.11)	**95.26**(0.10)
DTLE			99.86(0.03)	**98.87**(0.10)	99.52(0.16)	98.12(0.31)	**95.18**(0.06)	**95.70**(0.16)

6.1.3 Retrain and Transfer specific DTLE

We observe significant improvements in average accuracy using DTLE over both $\mathbf{DTLE}_r$ and $\mathbf{DTLE}_t$ using all the conditions as listed in Table 6.1 except for the transfer learning case II. Firstly, we observe that in $\mathbf{DTLE}_r$, 6 out of 6 experiments obtains better accuracy than BL and other established DTL approaches. Secondly, we observe that in $\mathbf{DTLE}_t$, 5 out of 6 experiments obtains better accuracy than BL and other established DTL approaches. Finally, we observe that in DTLE approach 3 out of 6 experiments obtain better accuracy than BL and other established DTL approaches.

6.2 Conclusions and discussion

We propose an ensemble of deep transfer learning approaches using 9 datasets with varied image recognition tasks like character and object image recognition. Our contributions are as listed below:

1. We analyzed all possible cases of transfer learning, based on change in distribution and change in classification task between the source and the target domains.

2. The experimental analysis of the retrain specific DTL approaches across all possible cases of transfer learning showed that the conditions of transfer all layers and/or retrain all layers, obtained better overall accuracy not only than the baseline, but also in comparison to other DTL_r conditions. This is due to two main reasons: 1) the transferred layer weights are better than random initialization and, 2) retraining the network target task improves the chances of better generalization by forcing splitting of fragile co-adapted neurons.

3. We observed that transfer specific DTL approaches obtained better overall accuracy than the baseline but were not as good as retrain specific DTL, since the fine-tuned weights of the transfer specific DTL forced the solution to the local minima.

4. An experimental analysis of retrain specific DTLE, transfer specific DTLE and DTLE approaches showed that $DTLE_r$, ensemble of posterior probabilities of four DTL_r models, obtained a statistically significant better accuracy than individual DTL_r. DTLE outperformed the baseline and other DTL approaches when the distributions and tasks were different.

In the future we would like to explore the possibility of transferring features from multiple source problems, and combining them under the DTLE framework.

Part III

Deep Transfer Learning Applications

Chapter 7

High-content Analysis of Breast Cancer Cells [1]

High-content Analysis has revolutionized cancer drug-discovery by identifying substances that alter the phenotype of a cell which prevent tumor growth and metastasis. The high-resolution biofluorescence images from the assays allow precise quantitative measures enabling the distinction of small molecules of a host cell from a tumor. In this Chapter, we are particularly interested in the application of Deep Transfer Learning (DTL), a cutting edge machine learning method, to the classification of chemical mechanisms of action (MOA).Compound classification has been performed using image-based profiling methods sometimes combined with feature reduction methods such as principal component analysis or factor analysis. In this article, we map the input features of each cell to a particular MOA class without using any type of profiling or feature reduction methods. To the best of our knowledge, this is the first application of DNN in this domain, leveraging single-cell information. Furthermore, we use Deep Transfer Learning (DTL) to alleviate the intensive and computational demanding effort of searching the huge parameter's space of a DNN and even more, to improve its original performance. Results show that using this approach, we obtain a 30% speed up and a 2% accuracy improvement.

7.1 Introduction

Recent advances in quantitative microscopy and high-performance computing have enabled a rapid progress in the development of high-throughput image-based assays. These high-content analysis (HCA) assays allow not only a precise quantitative observation of multiple parameters like nuclear size, nuclear morphology, DNA replication and many more subtle features derived from each image but also the screening of thousands of cells simultaneously highlighting the complex nature of such data. To tackle this high-throughput high-dimensional problem, biologists tend to use population-averages of per-cell information prior to machine learning (ML) algorithms such as principal component analysis, random forest, K-nearest neighbors or support vector machines.

[1] Some parts of this chapter are used from article (Kandaswamy et al., 2016b)

Moreover, a recent survey (Singh et al., 2014) shows that about 70% of the papers on HCA experiments published in Science, Nature, Cell, and the Proceedings of the National Academy of Sciences from 2000 to 2012 used only one or two of the cell's measured features and less than 15% used more than 6. Unfortunately and due to the exponential increase in the number of product terms (LeCun et al., 1998), such ML algorithms become impractical for these problems with thousands of samples and hundreds of measured features. As a result, about 85% of the research work in HCA underutilized potentially valuable information that might have helped in speeding up early stage drug discovery. In this paper, we are interested in exploring state-of-the-art algorithms developed in the field of artificial intelligence to address these high-throughput high-dimensional data.

The discovery of hierarchical visual sensory processing systems in the neocortex of the mammal brain motivated the field of artificial intelligence to develop algorithms to hierarchically extract information from the data (Serre et al., 2005), (Lee et al., 1998).

Our contribution can thus be summarized as follows: 1) use of per-cell information with all the extracted features from high-content images; 2) use of state-of-the-art deep learning models coupled with GPU computational power to analyze such high-throughput high-dimensional data; 3) use of transfer learning to improve the performance of the models. In particular, we consider Stacked Autoassociators (Vincent et al., 2010), (Amaral et al., 2013) (SAA) as classifiers of MOA on a freely available MFC7 wild-type breast cancer data (Ljosa et al., 2013) using a DTL framework that includes a Transfer Learning supervised (TLs) (Amaral et al., 2013), (Amaral et al., 2014a).

7.2 Materials and Methods

We used a publicly available (`http://www.broadinstitute.org/bbbc`, accession BBBC021) dataset from the genetically engineered MCF7-wt (breast cancer expressing wild-type p53) cell line. Briefly (all details of sample preparation and image analysis can be found in Ljosa et al. (Ljosa et al., 2013)), images of cell cultures with a given treatment (specific compound x concentration combination) were acquired on a high-content imaging platform using a 16-bit camera. Each image was further segmented using CellProfiler Analyst (Carpenter et al., 2006) (CPA) by identifying nuclear and cytoplasmic boundaries. Then, 453 distinct features for each cell representing a variety of geometric, intensity, subcellular localization and texture features (Young et al., 2008) were extracted with CPA. Figure 7.1 shows some examples of captured images representing some of the MOA as well as some of the features extracted with CPA.

Our problem consists in predicting the MOA of a given treatment using per-cell information, in contrast to other established methodologies that use some profiling technique (see Ljosa et al. (Ljosa et al., 2013) for a comparative study). There is a total of 103 treatments corresponding to combinations of 38 compounds at one to seven concentrations. We only used the 148,649 cells of non-control samples thus giving a data matrix with 148,649 lines (representing cells) and 453 columns (representing the extracted features). To perform transfer learning we need to define a

source and a target problem. For that purpose the original MFC7 dataset with 12 MOAs is split into two mutually exclusive datasets with 6 MOAs each, Set1 and Set2. The distribution across the two subsets was performed in order to join MOAs with common batches (see Table 7.1).

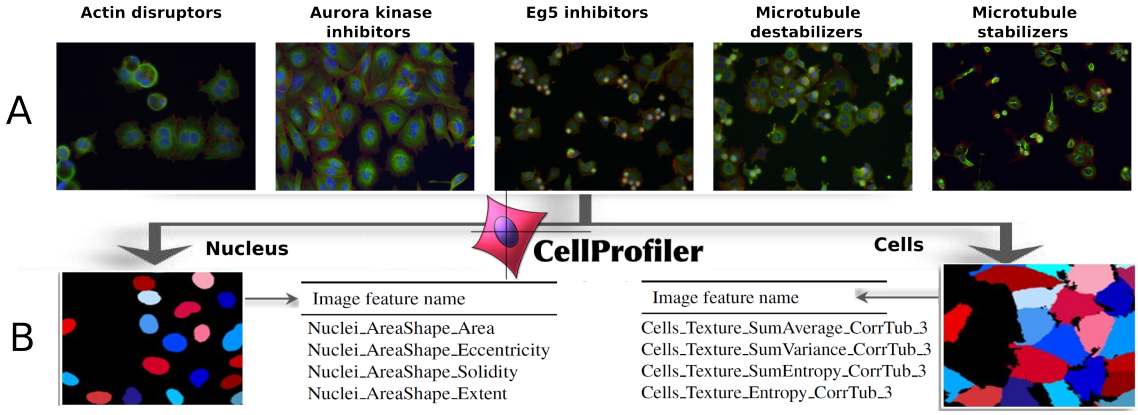


Figure 7.1: A- Examples of different phenotypes (MOA) captured after compound incubation of MFC7-wt cells. According to Ljosa et al. (Ljosa et al., 2013) only 6 of the 12 MOA were visually identifiable. B- Cell segmentation and feature extraction are performed using CellProfiler (Carpenter et al., 2006). For each cell, a variety of geometric, intensity, subcellular localization and texture features were extracted.

7.2.1 Data splitting

Network architectures, hyper-parameters and training Two deep network algorithms are used, namely Deepnet1 (in the paper) and Deepnet2 (some additional results in the supplementary material):

1. Deepnet1: Stacked autoassociators (SAA)

2. Deepnet2: Stacked denoising autoencoders (SDA)

SDA1 is a variant of the SAA where a corrupted version of the input is used instead during training in order to build a more robust model.

We used hyper-parameters from other models to save computational cost. The values of all the hyper-parameters were selected by performing an informal search on the MNIST, Chars74k and BabyAIshapes dataset taken from our previous models (Kandaswamy et al., 2014b). We did not perform a systematic grid search given the high computational cost, although it is conceivable that even better results could be obtained by systematically tuning the hyper-parameter values.

The classifier is trained on the training set and the validation set is used to periodically control how our model is doing in terms of accuracy on data not used for training, as well as to evaluate early-stopping criteria in the fine-tuning phase to prevent overfitting. The choice of when to stop fine-tuning is based on a geometrically increasing amount of patience. The patience is geometrically increased when the current validation score is below the best validation score. The

Table 7.1: Distribution of MOAs across batches for P_{set1} and P_{set2} with at least one common batch between MOAs. P_{set1} and P_{set2} datasets have 6 mutually exclusive MOAs.

Set Nr.	Mechanism of action	Short name	Number of compounds	Batches	Common Batches
P_{set1}	Actin disruptors	Act	3	01, 02	02, 07,08
P_{set1}	DNA replication	DR	4	02, 08, 09	
P_{set1}	Epithelial	Epi	3	05, 08, 10	
P_{set1}	Kinase inhibitors	KI	3	07	
P_{set1}	Microtubule stabilizers	MS	3	01, 07	
P_{set1}	Protein degradation	PD	4	02, 06, 07	
	Total Nr. of Compounds		20		
	Total Nr. of Treatments		42		
P_{set2}	Aurora kinase inhibitors	Aur	3	01, 03, 04	01, 03, 04
P_{set2}	Cholesterol-lowering	Ch	2	09	
P_{set2}	DNA damage	DD	4	03,04	
P_{set2}	Eg5 inhibitors	Eg5	2	01,03	
P_{set2}	Microtubule destabilizers	MD	4	01,03	
P_{set2}	Protein synthesis	PS	3	03,04	
	Total Nr. of Compounds		18		
	Total Nr. of Treatments		61		

backpropagation error is fine-tuned until it runs out of patience or stops at reaching max fine-tuning epochs. The trained classifier is then tested on the unseen individual cells from the test set and each prediction is matched with their ground-truth of MOA.

The code for reproducing the results, available in the link: `https://github.com/chetakks/DTL`.

High performance computing (HPC) We used HPC machines to perform all our experiments:

1. HPC1: i7-377 (3.50GHz), 16GB RAM with five GTX 980 GPU processors.
2. HPC2: i7-377 (3.50GHz), 16GB RAM with two GTX 770 GPU processors.

We measured the performance of a single thread CPU i7-377 (3.50GHz) with 16GB RAM versus the GTX 770 graphics card. GPU is preferable for large matrices as is the case of the MFC7 data. This allows us to exploit the advantage of parallel computing capability of GPUs.

7.2.2 Layerwise Transfer Learning using Stacked Autoassociators

We consider a Stacked Autoassociators (SAA) (Vincent et al., 2010) to build our classifier of MOA. An autoencoder or autoassociator is a simple neural network with one hidden layer designed to reconstruct its own input. We additionally constrain the encoding and decoding feature sets (input-hidden and hidden-output weights, respectively) to be transpose of each other (tied

weights). SAA training (Vincent et al., 2008) comprises two stages: an unsupervised pre-training stage where the information of the labels (MOA) is not used, followed by a supervised fine-tuning stage, now using the MOA information. In the pre-training stage, a greedy layer-wise approach is used to train the hidden layers of the SAA. The first hidden layer is considered as a regular autoassociator and its features (weights) are trained for several epochs in order to reconstruct the original inputs. After the first layer is pre-trained, we keep only the encoding features and stack a second (hidden) layer over with weights that are trained in a similar way, but now to reconstruct the values. This process is repeated until the k-th hidden layer is pre-trained. In the fine-tuning stage, a logistic regression layer with neurons and weight vector is added to the top of the pre-trained machine and this entire network is fine-tuned using the training subset (now with the labels) in order to minimize a cross-entropy loss function measuring the error between the classifier's predictions and the correct label codes. The optimization process uses a stochastic gradient descent approach of backpropagation using batches of training data to speed up computation time. The learnt features are represented by the weights and biases of the trained SAA. For a SAA with hidden layers is the set of all such parameters. Figure 2 describes these two stages.

To be more precise, let us introduce some notation considering a SAA with 7 hidden layers plus 1 logistic layer, both for the source and target models. We use four different TL settings for supervised layerwise feature transference. In such settings the "0" represents "no transfer", that is, the weights of that specific layer of the target model are randomly initialized and not reused from the source model; the"1" represents "transferred", that is, the initial weights of that specific layer are obtained (reused) from the trained source model. Note that for each setting, the logistic regression layer is also transferred from the source model to the target model. The setting [00111111] means that we randomly initialized the first and second layers of the target model and transferred all the remaining layers from the source problem. The target network thus built is then fine-tuned with the target data.

7.2.3 LOOCV Training and Network Hyper-parameters

Regarding the training process we followed a similar procedure as in Ljosa et al. (Ljosa et al., 2013). To prevent sharing of batch-specific image properties/features or compound properties between the training and test sets and thus to prevent the classifiers to learn artifact properties of the set of individual images rather than the more general cell phenotype (Shamir, 2011), we considered using a leave-one-compound-out cross validation (LOOCV) procedure where all the cells treated with the same compound as the treatment being classified are hold out, even if those other cells were treated with a different concentration. Thus, the test set in LOOCV is composed of all the cells from one of the compounds that is held out; the remaining cells (from all the other compounds) are split in a training set, used to train the model, and a validation set, used to prevent overfitting by evaluating early-stopping criteria in the fine-tuning phase. The choice of when to stop fine-tuning is based on a geometrically increasing amount of patience. The patience is geometrically increased when the current validation score is below the best validation score. The backpropagation error is fine-tuned until it runs out of patience or the maximum fine-tuning epochs

allowed is reached. The trained classifier is then tested on the unseen individual cells from the test set and each prediction is matched with their ground-truth of MOA. The classifier prediction of each cell from the same field of view is then combined to calculate treatment prediction accuracy using majority voting. Each of the experiments is repeated 10 times. Tuning hyper-parameters such as the learning rate or setting the appropriate network architecture for training the deep model is desirable but is highly time consuming. The results of the following section were obtained using SAAs with 7 hidden layers of 500 neurons each. We used pre-training and fine-tuning learning rates of 0.001 and 0.1, respectively. The stopping criteria for pre-training was fixed to 60 epochs, which is the value where the reconstruction cost saturates; stopping criteria for fine-tuning was set to a maximum of 1000 epochs with the validation set. The complete details of these networks are listed in the (Kandaswamy et al., 2016b), Supplementary Table S2. Processing large data as we did, on millions of neural connections, would take several weeks using traditional CPUs. For that reason, we used Theano (Bergstra et al., 2010), a GPU compatible machine learning library, to perform all our experiments on a i7-377 (3.50GHz), 16GB RAM with two GTX 770 and five GTX 980 GPU processors (see (Kandaswamy et al., 2016b), Supplementary material High performance computing section). The code and software to reproduce the results are available at `http://www.deepnets.ineb.up.pt/files/software/DTL_frontend.html`

Table 7.2: Average accuracy in percentage and average computation time in minutes (standard deviation in parenthesis) of the baseline (BL) and DTL approaches. The results are over 10 repetitions for the target data (P_T) with compounds (C) and source data (P_S).

Settings					**Test**		**Time per compound (m)**		**Total time per**
Approach	**Transfer**	P_S	P_T	**C**	**Accuracy**	**p-value (to BL)**	**Pre-train**	**Fine-tune**	**repetition (m)**
BL			P_{set1}	20	84.29(3.21)		8.34(0.0)	16.98(1.3)	506(29)
DTL_1	[00000011]	P_{set2}	P_{set1}	20	**87.62(6.96)**	0.187	-	17.54(2.5)	350(51)
DTL_2	[00001111]	P_{set2}	P_{set1}	20	77.62(8.80)	0.351	-	15.08(1.4)	301(29)
DTL_3	[00111111]	P_{set2}	P_{set1}	20	86.19(8.73)	0.589	-	16.72(2.0)	334(41)
DTL_4	[11111111]	P_{set2}	P_{set1}	20	86.43(3.38)	0.331	-	10.35(0.9)	207(18)
BL			P_{set2}	18	87.05(4.25)		12.71(0.2)	26.10(1.8)	698(37)
DTL_1	[00000011]	P_{set1}	P_{set2}	18	**87.87(6.86)**	0.734	-	27.36(2.3)	492(42)
DTL_2	[00001111]	P_{set1}	P_{set2}	18	69.67(11.4)	<0.001	-	21.39(2.7)	385(49)
DTL_3	[00111111]	P_{set1}	P_{set2}	18	85.08(6.99)	0.513	-	25.33(2.8)	455(50)
DTL_4	[11111111]	P_{set1}	P_{set2}	18	75.57(4.72)	<0.001	-	19.79(2.2)	356(41)

Table 7.3: Comparison of accuracy obtained and total time taken per repetition in minutes with other state-of-the-art methods.

Method	P_{set1}		P_{set2}	
	Accuracy (%)	Time (m)	Accuracy (%)	Time (m)
Linear SVM	20.95	32	23.49	49
SVM using RBF (model trained using 1% of total training data)	21.04	78	17.5	125
8 layers deep architecture (Baseline)	84.29	506	87.05	698
DTL_1 [00000011]	**87.62**	350	**87.87**	492

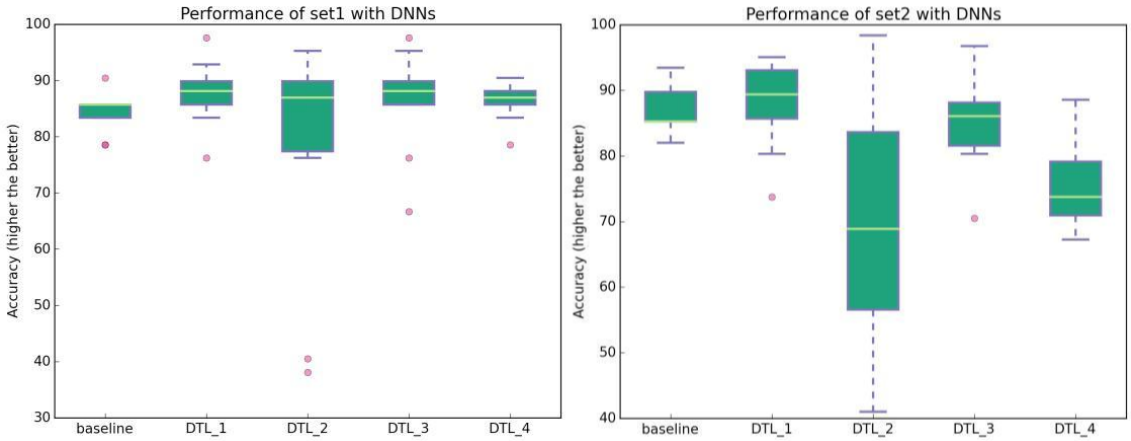


Figure 7.2: Comparison of Baseline versus DTL approaches. Left: Baseline average accuracy for classifying P_{set1} and DTL approaches for classifying P_{set1} reusing P_{set2}. Right: Baseline average accuracy for classifying P_{set2} and DTL approaches for classifying P_{set2} reusing P_{set1}.

7.3 Results

The analysis of large volumes of multiparametric high-dimensional data without overfitting the network using a high number of cytological features in a time frame suitable for drug discovery presents a significant challenge for any learning algorithm. In the following we present the results obtained by our approach. The results of the baseline SAA for classifying MOAs for P_{set1} and P_{set2} datasets are listed in Table 7.2. We observe that classifying MOAs of P_{set2} is about 2.8% more accurate than classifying MOAs of P_{set1}, even though both datasets have an equal number of MOAs. Also, the computation time to classify P_{set2} dataset is greater than that of P_{set1} dataset. The P_{set2} dataset has 61 treatments for 18 compounds, whereas P_{set1} has 42 treatments for 20 compounds. The confusion matrix for classifying MOAs using the baseline approach for both P_{set1} and P_{set2} datasets is shown in Fig. 7.3 and the precision, recall and f1-score are listed in (Kandaswamy et al., 2016b), Supplementary Table S3.

To further improve the results over the baseline approach, we considered a deep transfer learning framework where the knowledge gained with the source problem is reused to solve the target problem. The results for four DTL settings are presented in Table 7.2 and the respective boxplots displayed in Fig. 7.2. Essentially, we observe that the DTL_1 setting improves over the baseline for both P_{set1} and P_{set2} datasets. It is interesting to note that the best results are obtained when such specific (top) layer weights are transferred from the source to the target problem (the 7th hidden layer weights and the logistic regression weights are reused) and the rest of the (lower) layers are randomly initialized. For example, classifying P_{set1} reusing P_{set2} with the DTL_1 transfer setting, produces models 2% more accurate than the baseline and about 0.8% over the transfer all case DTL_4. One of the reasons for this behavior is that higher layers of the network learn problem-specific features from the data while the lower layers learn generic features (Kandaswamy et al., 2014b), (Yosinski et al., 2014), thus it seems beneficial to use the knowledge acquired in the source problem on its higher layers. Moreover, the DTL_1 setting speeds up computation time by 30%

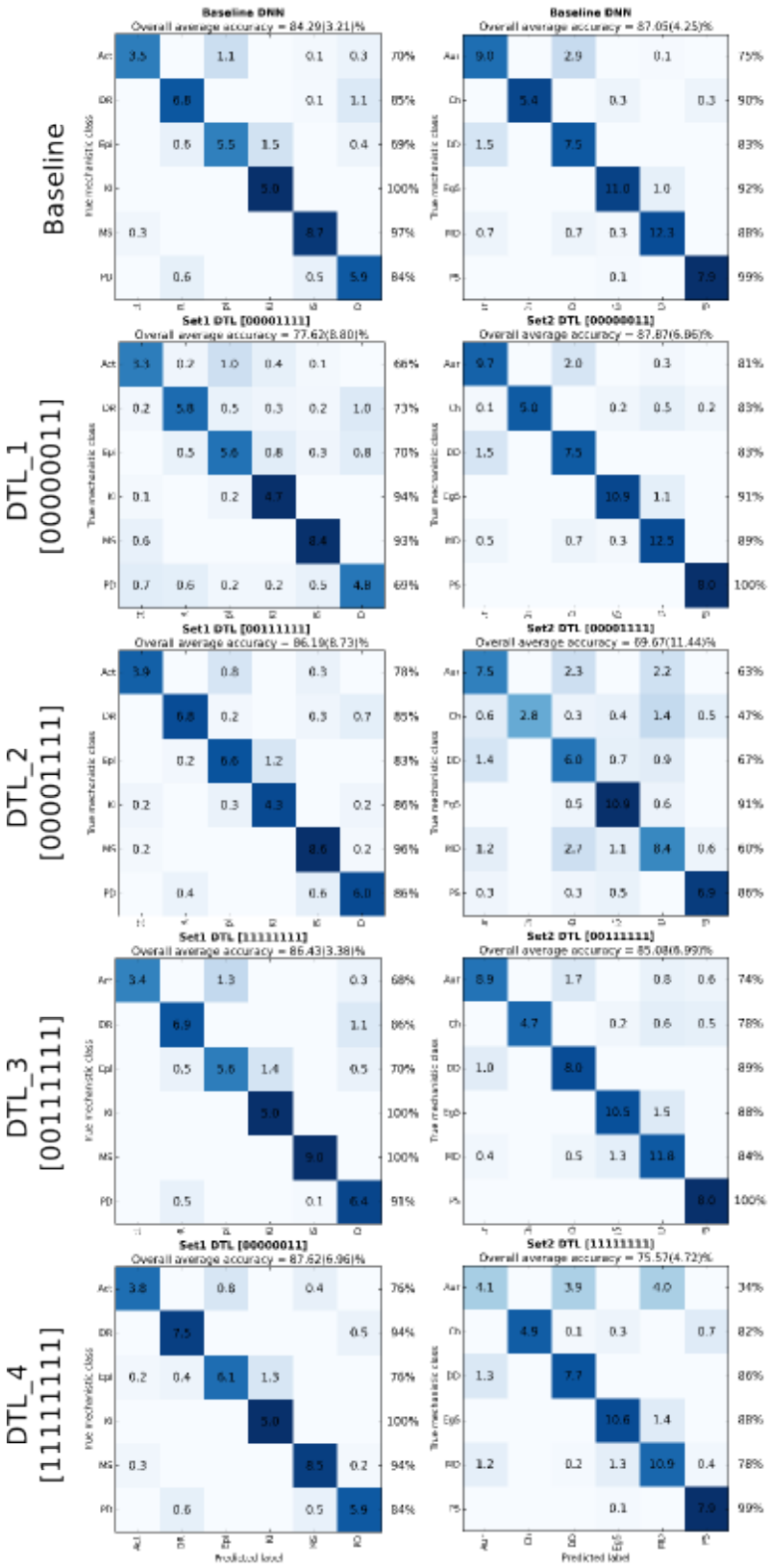


Figure 7.3: Confusion matrices for the baseline and TL settings on the MOA problem (average outcomes over 10 repetitions).

over the baseline approach. Confusion matrices for all DTL settings can be analyzed in Fig. 7.3. To represent class imbalance the confusion matrix represent number of elements in each class and the background blue color is normalized confusion matrices (higher the accuracy darker the color). Given these results, we believe that DTL_1 would be a good setting to use on similar problems by a researcher who wishes to use DTL on this type of problem.

7.3.1 Comparison with other state-of-the-art methods

Table 7.3 lists a comparison of our deep learning (Baseline and best TL setting) results with two state-of-the-art machine learning algorithms; Support Vector Machines (SVM) (Cortes and Vapnik, 1995) using linear and radial basis function (RBF) kernel using a freely available and fast C-based implementation of multi-class SVM (SVM^{multiclass}, version 2.20). For linear SVM we optimized the trade-off between training error and margin cost from 0.001 to 50000 (see (Kandaswamy et al., 2016b), Supplementary Table S4) and the best model obtained an overall accuracy of about 21% for P_{set1} and 23% for P_{set2} (see (Kandaswamy et al., 2016b), Supplementary Tables S7 and S8). For SVM RBF we optimized the margin cost from 1 to 1000 and the gamma parameter from 0.001 to 0.00001 (see (Kandaswamy et al., 2016b), Supplementary Table S5). As the grid search is computationally expensive, we restricted to only one compound using 10% of the total training data. We observed the best model at margin cost 100 and gamma 0.001 but taking between 419 to 755 minutes to obtain a 45% accuracy (see (Kandaswamy et al., 2016b), Supplementary Table S6). Thus we performed the experiments with 1% of the total training data to train the SVM RBF and obtained an overall accuracy of about 21% for and 18% for (see (Kandaswamy et al., 2016b), Supplementary Tables S9 and S10). Further increasing the number of training samples improves the overall accuracy but leads to an exponential increase in computation time.

7.4 Conclusion

To stimulate the development of new drugs effective against a wide spectrum of cancers, we propose a Deep Transfer Learning (DTL) classifying framework that uses high-content HCA data. Our classifiers are built upon individual cell information without employing any type of profiling or reduction methods on extracted cell features. The main motivation to use a DTL approach was to show that we can reuse, with minor modifications, the knowledge acquired in solving a classification of MOA from one cell line to solve a new classification of MOA from another cell line *without having to follow the whole training procedure*. This is particularly useful for new drug testing as computational time is saved. For that purpose, the data was carefully split into two mutually exclusive 6-class problems represented by P_{set1} and P_{set2} datasets. The average accuracies of the baseline SAA for P_{set1} and P_{set2} datasets are about 84% and 87%, respectively, using a 7 hidden layer SAA with 500 neurons in each layer. The DTL approach showed that the transference of specific weights of the source model was useful and we have obtained positive transference for both the data sets. Although the difference in accuracy of P_{set1} and P_{set2}, between Baseline and Transfer learning is not statistically significant, we observed around 30% computational speed up,

when using the DTL approach. Our approach was also superior when compared to multi-class Support Vector Machines.

Regarding the 12-class problem we trained several SAAs ranging from 3 to 8 hidden layers with 500 to 1000 neurons in each layer. However, training a 7 hidden layer SAA with 500 neurons in each layer may take, on average, 30 to 48 hours per repetition. We performed some preliminary experiments using the adequate leave-one-out approach and, without too much hyperparameter search, the best model obtained around 77% accuracy. As future work we intend to explore a different approach for the 12-class problem using Convolutional Neural Networks (CNN) directly applied to the images and not to hand-crafted features. CNNs are state-of-the-art deep neural networks that use a sort of hierarchical representation of the data similar to that of the neocortex and are especially designed for image recognition tasks. We expect to obtain a similar hierarchical feature extraction directly from the images, giving the possibility of the deep network to self-extracting relevant cytological features layer-by-layer.

Chapter 8

Cross-sensor Biometrics [1]

In this Chapter, we work on biometric recognition problem with multiple sensor scenario. In information technology, biometrics refers to the quantitative measure and analysis of human anatomical or behavioural characteristics, such as Deoxyribonucleic acid (DNA), fingerprints, eye retinas and irises, voice patterns, facial patterns and hand measurements, for authentication purposes.

With the increasing popularity and availability of mobile devices, capable of performing the whole biometric recognition framework, from data acquisition to the final decision, a new obstacle is presented to the development of such systems: the need to adapt to the wide variety of available sensors and their respective heterogeneity with regards to image quality. The question of whether or not sensors from different manufacturers show a high degree of interoperability allowing, for example, for an individual to be enrolled in a single system and then be successfully recognized in a vast variety of alternative devices, is of growing importance in the research field of biometrics. With this formulation in mind, it is trivial to understand how the principles of transfer learning may be adopted for this rising challenge.

With recent studies showing that cross-sensor matching, where the test instances are verified using data enrolled with a different sensor, often lead to reduced performance, we attempt to overcome this challenge by making use of transfer learning principles and, thus, achieve state-of-the-art performance for a large variety of acquisition scenarios. For that purpose we choose, from the vast array of transfer learning approaches, to explore and extend the Source-Target-Source (STS) approach, first proposed in (Kandaswamy et al., 2015b), while applying it to the specific challenge of cross-sensor periocular recognition. STS is a recent alternative that has shown both increased performance with object and computer vision recognition tasks, as well as an gain in processing speed.

The practical problem of *cross-sensor* biometrics has also been the focus of many works in recent years, highly motivated by the growing variety and availablity of mobile sensors. The most commonly found works concern mostly iris recognition. Connaughton et al. (Connaughton et al., 2011) performed a comparison between three commercially available iris cameras, with the aim of assessing the interoperability between them and the impact of some state-of-the-art

[1] Some parts of this chapter are used from article (Kandaswamy et al., 2016a)

recognition algorithms in both single as well as cross-sensor scenarios. The authors arrived at some conclusions, namely the fact that the relative performance of a given algorithm in a variety of single-sensor scenarios does not relate reliably to the performance of the same algorithm when tested in cross-sensor scenarios. Furthermore, performance observed for all cross-sensor scenarios was consistently worse than their single-sensor counterparts.

Another recent work on the field of iris recognition, proposed by Pillai et al. (Pillai et al., 2014), attempted to adapt iris instances acquired with one sensor to the characteristics of a new sensor, in an attempt to mitigate the performance-drop commonly observed in cross-sensor scenarios. Both Santos et al. (Santos et al., 2015) and Jilela and Ross (Jillela and Ross, 2014) propose methods based on information extracted from the periocular region. While Santos et al. propose a framework based on multiple descriptors to work on periocular data on multiple mobile sensors, Jilela and Ross attempt to match iris and face images from the same individual, acquired with distinct sensors, using periocular traits to help in the recognition process. With the marked advantages of periocular recognition over its iris and face counterparts becoming more widely accepted and researched, especially when unconstrained acquisition settings are considered, the present work will focus on exploring transfer learning alternatives to periocular recognition in order to attenuate the problems commonly associated with cross-sensor scenarios.

8.1 Cross-Sensor Recognition

In the present work we explore the approaches outlined in the previous section as an alternative to tackle the cross-sensor biometric recognition problem. This problem can be understood as the problem of successfully performing biometric recognition on a specific image acquisition device without the need of performing a new enrollment phase for the new device specifically. This interpretation can be easily extrapolated to the domain of the aforementioned approaches if both devices are understood as the target (where recognition is to be performed) and the source (where enrollment was carried out). In the following sections we outline the experimental setups designed to assess the performance of the proposed methodologies in the specific practical problem of cross-sensor periocular recognition.

The periocular region is commonly described as the region in the immediate vicinity of the eye. Periocular recognition can be motivated as a representation in between face and iris recognition. It has been shown to present increased performance when only degraded facial data or low quality iris images are made available to the recognition system.

We start by detailing a baseline algorithm, first proposed by Monteiro et al. (Monteiro and Cardoso, 2015), that has presented state-of-the-art performance for multiple single-sensor scenarios, as well as a commonly used feature representation technique - Gaussian Mixture Models (GMM) supervectors - which will be explored for SDA approaches. We then present the experimental setup under which each of the tested methodologies was assessed as well as the performance

metrics chosen for such process. Finally, we present the most significant results as well as a detailed discussion concerning the relative performance of each method for each of the proposed challenges.

8.1.1 GMM-Universal Background Model (GMM-UBM)

The GMM-UBM algorithm for periocular recognition, first proposed by Monteiro et al. (Monteiro and Cardoso, 2015), is schematically represented in Figure 8.1. During the enrollment, a set of G models describing the unique statistical distribution of biometric features for each individual $g \in \{1, ..., G\}$ is trained by maximum *a posteriori* (MAP) adaptation of an Universal Background Model (UBM). The UBM is a representation of the variability that the chosen biometric trait presents in the universe of all individuals. MAP adaptation works as a specialization of the UBM based on each individual's biometric data. The idea of MAP adaptation of the UBM was first proposed by Reynolds (Reynolds et al., 2000), for speaker verification. The tuning of the UBM parameters in a maximum *a posteriori* sense, using individual specific biometric data, provides a tight coupling between the individual models and the UBM, resulting in better performance and faster scoring than uncoupled methods, as well as a robust and precise parameter estimation, even when only a small amount of data is available.

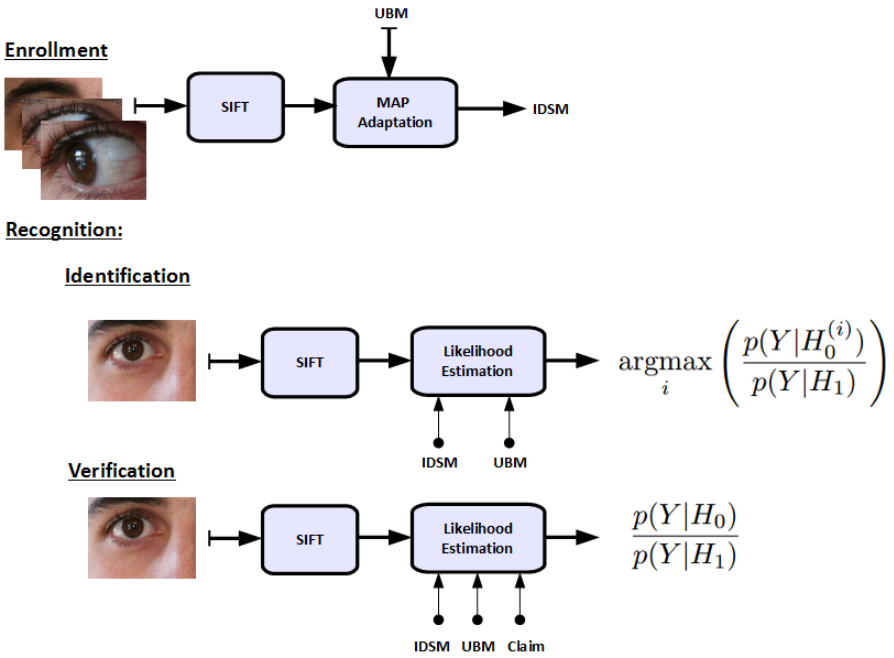


Figure 8.1: Schematic representation of the GMM-UBM periocular recognition algorithm proposed by Monteiro et al. (Monteiro and Cardoso, 2015).

The recognition stage is carried out through the projection of the features extracted from an unknown sample onto both the UBM and the individual specific models (IDSM) of interest. A likelihood-ratio between both projections outputs the final recognition score. Depending on the functioning mode of the system - verification or identification - decision is carried out by thresholding or maximum likelihood-ratio respectively. The use of a likelihood-ratio score with an

universal reference works as a normalization step, mapping the likelihood values in accordance to their global projection. Without such step, finding a global optimal value for the decision threshold would be a far more complex process.

Gaussian Mixture Models (GMM) were chosen to model both the UBM and the individual specific models (IDSM). From the most common interpretations, GMMs are seen as capable of representing broad "hidden" classes, reflective of the unique structural arrangements observed in the analysed biometric. The original work was proposed using SIFT keypoint descriptors as the only features, but a more recent version (Monteiro et al., 2015) proposed a score-level fusion of multiple descriptors (SIFT, HOG, Local Binary Pattern (LBP) and GIST), resulting in improved performance.

The original work was designed with single-sensor recognition in mind, i.e. the source and target data are the same. In the present work we also assess the performance in cross-sensor scenarios, where training of models and classification are carried out on distinct data sources. Some preliminary results for such setup have already been reported in a follow-up work by the original authors (Monteiro et al., 2015). The present work will more thoroughly analyse and discuss such results, as well as presenting a comparative analysis with alternative approaches.

8.1.2 GMM Supervectors (SV-SDA)

In the previous section, recognition was carried out through a likelihood ratio between a target IDSM and the UBM. Recently, a significant amount of works have explored the use of an alternative GMM representation - GMM supervectors - as the input for classification algorithms, with some promising results being reported in the literature (Campbell et al., 2006). Super-vector notation consists on concatenating in a single vector all the parameters describing a GMM (weights, means and covariance matrices). For example, the mean values of the UBM can be concatenated to form a single mean super-vector, m, given by $m = [\mu_{T1}, \mu_{T2}, ..., \mu_{Tk}]$, where k is the total number of mixtures in the UBM (Ge et al., 2015). A similar representation can be extracted for the IDSM parameters or even for single images. On the present work we describe each training image t belonging to subject i, $Im_{t,i}$, by its supervector representation, obtained by MAP adaptation of the UBM parameters using the feature data extracted solely from $Im_{t,i}$. SIFT keypoint descriptors are used for feature description and model training, as proposed by Monteiro et al. (Monteiro and Cardoso, 2015). We then perform training, validation and classification using the SDA methodology for both TLs and STS approaches, as detailed in Chapter 4, Section 4.1.2.

8.1.3 CNN

The CNN methodology for both TLs, STS and MSTS approaches was also carried out, as described in Section 3.1.2, using raw pixel intensity values. We use three main types of layers to build CNN architectures: Convolutional Layer (conv), Pooling Layer (pool), and Fully-Connected Layer (FC). We will stack these layers to form a full CNN architecture with a logistic regression classifier (LR). Architecture of our 5 layer CNN model has [Conv - Pool] x 3 -FC -LR. We first

crop the image into 200 by 120 and then convert the image to greyscale which is presented as the input. This is convolved with 30 different 1st layer filters, each of size 12 by 12, using a stride of 1 in both x and y. The resulting feature maps are then pooled in (max within 2x2 regions, using stride 1) to give 30 different 94 by 54 element feature maps. Similar operations are repeated with 60 and 90 different layer filters in 2nd and 3rd layers respectively. The 4th layer is fully connected, taking features from the top convolutional layer as input in vector form. The final layer is a c-way logistic regression classifier, c being the number of classes. All filters and feature maps are square in shape.

8.2 Cross-sensor dataset

The methodologies outlined in the previous sections were assessed on the Cross-Sensor Iris and Periocular (CSIP) dataset. The CSIP database, created for the assessment of the algorithm proposed by Santos et al. (Santos et al., 2015), is a recent and publicly available dataset, designed with the main goal of gathering periocular images from a representative group of participants, acquired using a variety of mobile sensors under a set of variable acquisition conditions. Given the heterogeneity of the camera sensors and lens setups of consumer mobile devices, 10 different setups were used during the dataset acquisition stage: four different devices, some of which had both frontal and rear cameras, and LED flash. This variety of sensors confers a strong appeal to the CSIP database regarding its potential use for the assessment of algorithms under a highly heterogeneous set of conditions. A summary of the details concerning each of such setups may be observed in Table 8.1, while a visual example of an image for each subset of the same individual is depicted in Figure 8.2. Each participant was imaged using all of the presented setups.

Table 8.1: Technical details concerning the acquisitions setups used for each subset of the CSIP database.

Setup ID	AR0	AR1	BF0	BR0	BR1	CF0	CR0	CR1	DF0	DR0
Device	*A*		*B*			*C*			*D*	
Manufacturer	Sony Ericsson		Apple			ThL			Huawei	
Model	Xperia Arc S		iPhone 4			W200			U8510	
O.S.	Android 2.3.4		iOS 7.1			Android 4.2.1			Android 4.3.3	
Camera	Rear		Frontal	Rear		Frontal	Rear		Frontal	Rear
Resolution	3264×2448		640×480	2592×1936		2592×1920	3264×2448		640×480	2048×1536
Flash	No	Yes	No	No	Yes	No	No	Yes	No	No

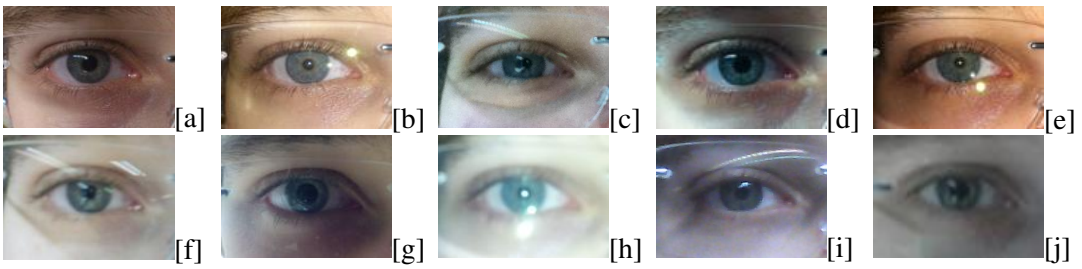

[a] [b] [c] [d] [e] [f] [g] [h] [i] [j]

Figure 8.2: Examples of images from each subset of the CSIP database. From (a-j) respectively: $AR0$, $AR1$, $BF0$, $BR0$, $BR1$, $CF0$, $CR0$, $CR1$, $DF0$ and $DR0$.

To simulate the variable noise associated with on-the-go recognition, participants were not imaged at a single location, but instead they were enrolled at multiple sites with artificial, natural, and mixed illumination conditions. In total, 50 participants were enrolled, all Caucasian and mostly males (82%), with ages ranging between 21 and 62 years ($mean \pm std = 31.18 \pm 9.93$ years). For each periocular image acquired by the mobile devices, a binary iris segmentation mask was also produced. The masks were obtained automatically using the state-of-the-art iris segmentation approach proposed by Tan et al. (Tan et al., 2010), which is particularly suitable for uncontrolled acquisition conditions, as demonstrated by its first place ranking at Noisy Iris Challenge Evaluation - Part 1 (NICE.I) (Proença and Alexandre, 2007).

8.2.1 Image pre-processing

Images from the CSIP database were converted to grayscale and re-sized so as to present a fixed number of pixels, necessary for the implementation of all the approaches based on the CNN methodology. Resizing was carried out in such a way that geometrical proportions were kept from the original images.

8.2.2 Data partitioning

In order to achieve a fair and meaningful comparison between the tested methodologies, a common experimental setup was designed. The set of all images of the CSIP dataset was divided as follows: 50% of the images per individual and per subset were kept for model training, 25% were chosen for validation of the trained models and the remaining 25% were used to assess performance. Train, validation and test subsets were randomly selected and all experiments were cross-validated 10 times.

8.2.3 Evaluation metrics

Performance was evaluated only for identification problems, where, given a biometric sample from an unknown source, the e most probable identities are assessed. For such problems the most commonly found performance metric is the *rank-1 recognition rate*, which refers to the ratio of correctly assessed identities, when $e = 1$.

8.3 Cross-sensor recognition performance

The main results obtained for the experimental setups detailed in the last section are summarized in Tables 8.2 through 8.8. Discussion of these results will be carried out, from this point onwards, starting with the BL and TLs approaches, followed by an analysis on how the STS strategy may improve performance in cross-sensor scenarios and, finally, on the effect that multiple sources of information may present.

8.3.1 Baseline and Transfer Learning

The baseline results for each tested methodology (GMM-UBM, SV-SDA and CNN) are presented in the diagonal values of Tables 8.2, 8.3 and 8.4 respectively. By the sole analysis of these results some conclusions may already be drawn. First of all it is easily discernible how the GMM-UBM methodology, specifically designed to solve the single-sensor periocular recognition problem, outperforms both alternatives in such conditions. Even for the CSIP subsets that, in theory, offer the least challenging conditions (*AR*1, *BR*1 and *CR*1), the performance drop observed is non-negligible. Taking *AR*1 as a specific example a relative performance drop of 26.6% and 18.3% is observed against the SV-SDA and CNN methodologies respectively. This effect is, however, reversed when cross-sensor scenarios are taken into consideration.

If we consider a single target dataset, it is readily observable that the variance in performance in a lot less pronounced for the CNN and SV-SDA methodologies than for GMM-UBM. Furthermore, it also notorious how the significantly better single-sensor scenario results of the GMM-UBM are severely degraded when a more complex challenge is presented to the algorithm. A trivial conclusion can be taken from such observations: even though the GMM-UBM presents the best baseline results, as expected from an algorithm tailored for that specific challenge, the application of transfer learning to both CNN and SV-SDA methodologies results in a considerably lower variance in the performance values observed for a single target dataset, regardless of the chosen source. A valid deduction, following such conclusions, is that improving the baseline performance of such methodologies will also result in an increased cross-sensor performance. Given that the challenge of cross-sensor biometric recognition is mostly concerned with the performance loss observed in such cases, the global behaviour of the tested methodologies seems to, at least, motivate further research seeking to improve the baseline performance.

In the next section we explore the effect that the source-target-source approach presents over the simpler TLs alternative.

Table 8.2: Rank-1 recognition rates, in %, observed for the GMM-UBM algorithm for all possible cross-sensor scenarios in the CSIP database.

		Target								
		AR0	**AR1**	**BF0**	**BR0**	**BR1**	**CF0**	**CR0**	**CR1**	**DR0**
Source	**AR0**	**94.4**	57.8	34.1	76.8	45.9	41.8	52.8	40.7	65.6
	AR1	64.6	**97.1**	26.5	47.9	83.6	25.5	35.4	66.8	33.7
	BF0	33.1	23.1	**78.2**	21.1	19.4	30.8	24.0	19.8	16.0
	BR0	67.4	39.6	19.7	**92.4**	54.2	36.5	42.3	34.3	67.7
	BR1	31.8	62.3	12.0	48.1	**95.5**	28.3	25.6	52.5	35.3
	CF0	36.4	29.1	34.7	36.5	30.8	**89.8**	55.8	39.8	46.3
	CR0	59.5	30.2	24.4	58.1	36.4	59.3	**80.3**	45.7	71.9
	CR1	42.6	64.9	21.2	47.8	70.8	47.5	50.5	**90.0**	49.0
	DR0	41.3	18.0	17.4	53.0	23.1	30.3	39.8	24.8	**88.7**

Table 8.3: Rank-1 recognition rates, in %, observed for the SV-SDA algorithm for all possible cross-sensor scenarios in the CSIP database.

	Target									
	AR0	**AR1**	**BF0**	**BR0**	**BR1**	**CF0**	**CR0**	**CR1**	**DR0**	
AR0	38.0	**81.8**	39.3	**33.4**	79.3	14.1	**27.7**	52.5	20.0	
AR1	35.4	76.7	22.7	24.6	77.3	15.9	20.9	**61.6**	17.6	
BF0	**43.8**	81.4	**41.4**	27.3	80.1	**22.3**	17.8	59.7	21.5	
BR0	35.2	79.0	35.8	25.6	77.3	14.6	21.8	45.9	22.1	
BR1	32.4	78.4	37.7	25.1	**82.1**	22.1	18.0	55.6	23.0	*Source*
CF0	32.4	79.8	36.4	24.1	75.3	12.1	16.3	50.6	**24.6**	
CR0	41.5	79.0	36.6	24.8	81.6	16.1	23.5	51.7	23.9	
CR1	28.7	79.0	36.6	22.1	78.0	16.4	18.2	57.9	22.4	
DR0	28.9	80.4	39.0	29.0	72.2	15.0	22.0	52.5	19.7	

Table 8.4: Rank-1 recognition rates, in %, observed for the CNN algorithm for all possible cross-sensor scenarios in the CSIP database.

	Target									
	AR0	**AR1**	**BF0**	**BR0**	**BR1**	**CF0**	**CR0**	**CR1**	**DR0**	
AR0	61.5	84.2	**54.0**	**69.0**	91.0	**64.4**	63.9	92.0	**65.3**	
AR1	60.8	82.1	52.0	66.0	88.5	58.1	63.9	92.0	56.7	
BF0	**65.9**	84.5	50.0	68.5	89.5	62.6	67.4	92.3	**65.3**	
BR0	63.0	82.6	52.4	63.5	88.0	61.8	67.0	92.0	56.7	
BR1	63.0	85.0	52.4	60.5	85.5	60.4	64.3	90.3	57.3	*Source*
CF0	64.4	85.0	50.8	63.5	**91.5**	54.8	66.1	93.0	58.0	
CR0	62.2	**86.8**	51.2	66.5	90.5	60.7	67.9	**93.3**	62.7	
CR1	61.4	82.9	50.4	66.5	89.5	57.8	65.2	88.0	57.3	
DR0	60.0	83.2	53.2	62.0	86.5	55.2	**68.7**	90.0	53.3	

8.3.2 Source-target-source

As detailed in (Kandaswamy et al., 2015b), we propose a cyclic source-target-source (STS) approach for classification using the CNN and SDA methodologies. Tables 8.7 and 8.8 present the STS results observed for the CNN methodology for a single cycle (STS_1) and for a total of 10 cycles, respectively. Analogous results for the SDA methodology may be observed in Tables 8.6 and 8.5. For a simpler analysis the baseline results are kept on the diagonal of each table, as in the last section.

The first observation to be taken from the analysis of the aforementioned tables is how even a single cycle of STS can significantly improve some of the baseline results. The $CF0$ baseline for example, presents a relative improvement of 18.2% for the CNN methodology, and most of the observed results already exceed those observed for the simpler TLs approach. This improvement is even more discernible when multiple STS cycles are carried out. The results presented in Tables 8.5 and 8.8 depict this behaviour. Here, and taking the same $CF0$ baseline result as referred before, the performance, comparing to the baseline, is increased to 29.0%. It is interesting to note how the stability observed in the last section, when a single target dataset is considered, is also observed in this approach, with the addition of significantly increased performance. The same conclusion can, thus, be achieved: if a stronger baseline performance is achieved, STS approaches to classification seem to present the capability of both improving the baseline performance, as well as guaranteeing the maintenance of such performance when different acquisition scenarios are considered.

Another consideration to be taken from the analysis of these results is how significantly worse the results using supervector-based SDA classification are when compared with their CNN counterparts. This observation can also be made from the analysis of Tables 8.2, 8.3 and 8.4 from last section. The most obvious explanation concerns the fact that the supervector representation based on the GMM modelling of SIFT keypoint descriptors might not present enough discriminative information for accurate SDA classification, except in some specific cases. For example, the datasets composed by higher quality images ($AR1$ and $BR1$) present considerably better performance, even surpassing the performance of their CNN counterparts. These results show that some discriminative power exists, even though it seems severely compromised when the quality of the input images decreases. Regardless of that, the STS behaviours described above still remain relevant for the SDA methodology, and may earn some further research regarding the use of more adequate feature representation techniques.

As a final approach we also explored the effect of using information from multiple sources in order to improve the performance of the cyclic STS algorithm. The main results and observations regarding this approach will be outlined in the next section, in an attempt to summarize all the results and observations obtained in the present work.

Table 8.5: Rank-1 recognition rates, in %, observed for the SDA methodology and the STS approach.

	Target									
	AR0	**AR1**	**BF0**	**BR0**	**BR1**	**CF0**	**CR0**	**CR1**	**DR0**	
AR0	38.0	91.3	**54.4**	**38.7**	90.6	19.0	**30.9**	59.6	**32.7**	
AR1	40.8	76.7	42.1	29.7	93.9	**19.8**	25.4	**71.6**	32.0	
BF0	**48.5**	**91.6**	41.4	32.2	90.8	**19.8**	23.5	65.9	31.7	
BR0	43.1	88.7	48.2	25.5	90.6	17.3	27.9	62.5	31.7	
BR1	43.1	90.0	50.3	29.5	82.1	19.0	26.1	64.3	30.3	*Source*
CF0	40.0	88.0	47.4	33.8	89.4	12.0	26.1	63.9	28.7	
CR0	44.4	88.0	50.9	32.2	**93.1**	**19.8**	23.5	66.1	29.3	
CR1	37.4	90.0	50.6	28.9	92.2	18.3	25.1	57.8	30.0	
DR0	43.3	88.4	53.2	31.9	90.0	16.8	29.1	61.6	19.7	

Table 8.6: Rank-1 recognition rates, in %, observed for the SV-SDA methodology and a single cycle of the STS approach.

	Target									
	AR0	**AR1**	**BF0**	**BR0**	**BR1**	**CF0**	**CR0**	**CR1**	**DR0**	
AR0	38.0	89.1	**54.4**	**31.1**	90.6	16.0	**27.7**	59.6	27.3	
AR1	40.8	76.7	42.1	29.2	92.2	**17.0**	25.4	**68.9**	**32.0**	
BF0	42.3	**91.6**	41.4	30.3	90.8	16.5	22.8	63.9	29.3	
BR0	**43.1**	88.7	48.2	25.5	90.6	14.0	27.9	62.5	25.7	
BR1	41.8	90.0	50.3	28.4	82.1	16.8	23.5	64.1	26.0	*Source*
CF0	39.2	88.0	47.4	30.0	89.4	12.0	25.4	63.9	25.0	
CR0	42.6	88.0	50.9	30.0	**93.1**	16.5	23.5	66.1	26.7	
CR1	37.4	89.6	50.0	28.9	90.0	18.3	25.1	57.8	26.0	
DR0	41.8	88.4	53.2	30.2	90.0	16.8	25.6	61.6	19.7	

Table 8.7: Rank-1 recognition rates, in %, observed for the CNN methodology and a single cycle of the STS approach.

	Target									
	AR0	**AR1**	**BF0**	**BR0**	**BR1**	**CF0**	**CR0**	**CR1**	**DR0**	
AR0	61.5	84.5	**55.2**	**70.0**	87.5	63.0	68.3	92.3	59.3	
AR1	64.4	82.1	50.4	66.5	88.5	59.3	68.7	91.0	60.0	
BF0	63.7	84.0	50.0	67.5	88.0	**64.8**	66.1	92.0	57.3	
BR0	62.2	84.0	54.0	63.5	**89.5**	64.1	67.0	92.0	61.3	
BR1	64.4	84.2	52.4	65.0	85.5	60.4	**70.9**	**93.3**	54.7	*Source*
CF0	61.5	84.5	53.6	64.0	88.0	54.8	69.6	90.7	62.0	
CR0	63.3	84.5	52.0	67.5	88.5	63.0	67.9	90.0	62.0	
CR1	**65.2**	85.3	52.8	68.5	86.5	64.1	68.7	88.0	**64.0**	
DR0	63.7	**85.5**	50.4	65.5	89.0	62.6	64.8	91.7	53.3	

Table 8.8: Rank-1 recognition rates, in %, observed for the CNN methodology and STS approach.

	Target									
	AR0	**AR1**	**BF0**	**BR0**	**BR1**	**CF0**	**CR0**	**CR1**	**DR0**	
AR0	61.5	88.7	**58.4**	72.5	91.0	70.0	67.8	92.3	70.0	
AR1	71.9	82.1	55.2	69.5	**92.5**	66.7	70.9	92.0	66.7	
BF0	**73.3**	90.3	50.0	71.5	91.5	**70.7**	69.1	92.0	71.3	
BR0	72.6	89.0	57.6	63.5	90.5	69.6	70.4	92.7	**72.0**	
BR1	71.5	90.8	54.8	69.0	85.5	64.4	**72.2**	**93.3**	64.7	*Source*
CF0	72.6	90.5	57.2	72.5	91.5	54.8	70.0	92.3	69.3	
CR0	72.2	90.8	**58.4**	**73.5**	91.0	68.9	67.9	92.3	69.3	
CR1	70.7	**92.1**	56.4	72.5	90.5	65.9	70.4	88.0	68.0	
DR0	71.1	89.7	57.2	70.0	91.0	67.0	68.7	92.3	53.3	

8.3.3 Multiple Source STS

Figure 8.3 summarizes both the results obtained for the STS algorithm using multiple sources (MS-STS) as well as all the most relevant results presented in the last sections. The main goal of MS-STS is to achieve a high degree of domain generalization, in order to allow the trained classifiers to perform well for the widest possible variety of scenarios. For the multiple source examples we chose the flash subsets (*AR*1, *BR*1 and *CR*1) as the sources and all other no-flash datasets as the targets. This choice can be motivated by the fact that the three flash datasets consistently presented the best absolute performance among all the experiments that we carried out. Such observation seems to indicate that the intrinsic discriminative power of such datasets might be higher than the remaining alternatives, thus conferring them, at least in theory, a marked advantage as choice for source datasets. We also chose to work only with the CNN methodology, as the vast majority of the results observed in the last section seemed to point towards its better fit for the problem at hand.

So as to better visualize and understand the effect of the MS-STS approach over the approaches presented in the last sections we decided to present the results in the radial plot representation that can be observed in the 6 images from Figure 8.3. For each image a series of features can be observed:

- **Source and Target Datasets:** Each of the axis of the radial plot represents the rank-1 recognition rate (in %) for the chosen target (positive vertical axis) as well all the three source datasets.

- **BL, TLs, STS_1 and STS:** The BL label represents the baseline performance as already presented in the diagonal values of Tables 8.2 to 8.4. TLs and STS_1/STS, on the other hand, represent the best results for each of the 4 depicted datasets (3 sources and 1 target), for each of their individual TLs and STS_1/STS experiments (bold values in Tables 8.3 to 8.8). STS_1 represents the performance after a single cycle of the STS approach, whereas STS_{10} refers to the best performance observed after 10 cycles.

- **STS (3 sources):** This label depicts the best performance observed for the target dataset, for its individual STS_1/STS experiment, considering only $AR1$, $BR1$ and $CR1$ as a possible source. We chose to include this label in order to achieve the fairest possible comparison between the MS-STS performance and the optimal single-source experiment.

- **MS-STS_1 and MS-STS:** The two polygons, in blue and red respectively, represent the first cycle and optimal performances, after 10 repetitions of the whole multi-source cyclic process, for each of the source and target datasets.

By the analysis of the plots, independent of the chosen target dataset, a few interesting conclusions can be drawn. First, there seems to be no significant performance change, regarding the target dataset, between the MS-STS (after 10 cycles) approach and the analogous results for STS using only the best single source from the $[AR1, BR1, CR1]$ set of sources. What the MS-STS offers is a way of achieving this optimal performance without the need of an empirical choice of the best source subset, thus conferring a more robust nature to the whole process. This is also the main advantage of the multiple source approach when compared to the optimal STS performance obtained when considering all 8 possible sources for a specific target: as the only way to achieve the best individual performance for a given target dataset is to extensively test all possible sources and, then, choosing the best, the real-word applicability of an approach based on STS will be limited by the amount of available data sources. By using the proposed multiple source approach we can achieve, with high confidence, a performance for the chosen target similar to the individual best observed among all the chosen sources. This observation, however, does not compensate the fact that by manually choosing a single optimal source, the performance observed for the chosen target dataset is consistently better or, in the worst case, in a similar range to the one observed for MS-STS. Further research is needed in order to optimize the choice of source datasets so as to reduce this performance gap.

Another interesting observation concerns the effect of the order in which the multiple sources $[AR1, BR1, CR1]$ are considered during the cyclic evolution of the MS-STS process. In order to assess whether this variable had any discernible effect over the observed performance we chose to run, for each target dataset, a set of six variants of the original results, changing the order in which the three sources are organized during a single cycle: $[[A,B,C], [A,C,B], ..., [C,B,A]]$. The performance plots from Figure 8.4 seem to point to the conclusion that the performance in all 4 datasets converges to a set of values in very similar ranges, regardless of the chosen organization of source datasets along the MS-STS pipeline. This observation leads to the conclusion that, if the best sources are found, there is no need to optimize their order. Whereas the presented example was considerably simple, with a very small number of sources, in a practical application there is no guarantee that the number of combinations becomes unfeasible for a brute force optimization step of their organization. The observed results seem to indicate that this optimization process might be less relevant, especially in scenarios such as the tested, where all sources present a relatively similar nature (flash illumination in this specific case). It is still unclear, due to the preliminary nature of this study how increasing variability in the source dataset conditions would affect these

observations. The focus of future research should, thus, fall on the optimal choice of sources so that the most complete domain generalization is achieved. With this in mind, the aim of future work would be to accurately and intelligibly perform classification under highly variable scenarios, especially using more heterogeneous sets of source information.

8.4 Conclusions

In the present work we proposed an extended version of the Source-Target-Source approach to Deep Transfer Learning, making use of multiple sources of information. We successfully applied the developed algorithm to the specific problem of cross-sensor biometrics, a recent field of research that aims to mitigate the performance drop observed when training and testing acquisition conditions are considerably heterogeneous.

We observed that, when compared to a state-of-the-art algorithm designed for single-sensor scenarios, the proposed STS and MS-STS approaches revealed a worse baseline performance but managed to present a very interesting cross-sensor stability regardless of the nature of the data used in the training process. It is trivial to deduce that an improvement in the baseline performance of any of the proposed methodologies - CNN or SV-SDA - would, necessarily, result in a stable increase of performance in all cross-sensor scenarios. Some ideas to achieve such improvement would necessarily consist on exploring alternatives to the SIFT description chosen for the supervector generation, or on the development of ensemble or joint strategies capable of making the most of the pros of both GMM-UBM (or any other state-of-the-art single-sensor methodology) and STS strategies to simultaneously achieve good baseline and cross-sensor performance. Achieving a tight coupling between both methodologies will, most certainly, represent a very significant step in the field of cross-sensor biometrics.

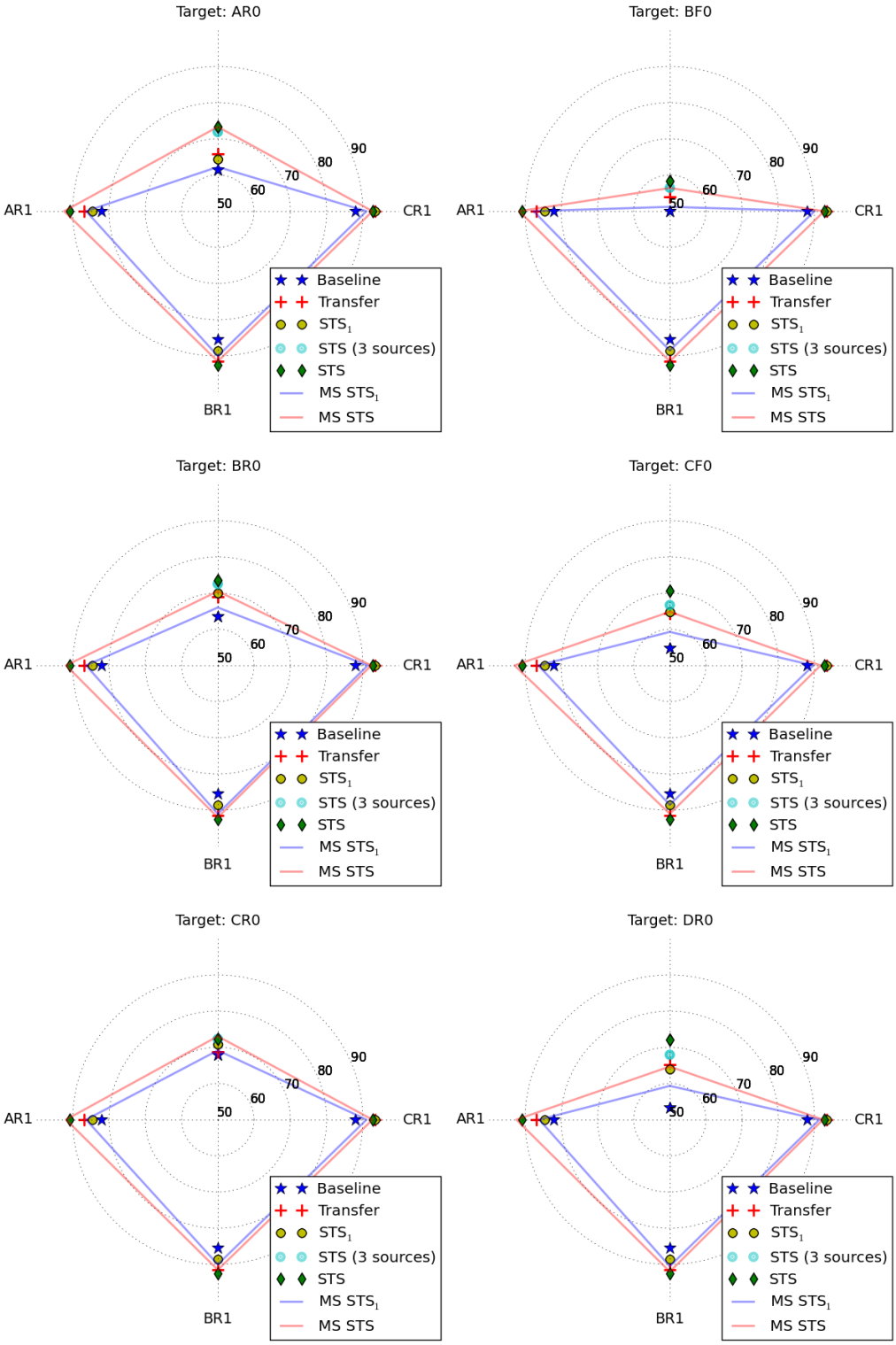


Figure 8.3: Graphical representation of the MS-STS Rank-1 recognition rates obtained for all the no-flash subsets of the CSIP database using all the flash datasets as sources, plotted against the respective BL, TLs and STS results.

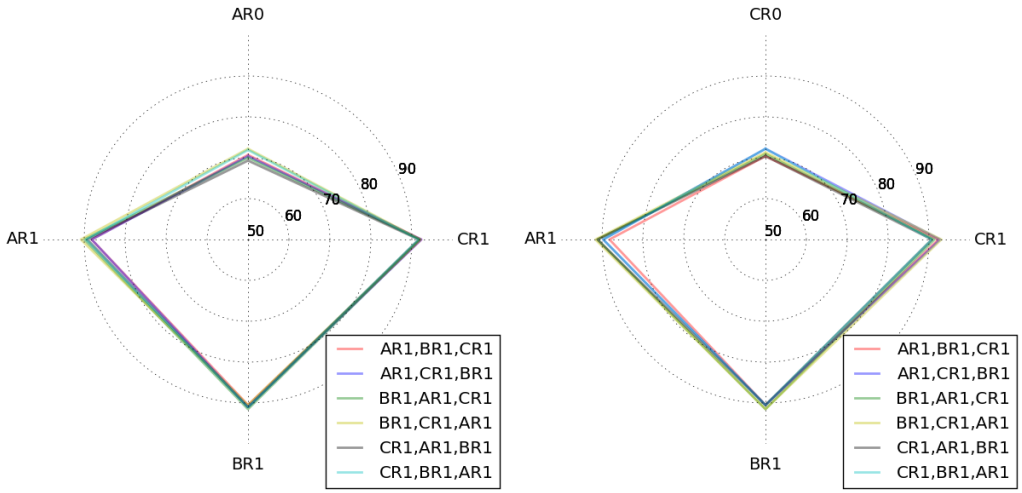


Figure 8.4: Graphical representation of the MS-STS Rank-1 recognition rates obtained for all the six possible orders of the chosen source datasets. Results concern to (a) *AR*0 and (b) *CR*0 as targets.

Part IV

Conclusion and Future work

Chapter 9

Conclusion and Future Work

In this thesis we have designed a Deep Transfer Learning (DTL) framework by combining the advantage of the hierarchical feature representation property of deep networks with the feature reuse property of Transfer Learning methodology.

9.1 DTL mechanisms

We have designed three main mechanisms that harness the advantages of our proposed DTL framework. First, we developed a layer-wise feature transference mechanism called Layerwise Transfer Learning (LTL) using either an unsupervised or supervised learning method. The second mechanism, the Source-Target-Source (STS) mechanism, optimizes the layer-wise feature transference by expanding the optimal solution search space and switching between multiple domains (both source and target). Third, we have developed a Deep Transfer Learning Ensemble (DTLE) mechanism by combining various LTL models as a ensemble for a producing a generalized model. We verified these three DTL mechanisms for the transfer learning settings:

1. **Traditional ML:** The distributions are equal $P_S(X) = P_T(X)$ and the labels are also equal $Y_S = Y_T$.

2. **Traditional TL:** The distributions are equal $P_S(X) = P_T(X)$ and the labels are not equal $Y_S \neq Y_T$.

3. **Labels are equal:** The distributions are different $P_S(X) \neq P_T(X)$ and the labels are equal $Y_S = Y_T$.

4. **Labels are different:** The distributions are different $P_S(X) \neq P_T(X)$ and the labels are not equal $Y_S \neq Y_T$.

Case 1 is a traditional machine learning setting in which the distributions between the source and the target problem are equal $P_S(X) = P_T(X)$ and $Y_S = Y_T$. Therefore there is no difference between the source and the target data. Our literature review showed that the many popular and

successful techniques like domain adaptation, concept drift, and co-variant shift are already developed for the traditional TL transfer setting, in which the distributions are drawn from the same or very closely correlated feature spaces. In our thesis we focused our study to evaluate the performances of our proposed DTL mechanisms for the other two key transfer learning settings- case 3 (labels are equal) and 4 (labels are different)- in which the source and the target distributions are different, $P_S(X) \neq P_T(X)$. We used well-known Jensen-Shannon (JS) divergence (Lin, 1991) to measure the distance between the source and the target distributions.

Toy datasets: We selected specific state-of-the-art toy datasets such that it would enable us to study both case 3 and 4 transfer settings for both object recognition and character recognition applications.

- We used two sets of object recognition problems based on the level of complexity of the classification tasks. The *basic object recognition problems* contains three datasets: canonical, non-canonical and curve vs. corner as shown in Fig 3.3. The (more) *complex object recognition problems* contains two datasets: Shape1 and Shape2.

- We evaluated the DTL framework in two different settings for recognizing characters. We used the *MNIST* dataset and renamed it as *Latin* P_L which had labeled hand-written latin digits from 0 to 9. Then we used the *MADbase* dataset and called it as *Arabic* P_{Ar} which had labeled hand-written arabic digits from 0 to 9. Additionally, the Chars74k dataset was modified to obtain the *Lowercase* dataset P_{LC} with lowercase letters from a-to-z, the *Uppercase* dataset P_{UC} with uppercase letters from A-to-Z, and the *Digits* P_D dataset with digits from 0-to-9. The Latin-2 dataset is a modified version of MNIST to match the number of training and validation instances of the Lowercase dataset.

Problem Categorization: The categorization of the transfer learning problems is subjective in literature. Therefore to add a quantitative measure to study the nature of source and target learning function, we categorize a problem as *harder* if a problem has higher classification error than another. Moreover, if the source problem is harder than the target problem we categorize the transfer learning setting as *Hard Transfer* (HT). In the reverse case, that is, when the roles of such source and target problems are interchanged, the transfer learning setting is categorized as *Reverse Transfer* (RT).

9.1.1 Layerwise Transfer Learning (LTL) mechanism

We analyze two approaches for LTL mechanism: 1) Transfer Learning unsupervised (TLu) and 2) Transfer Learning supervised (TLs) on two deep network models: a) Stacked Denoising Autoencoders and b) Convolutional Neural Network. Both approaches take a classifier trained on a harder problem and then reuse it on a simpler problem with a completely *different task* drawn from a different distribution that performed better than the baseline.

In case 3 transfer settings, **labels are equal** $P_S(X) \neq P_T(X)$ and $Y_S = Y_T$: by transferring features from geometrical shapes to canonical shapes, we achieved a 7.4% relative improvement on

the average error rate in the TLs approach. We also observed negative transfer learning (performance degradation) in both approaches for reverse transfer cases drawn from different distributions.

In case 4 transfer settings, **labels are different** $P_S(X) \neq P_T(X)$ and $Y_S \neq Y_T$: the TLu approach achieved a 7% improvement on accuracy and 41% reduction on computation time for uppercase datasets. Similar results were observed in lowercase datasets. Subsequently, in the TLs approach we achieved lower average error rates than the baseline. The best result was obtained for the TLs:FT approach in which we reused the three supervised hidden layers of the source problem for solving the target problem, and it resulted in a 54% speed up with respect to the baseline. We observed that features trained on harder problems are generic and are able to adapt better to the target problem than ones trained on simpler problems.

9.1.2 Source-Target-Source (STS) mechanism

The main idea of the STS mechanism is to iterate the learning between both source and target domains. The intuition is that, like in typical metaheuristics in optimization, moving the learning from one domain to the other will "shake" the current local optimal solution, allowing us to keep exploring the space of solutions. Ideally, this would allow us to reach a better solution in the process. The STS approach outperforms both baseline and transfer learning approaches. We studied TLu and TLs approaches for transferring generic features on distributions that are similar and transferring specific features on tasks that are different to study the impact of splitting co-adapted neurons. Finally, using the cyclic STS approach reduced the transferability gap between the source and the target tasks. We summarize that the STS outperforms both the baseline and the transfer learning approaches. As a extension we studied the STS approach transferring features from multiple sources, Multi-source STS (MS-STS). The MS-STS demonstrated better domain generalization.

9.1.3 Deep Transfer Learning Ensemble (DTLE) mechanism

When we analyzed case 3 and case 4 transfer learning settings, we observed that transfer specific DTL approaches obtained better overall accuracy than the baseline but were not as good as retrain specific DTL, since the fine-tuned weights of the transfer specific DTL forced the solution to the local minima. DTLE outperformed the baseline and other DTL approaches when the distributions and tasks were different.

9.1.4 User interface

DTL software is an interactive interface for GPU based machine learning algorithims. The software provides an interface to the baseline and various transfer learning methods. The intention of building a user interface for the DTL framework makes it easier for other researchers to comparatively evaluate their own methods.

How to use the DTL software?

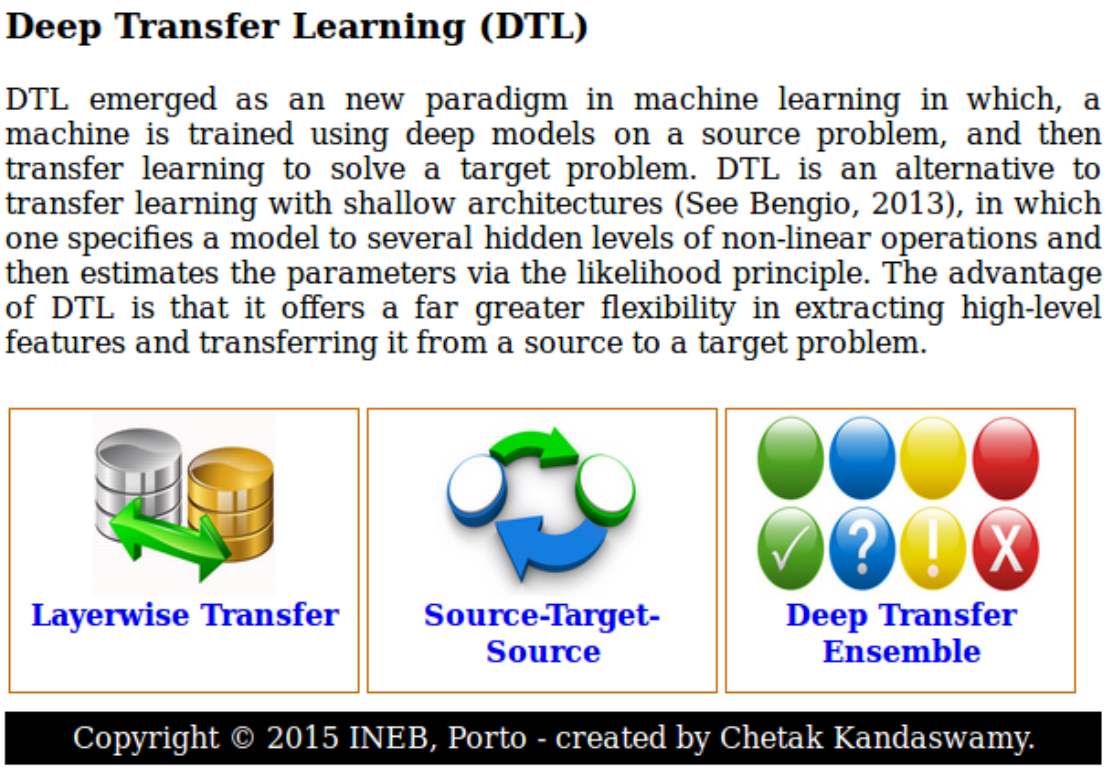


Figure 9.1: A pictorial representation of DTL soft user interface depicting the three DTL mechanisms

The step by step instructions are provided for each of the approaches along with the glossary and DTL software are available at this link: `http://www.deepnets.ineb.up.pt/files/software/Deep_software_interface/DTL_frontend.html`.

9.2 DTL for real-world applications and scenarios

The salient intentions of this research were to demonstrate the designed DTL framework for practical applications like drug discovery and cross-sensor biometric identification. For the first application, we produced state-of-the-art performance on the analysis of high-content breast cancer cell images for drug discovery using the LTL mechanism. We obtained significant improvements in both time and accuracy. For the second application we extended our Source-Target-Source mechanism for cross-sensor biometric classification to identify of human anatomical structure in the periocular region with a multi-source version.

9.2.1 High-content analysis for drug-discovery

We study LTL mechanism to provide classification of chemical mechanisms in action (MOA) of breast cancer cells. In this application we map the input features of each cell to a particular MOA class without using feature reduction methods. This is particularly useful for new drug testing as computational time is saved. The average accuracies of the baseline SAA for P_{set1} and P_{set2}

datasets are about 84% and 87%, respectively, using a seven hidden layer SAA with 500 neurons in each layer. We observed around 30% computational speed up when using the DTL approach.

9.2.2 Cross-sensor biometric recognition

With the increasing popularity and availability of mobile devices capable of performing the whole biometric recognition framework from data acquisition to the final decision, a new obstacle is presented to the development of such systems: the need to adapt to the wide variety of available sensors and their respective heterogeneity with regards to image quality. We used MS-STS approach on the CISP dataset which has ten different datasets. We observed that, when compared to a state-of-the-art algorithm designed for single-sensor scenarios, the proposed STS and MS-STS approaches revealed a worse baseline performance but managed to present a very interesting cross-sensor stability regardless of the nature of the data used in the training process. It is trivial to deduce that an improvement in the baseline performance of any of the proposed methodologies - CNN or SV-SDA - would, necessarily, result in a stable increase of performance in all cross-sensor scenarios.

9.3 Future work

The golden age of machine learning is just beginning (Hemsoth, 2015). The machine learning community is focusing on more broader and ambitious challenges. Never ending learning is one such methodology which attempts to replicate human like learning ability. Humans visualize one aspect from one scenario and reuse it continuously for various other scenarios, with very little training. As a pragmatic option towards the goal of General Intelligence paradigm, Tom Mitchell proposes a broader interdisciplinary view of machine learning involving computer programmers and statisticians who have already contributed to statistical-computational theories of learning processes, with other field experts like psychologists, economists, biologists and neuroscientists, collectively questioning *What kind of process can lead to learning under what conditions for what kind of data?* (Mitchell, 2006).

Mechanism specific extensions to DTL framework

- CNN methodology: Traditionally, this strategy is explored when large datasets are available, so as to achieve the most robust modeling possible. In the current work we used only two training instances from each source dataset (Cross-sensor CISP), thus, theoretically, limiting the potential of achieving good results for the problem at hand. With this observation in mind, we can conclude that testing the proposed approaches on a larger cross-sensor periocular dataset would probably result both in higher baseline as well as higher cross-sensor performances. One must note, however, that the availability of large amounts of data to perform the enrollment step is not guaranteed in real-life applications. This limitation should, therefore, be overcome in the long run if this strategy is expected to be implemented in more

practical solutions. Another focus of future work for CNN would be using information from all three RGB channels instead of grayscale transformation used on the present work.

- Regarding the MS-STS, we may conclude that, even though the optimal STS performance managed to outperform its multi-source counterpart in almost all scenarios, the reasoning for this behaviour is both expected as well as negligible for practical applications. As we are manually choosing the best performing source when presenting the STS results, whereas in the MS-STS we are fixing the same set of sources for all experiments, it is expected that optimal performance is not achieved in the situations where the best performing single source is not included on the set of chosen sources. From a practical point of view, testing the universe of all possible sources to empirically choose the best one is not a viable possibility. The focus of future work should, therefore, fall on the automatic choice of the fittest sources to achieve the highest degree of domain generalization during the MS-STS learning process and, thus, cause the convergence of the MS-STS performance to the best possible STS result. Future work includes testing MS-STS using SV-SDA with instance weighting and pre-training the network with unlabelled biometric data.

- The DTLE method has several options such as retrain specific feature, generic feature, supervised, unsupervised, locked, or unlocked layers. Including a possibility of training from multiple source problem as in the MS-STS approach may lead to better domain generalization. This in combination with having large data samples will have a huge training requirement. The DTLE method can be optimized by developing the algorithm to learn these choices.

Application specific Extensions to DTL framework

- Traffic management using drones, enabling the detection of traffic congestion for both human driven and self-driving cars. Let us consider a scenario, in which drones are assigned the task of performing aerial surveillance of traffic jam and report the situation back to the ground station. The ground based server will assign tasks in real-time while the swarms of drones will be used to communicate among themselves and make task allocation. To perform the task, we would need smart drones that can identify the traffic jam through cooperative communication among themselves and then report back to the ground station with reliable information exchange using suitable routing protocols.

- In many real life problems humans are called to compare or rank items or objects in order to choose the most appropriate item for a specific goal. Think for example of choosing a music to listen, buying clothes, ordering a dish in a restaurant, etc. Other applications include stock trading support systems, where one wants to predict, for instance, whether to buy, keep or sell a stock, and biomedical classification problems, where frequently the classes are ordered. Even when many learning problems involve classifying examples into

classes which have a natural order, this scenario has not received as much attention as the standard classification problem.

- Biometric application: FBI is interested in identifying people or groups using tattoos[1]. This is a biometric application more challenging than the simple finger print or facial recognition, as there are no rigid curves like in finger print scans nor is there specific features of a face.

[1] http://spectrum.ieee.org/computing/software/fbi-wants-better-automated-image-analysis-for-tattoos

Appendix A

Model compression for real-time application

In this appendix, we report preliminary work developed during an internship at National Institute of Informatics, Tokyo at Prendingers' lab. As discussed in the future work of the previous chapter, an important task is the development of a shared dynamic map that helps intelligent transport system for efficient traffic management assisted with smart drones. Traditional mapping techniques require numerous expensive sensors in the car to collect large volumes of data that then need to be recorded and processed offline. By contrast, drone ecosystems could efficiently move much of the data processing into the drone, minimizing communication with the cloud. Mapping helps to track the performance of not only automated cars but also other vehicles.

Let us consider a scenario of traffic management at a freeway junction as shown in Fig A.1, in which swarms of drones are assigned the task of performing aerial surveillance of traffic jam and report the situation back to the ground station. To analyze the traffic, the drones need to understand high level semantic information of the aerial scene in real-time through cooperative communication among themselves and then report back to the ground station with reliable information exchange using suitable routing protocols. We opted semantic segmentation as a classification problem for the smart drones.

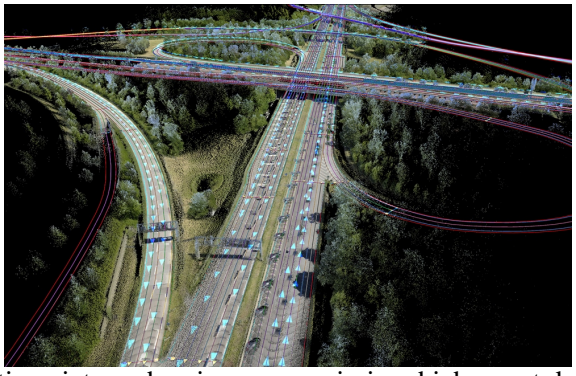


Figure A.1: An illustrative picture showing a scenario in which smart drones build a shared map and track the traffic movements at a freeway junction. Picture courtesy of NVIDIA.

Table A.1: Comparison of Deep Learning object recognition architectures

Models	Network depth	Model size (MB)	Number of parameters (Millions)	Number of operations (GFLOP)
AlexNet	8	58	61	1.5
VGG-16	16	553	138	15.3
GoogLeNet	22	35	6.8	1.5
ResNet-50	50	102	0.8	3.8
Resnet-101	101	178	1.6	7.6
Resnet-152	152	241	2.3	11.3

Semantic segmentation is the problem of partitioning an image into discrete components that correspond to semantically meaningful categories. To perform semantic segmentation, we follow the techniques proposed in (Long et al., 2015) and train extract-upscale CNN architectures. These are CNN architectures consisting of an 'extract' stage, a modified object classification CNN architecture such as AlexNet (Krizhevsky et al., 2012), VGG-16 (Simonyan and Zisserman, 2014), GoogLeNet (Szegedy et al., 2015), or Resnet (He et al., 2016), followed by an second 'upscale' stage, which up-samples the coarse output of the extract stage to the original resolution of the image. We performed a comparison of the sizes and computational costs of a variety of different CNN architectures for object recognition to assess their suitability for real-time applications in embedded devices (see Table A.1).

Standard dataset: To test our proposed DTL approaches, we use standard semantic segmentation toy datasets, an extended version of Pascal VOC dataset (Everingham et al., 2012). The train set contained 8498 images from the SBD training set (Hariharan et al., 2011) and the test set contains 736 images from PASCAL VOC 2011, none of which are contained in the train set used in (Long et al., 2015). We modified the Microsoft Common Objects in COntext (MS COCO) (Lin et al., 2014) dataset that contains 91 object categories to match the Pascal VOC dataset for testing our proposed LTL, MS-STS and DTLE approaches.

Drone dataset: We built two aerial dataset: 1) 31 aerial images from Okutama (Oku-data), Japan and tested on the drone videos, and 2) 100 aerial images from Switzerland (Swiss-data). Both the dataset contains 10 classes, that are background, outdoor structures, buildings, paved ground, non-paved ground, train tracks, plants, wheeled vehicles, water, and people classes. The data collection is at very preliminary stage and we plan to collect more data at a later stage of the project. This initial very small data for complex semantic segmentation problem would be useful to demonstrate the data-efficiency and domain-generalization of DTL approaches.

Model compression for DTLE

For real-time applications with drones, the propose DTLE model is bulky and slow. To have a light, fast and accurate model we use model compression (Bucilua et al., 2006), and then expand

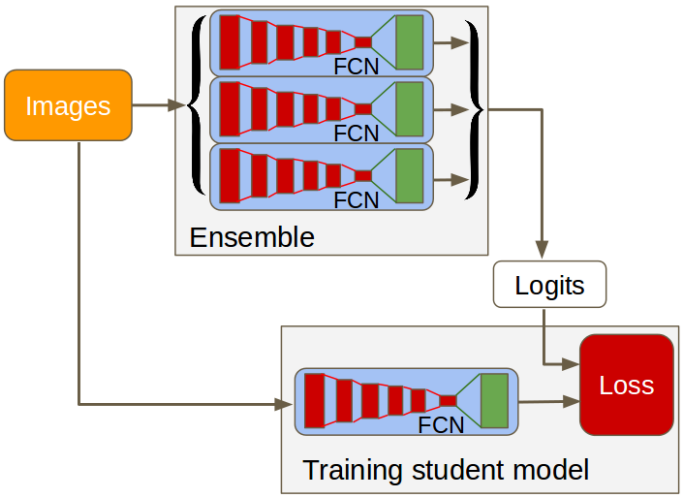


Figure A.2: Block diagram of model compression method for Ensemble of Deep Learning Models for Semantic Segmentation.

on the 'distillation' a transfer learning technique proposed in (Hinton et al., 2015). Model compression shows that large ensembles of models can be compressed to a single model, by training the single model to mimic the outputs (logits) of the ensemble. The resultant compressed model outperforms a model of the same architecture that is trained from hand-labelled data instead of from the ensemble. The complex deep nets can be trained with smaller models (Ba and Caruana, 2014); a block diagram for the model compression method is shown in the Fig A.2.

Inspired by the FCN structures described in (Long et al., 2015), we transform a ResNet into an FCN-ResNet in the following way. Starting with a set of pre-trained ResNet weights, we remove the final average-pooling, fully-connected, and softmax layers of the network. We replace these with a 1×1 convolution layer with a number of filters corresponding to the number of categories, a 64×64 'deconvolution' layer with stride 32, and a crop layer. The 1×1 convolution layer is randomly initialized with variables drawn from a Gaussian distribution. The weights of the 64×64 'deconvolution' layer are manually set to bilinear interpolation, and it is "frozen" in this arrangement by setting its learning rate to 0. We adapted three residual networks of different depths: ResNet-50, ResNet-101 and ResNet-152, having respectively 50, 101 and 152 layers. To further improve the accuracy of Fully Convolutional residual networks, we introduce "skip layers" into our FCN-ResNet architecture, similar to the FCN-8s architecture described in (Long et al., 2015). By incorporating information from earlier stages in the network into the up-sampling procedure, we could likely increase the spatial accuracy of the network, See Fig A.3 for the adaptation block diagram.

The compression process occurs in two steps: pre-training and knowledge transfer. In the pre-training step, we train a semantic segmentation model on a set of labelled images in the usual way. When the training converges, we commence the knowledge transfer step.

In the knowledge transfer step, we use a second dataset. We produce labels for this data set by running it through a pre-existing ensemble model, and storing the full logit vector produced

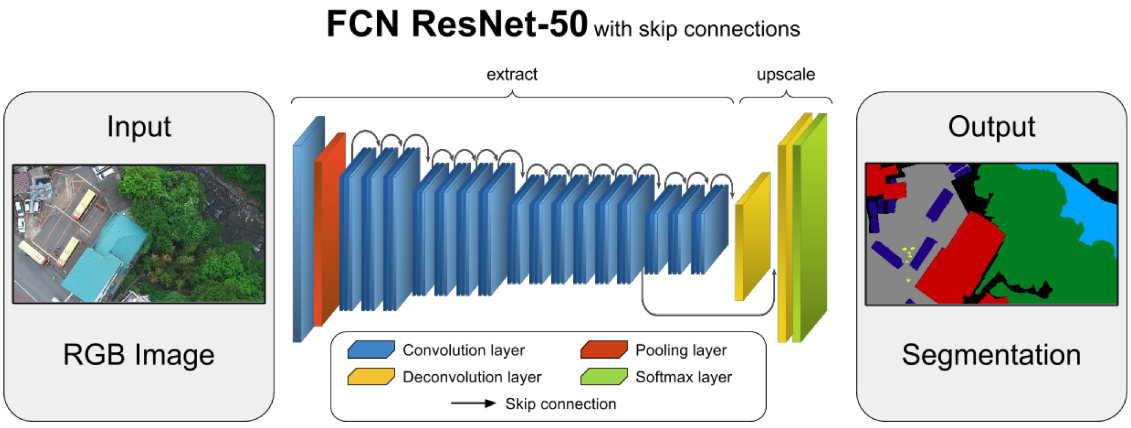


Figure A.3: Block diagram of Extract-upscale method of Deep Learning Models modified for Semantic Segmentation with skip connections.

by the ensemble for each pixel of each image. We then continue training the new model but now by training it to minimize the difference between its own output logit vectors and those of the ensemble.

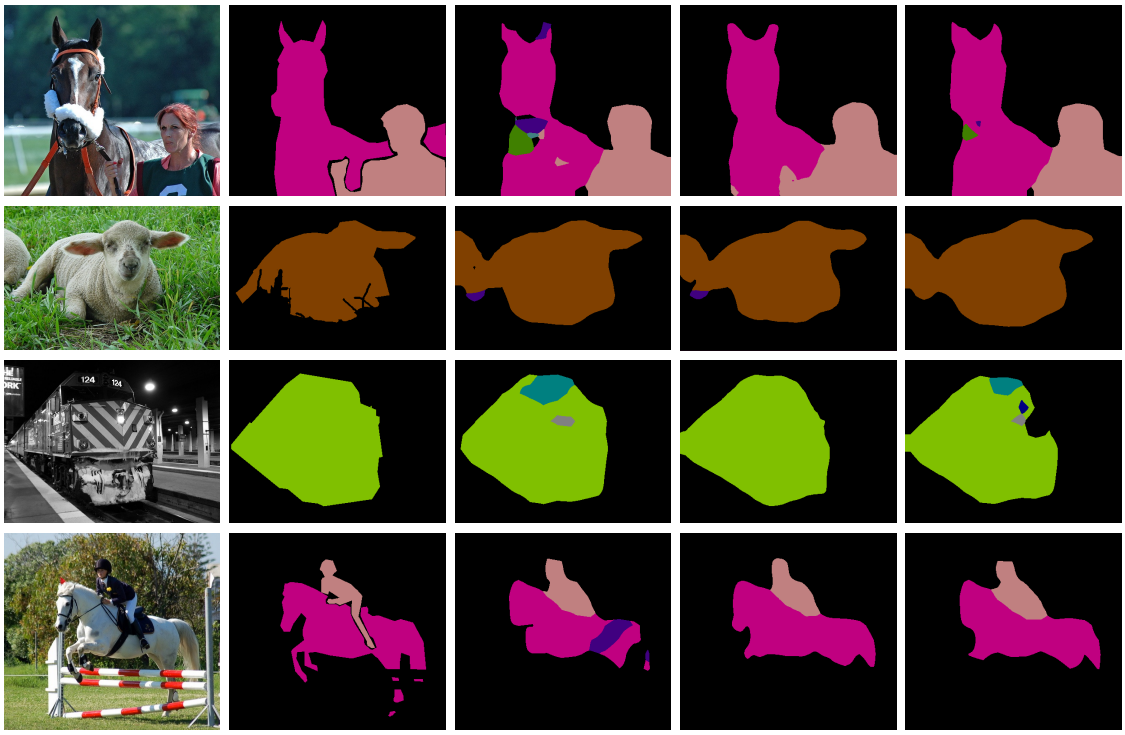
Table A.2: Model compression accuracy in Percentage with DTLE approach

FCN models	Mean IU (%)	Pixel acc. (%)	Forward time (sec)
VGG-16 (32s)	63.6	90.5	0.1
VGG-16 (16s)	65.0	91.0	0.1
VGG-16 (8s)	65.5	91.2	0.1
GoogLeNet	54.7	88.4	**0.06**
ResNet-50	60.6	90.4	**0.06**
Resnet-101	64.0	91.2	0.11
Resnet-152	65.0	91.5	0.11
Ensemble	**67.7**	**92.1**	1.29
Compressed model (Resnet-152)	66.1	91.7	0.11

A.1 Results

Standard dataset: PASCAL VOC. In Table A.2, we compare the accuracy and forward computation time of various trained or pre-trained single models, an ensemble of these models together and a compressed model from that ensemble. We observe that even though the compressed network does not achieve a mean IU[1] and pixel accuracy as high as the ensemble, it is better than any of the single models in the ensemble, and its forward computation time is less than 10% of the ensemble. Thus our method has allowed us to capture some of the ensemble's superior performance,

[1]Mean intersection over union, this is simply the unweighted average of the Jaccard indices of each category. This is the most widely-used metric for semantic segmentation.



Input Ground-truth Single model Ensemble Compressed

Figure A.4: Comparison between output labels for the single and compressed FCN-ResNet-152 models, the ensemble and the ground-truth. In most cases, the compressed model is getting closer to the segmentation quality of the ensemble.

while maintaining the speed of a single FCN-ResNet. When analysing the predictions output by our networks, we observe that knowledge transfer helps avoid some class confusions in certain pixel areas, as we can see in the horse's neck in Fig. A.4. In this case, the ensemble has largely corrected for what one of its component networks perceived as an ambiguity. Learning from the ensemble in this regard, the distilled network also knows how to handle this ambiguity.

Table A.3: Model compression accuracy in Percentage with MS-STS approach

FCN models	Mean IU (%)	Pixel acc. (%)	Forward time (sec)
VGG-16 (8s)	65.5	91.2	**0.1**
MS-STS Resnet-152	68.6	92.6	0.11
Ensemble	**70.6**	**92.9**	0.4
Compressed model (Resnet-152)	69.6	92.7	0.11

In Table A.3, we compare the accuracy and forward computation time of pre-trained single models FCN-8 and MS-STS approach using PASCAL VOC and MS COCO data set, an ensemble of these models together and a compressed model from that ensemble. We observe that the MS-STS approach performs better than the FCN-8s single model and the compressed model (Resnet-152) performed almost good as the ensemble.

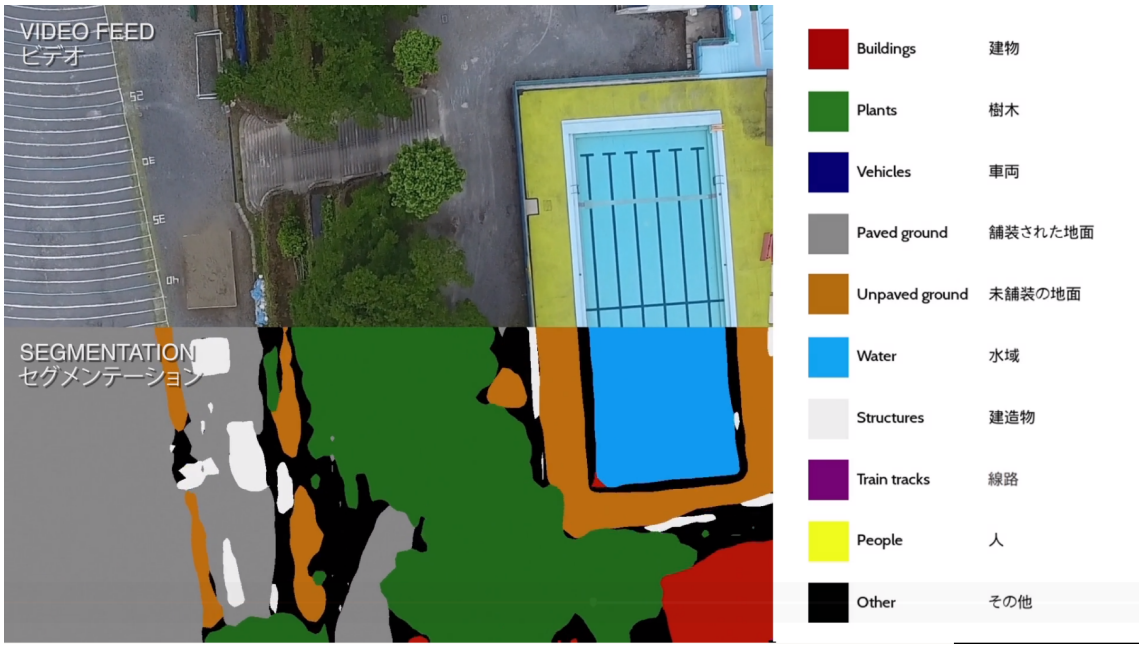


Figure A.5: Screen shot of the Hikawa primary school premises marking the pool and the play area performing semantic segmentation on FCN-Resnet-50 with skip connections.

Drone dataset: We used pre-trained FCN-Resnet-50 to train for the Okutama drone data with 31 images. This model is faster and as accurate as other FCN models. We could observe that the segmentation of people class were not accurate. We used a weighted class method for the classes to observe the results. We do not yet have the complete dataset, thus we have the results of the video segmentation, and thus we do not perform any metrics evaluation on this dataset. In Fig A.5 is a screen shot of the video segmentation in a Hikawa primary school.

A.2 Conclusions and Future Work

We have demonstrated a single semantic segmentation model can be trained to closely replicate the accuracy of an ensemble of deep model, outperforming all of the individual deep models in the ensemble, while retaining the real-time performance of an individual net. As mentioned previously, the knowledge transfer stage of the compression process does not require any labeled data. We are planning to explore further cases where unlabeled data might be essential, and where we can fully realize the potential of our model compression technique. We have collected only initial set of drone dataset with 31 images. We want to enrich the dataset with more complex category classification for the traffic management based scenarios.

Acknowledgements

We would like to thank the support of Prof. Helmut Prendinger, National Institute of Informatics, Tokyo and Dr. Kai Yan from LabRomance and enRoute Co., Ltd.for their guidance and support.

We thank Mr. Andrew Holliday, Mr. Johannes Laurmaa, Mr. Pierre Ecarlat and Mr. Kim Samba for the support and technical assistance to this research. We thank Ruben Geraldas and Naoki Tada for organizing the aerial data collection at the Okutama, Japan using drones.

References

Amaral, T., Silva, L. M., Alexandre, L. A., Kandaswamy, C., Santos, J. M., and de Sá, J. M. Using different cost functions to train stacked auto-encoders. In *Mexican international conference on artificial intelligence (MICAI)*, pages 114–120. IEEE, 2013. doi: 10.1109/MICAI.2013.20.

Amaral, T., Kandaswamy, C., Silva, L. M., Alexandre, L., Marques de Sá, J., and Santos, J. M. Improving performance on problems with few labelled data by reusing stacked auto-encoders. In *International conference on Machine Learning and Applications (ICMLA)*, pages 367–372. IEEE, 2014a. doi: 10.1109/ICMLA.2014.65.

Amaral, T., Silva, L. M., Alexandre, L. A., Kandaswamy, C., de Sá, J. M., and Santos, J. M. Transfer learning using rotated image data to improve deep neural network performance. In *Image Analysis and Recognition*, pages 290–300. Springer, 2014b. doi: 10.1007/978-3-319-11758-4_32.

Ba, J. and Caruana, R. Do deep nets really need to be deep? In *Advances in Neural Information Processing Systems*, pages 2654–2662, 2014.

Beadle, G. F., Silver, B., Botnick, L., Hellman, S., and Harris, J. R. Cosmetic results following primary radiation therapy for early breast cancer. *Cancer*, 54(12):2911–2918, 1984.

Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., and Vaughan, J. W. A theory of learning from different domains. *Machine learning*, 79(1-2):151–175, 2010.

Benenson, R. Discover the current state of the art in objects classification, accessed January 12, 2016. URL `http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html`.

Bengio, Y., Courville, A., and Vincent, P. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, Aug 2013. ISSN 0162-8828. doi: 10.1109/TPAMI.2013.50.

Bengio, Y. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2 (1):1–127, 2009. doi: 10.1561/2200000006. Now Publishers.

Bengio, Y. Deep learning of representations for unsupervised and transfer learning. *Unsupervised and Transfer Learning Challenges in Machine Learning*, 7:19, 2012.

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., and Bengio, Y. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, volume 4, 2010.

Bishop, C. M. *Pattern recognition and machine learning*. springer, 2006.

Blitzer, J., McDonald, R., and Pereira, F. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 120–128, 2006.

Bruzzone, L. and Marconcini, M. Domain adaptation problems: A dasvm classification technique and a circular validation strategy. *IEEE transactions on pattern analysis and machine intelligence*, 32(5):770–787, 2010.

Bucilua, C., Caruana, R., and Niculescu-Mizil, A. Model compression. In *International Conference on Knowledge Discovery and Data Mining*, pages 535–541. ACM, 2006.

Campbell, W. M., Sturim, D. E., and Reynolds, D. A. Support vector machines using gmm supervectors for speaker verification. *Signal Processing Letters, IEEE*, 13(5):308–311, 2006.

Carpenter, A. E., Jones, T. R., Lamprecht, M. R., Clarke, C., Kang, I. H., Friman, O., Guertin, D. A., Chang, J. H., Lindquist, R. A., Moffat, J., and others. CellProfiler: image analysis software for identifying and quantifying cell phenotypes. *Genome biology*, 7(10):R100, 2006.

Caruana, R. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

Ciresan, D. C., Meier, U., and Schmidhuber, J. Transfer learning for latin and chinese characters with deep neural networks. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE, 2012.

Connaughton, R., Sgroi, A., Bowyer, K. W., and Flynn, P. A cross-sensor evaluation of three commercial iris cameras for iris biometrics. In *Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 90–97. IEEE, 2011.

Cook, D., Feuz, K. D., and Krishnan, N. C. Transfer learning for activity recognition: A survey. *Knowledge and information systems*, 36(3):537–556, 2013.

Cortes, C. and Vapnik, V. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Dalal, N. and Triggs, B. Histograms of oriented gradients for human detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 886–893. IEEE, 2005.

Davis, S. B. and Mermelstein, P. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 28(4):357–366, 1980.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. The PASCAL Visual Object Classes Challenge 2012 (VOC2012), 2012.

Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193 – 202, April 1980. ISSN 0340-1200, 1432-0770.

Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4):44, 2014.

Gasser, L., Lakkaraju, K., Ray, S., and Swarup, S. Darpa baa 05-29: Transfer learning issues and potential contributions. *University of Illinois at Urbana-Champaign*, 2005.

Ge, Z., McCool, C., Sanderson, C., and Corke, P. Modelling local deep convolutional neural network features to improve fine-grained image classification. In *International Conference on Image Processing (ICIP)*, pages 4112–4116. IEEE, 2015.

Glorot, X., Bordes, A., and Bengio, Y. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *International Conference on Machine Learning (ICML)*, pages 513–520, 2011.

Google. Google Trends keyword search for state-of-the-art machine learning algorithms, 2015. URL `http://www.google.com/trends/explore?hl=en-US#q=Machine+Learning,Support+vector+machines,Deep+Learning,+Random+forest+&cmpt=q`.

Hariharan, B., Arbeláez, P., Bourdev, L., Maji, S., and Malik, J. Semantic contours from inverse detectors. In *Conference on Computer Vision (ICCV)*, pages 991–998. IEEE, 2011.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016.

Hemsoth, N. Why The Golden Age Of Machine Learning is Just Beginning, 2015. URL `http://www.theplatform.net/2015/10/20/why-the-golden-age-of-machine-learning-begins-now/`.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network, 2015.

Hinton, G. E., Sejnowski, T. J., and Ackley, D. H. *Boltzmann machines: Constraint satisfaction networks that learn*. Carnegie-Mellon University, Department of Computer Science Pittsburgh, PA, 1984.

Hinton, G. E., Osindero, S., and Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

Holst, A., de Giorgio, A., and Lansner, A. A study on the similarities of deep belief networks and stacked autoencoders. Master's thesis, KTH, School of Computer Science and Communication, 2015.

Hubel, D. H. and Wiesel, T. N. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

Jiang, J. A literature survey on domain adaptation of statistical classifiers. *URL: http://sifaka. cs. uiuc. edu/jiang4/domainadaptation/survey*, 2008.

Jillela, R. and Ross, A. Matching face against iris images using periocular information. In *International Conference on Image Processing (ICIP)*, pages 4997–5001. IEEE, 2014.

Kandaswamy, C., Silva, L. M., Alexandre, L. A., Santos, J. M., and de Sá, J. M. Improving deep neural network performance by reusing features trained with transductive transference. In *Artificial Neural Networks and Machine Learning–ICANN 2014*, pages 265–272. Springer, 2014a. doi: 10.1007/978-3-319-11179-7_34.

Kandaswamy, C., Silva, L. M., Alexandre, L. A., Sousa, R., Santos, J. M., de Sá, J. M., et al. Improving transfer learning accuracy by reusing stacked denoising autoencoders. In *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 1380–1387. IEEE, 2014b. doi: 10.1109/SMC.2014.6974107.

Kandaswamy, C., Silva, L. M., Alexandre, L., Sousa, R., Santos, J. M., de Sá, J. M., et al. Improving transfer learning accuracy by reusing stacked denoising autoencoders. In *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*, pages 1380–1387. IEEE, 2014c.

Kandaswamy, C., Silva, L. M., Alexandre, L. A., and Santos, J. M. Deep transfer learning ensemble for classification. In *Advances in Computational Intelligence*, pages 335–348. Springer, 2015a. doi: 10.1007/978-3-319-19258-1_29.

Kandaswamy, C., Silva, L. M., and Cardoso, J. S. Source-target-source classification using stacked denoising autoencoders. In *Pattern Recognition and Image Analysis*, pages 39–47. Springer, 2015b.

Kandaswamy, C., ao. C. Monteiro, J., Silva, L. M., and Cardoso, J. S. Multi-source deep transfer learning for cross-sensor biometrics. *Conditionally accepted in Neural Computing and Applications Journal*, 2016a.

Kandaswamy, C., Silva, L. M., Alexandre, L. A., and Santos, J. M. High-content analysis of breast cancer using single-cell deep transfer learning. *Journal of Biomolecular Screening, SAGE*, 2016b. doi: 10.1177/1087057115623451.

Krizhevsky, A., Sutskever, I., and Hinton, G. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1106–1114, 2012.

Kuncheva, L. I. *Combining pattern classifiers: methods and algorithms*. John Wiley & Sons, 2004.

Kurzweil, R. *The singularity is near: When humans transcend biology*. Penguin, 2005.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lee, T. S., Mumford, D., Romero, R., and Lamme, V. A. The role of the primary visual cortex in higher level vision. *Vision research*, 38(15):2429–2454, 1998.

Li, B. Cross-domain collaborative filtering: A brief survey. In *Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on*, pages 1085–1086. IEEE, 2011.

Li, F.-F. and Karpathy, A. Standford Course on convolutional neural networks for visual recognition, 2015. URL `http://cs231n.stanford.edu/`.

Lin, J. Divergence measures based on the shannon entropy. *Information Theory, IEEE Transactions on*, 37(1):145–151, 1991.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

Ljosa, V., Caie, P. D., ter Horst, R., Sokolnicki, K. L., Jenkins, E. L., Daya, S., Roberts, M. E., Jones, T. R., Singh, S., Genovesio, A., and others. Comparison of methods for image-based profiling of cellular morphological responses to small-molecule treatment. *Journal of biomolecular screening*, page 1087057113503553, 2013.

London, M. and Sessa, V. I. How groups learn, continuously. *Human Resource Management*, 46 (4):651–669, 2007.

Long, J., Shelhamer, E., and Darrell, T. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

Lowe, D. G. Object recognition from local scale-invariant features. In *IEEE international conference on Computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

Marr, B. 5 ways machine learning is reshaping our world, 2015. URL `http://www.forbes.com/sites/bernardmarr/2015/10/22/5-ways-machine-learning-is-reshaping-our-world/`.

Mitchell, T. M. *The discipline of machine learning*, volume 17. Carnegie Mellon University, School of Computer Science, Machine Learning Department, 2006.

Monteiro, J. C. and Cardoso, J. S. Periocular recognition under unconstrained settings with universal background models. In *International Conference on Bio-inspired Systems and Signal Processing (BIOSIGNALS)*, 2015.

Monteiro, J. C., Esteves, R., Santos, G., Fiadeiro, P. T., Lobo, J., and Cardoso, J. S. *A Comparative Analysis of Two Approaches to Periocular Recognition in Mobile Scenarios*, pages 268–280. Springer International Publishing, Cham, 2015. ISBN 978-3-319-27863-6. doi: 10.1007/978-3-319-27863-6_25. URL `http://dx.doi.org/10.1007/978-3-319-27863-6_25`.

Oliveira, H. *An Affordable and Practical 3D Solution for the Aesthetic Evaluation of Breast Cancer Conservative Treatment*. PhD thesis, University of Porto, 2013.

Pan, S. J. and Yang, Q. A survey on transfer learning. *Knowledge and Data Engineering, IEEE Transactions on*, 22(10):1345–1359, 2010.

Patricia, N. and Caputo, B. Learning to learn, from transfer learning to domain adaptation: A unifying perspective. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1442–1449. IEEE, 2014.

Perkins, D. N. and Salomon, G. Transfer of learning. *International encyclopedia of education*, 2, 1992.

Pillai, J., Puertas, M., and Chellappa, R. Cross-sensor iris recognition through kernel learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(1):73–85, Jan 2014.

Proença, H. and Alexandre, L. The nice.i: noisy iris challenge evaluation - part i. In *International Conference on Biometrics: Theory, Applications, and Systems*, pages 1–4. IEEE, 2007.

Raina, R., Battle, A., Lee, H., Packer, B., and Ng, A. Y. Self-taught learning: transfer learning from unlabeled data. In *International conference on Machine learning*, pages 759–766. ACM, 2007.

Razavian, A. S., Azizpour, H., Sullivan, J., and Carlsson, S. Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 512–519. IEEE, 2014.

Reynolds, D. A., Quatieri, T. F., and Dunn, R. B. Speaker verification using adapted gaussian mixture models. *Digital signal processing*, 10(1):19–41, 2000.

Salakhutdinov, R. and Hinton, G. E. Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pages 448–455, 2009.

Salakhutdinov, R., Mnih, A., and Hinton, G. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798, 2007.

Santos, G., Grancho, E., Bernardo, M. V., and Fiadeiro, P. T. Fusing iris and periocular information for cross-sensor recognition. *Pattern Recognition Letters*, 57:52–59, 2015.

Scherer, D., Muller, A., and Behnke, S. Evaluation of pooling operations in convolutional architectures for object recognition. In *Artificial Neural Network, 2010*, pages 92 – 101. Springer, 2010.

Serre, T., Wolf, L., and Poggio, T. Object recognition with features inspired by visual cortex. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 994–1000. IEEE, 2005.

Shamir, L. Assessing the efficacy of low-level image content descriptors for computer-based fluorescence microscopy image analysis. *Journal of microscopy*, 243(3):284–292, 2011.

Shao, L., Zhu, F., and Li, X. Transfer learning for visual categorization: A survey. *IEEE transactions on neural networks and learning systems*, 26(5):1019–1034, 2015.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

Singh, S., Carpenter, A. E., and Genovesio, A. Increasing the Content of High-Content Screening An Overview. *Journal of biomolecular screening*, 19(5):640–650, 2014.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

Tan, T., He, Z., and Sun, Z. Efficient and robust segmentation of noisy iris images for non-cooperative iris recognition. *Image and vision computing*, 28(2):223–230, 2010.

Taylor, M. E. and Stone, P. Transfer learning for reinforcement learning domains: A survey. *The Journal of Machine Learning Research*, 10:1633–1685, 2009.

Thrun, S. Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Springer, 1998.

Thrun, S. and Pratt, L. *Learning to learn*. Springer Science & Business Media, 2012.

Tomasz. Tombone's computer vision blog, 2015. URL `http://www.computervisionblog.com/2015/01/from-feature-descriptors-to-deep.html`.

Turing, A. M. Computing machinery and intelligence. *Mind*, pages 433–460, 1950.

Van Otterlo, M. A survey of reinforcement learning in relational domains, 2005.

Vilalta, R. and Drissi, Y. A perspective view and survey of meta-learning. *Artificial Intelligence Review*, 18(2):77–95, 2002.

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*, pages 1096–1103, 2008.

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. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010. ISSN 1532-4435.

Xu, Q. and Yang, Q. A survey of transfer and multitask learning in bioinformatics. *Journal of Computing Science and Engineering*, 5(3):257–268, 2011.

Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems*, pages 3320–3328, 2014.

Young, D. W., Bender, A., Hoyt, J., McWhinnie, E., Chirn, G.-W., Tao, C. Y., Tallarico, J. A., Labow, M., Jenkins, J. L., Mitchison, T. J., and others. Integrating high-content screening and ligand-target prediction to identify mechanism of action. *Nature chemical biology*, 4(1):59–68, 2008.

Zhu, X. Semi-supervised learning literature survey, 2006.