Final Report
(Work carried out from March 2014 to February 2015)

Ricardo Sousa

27 February 2015

Project "Reusable Deep Neural Networks: Applications to Biomedical Data"
(PDTC/EIA-EIA/119004/2010)
Neural Networks Interest Group
Instituto de Engenharia Biomédica (INEB)
Rua Dr. Roberto Frias, 4200-465, Porto, Portugal

Abstract

Transfer Learning (TL) aims to transfer knowledge acquired in one problem, the source problem, onto another problem, the target problem, dispensing with the bottom-up construction of the target model. The TL approach has gained significant interest in the Machine Learning (ML) community since it paves the way to devise intelligent learning models that can easily be tailored to many different domains of applicability. As it is natural in a fast evolving area, a wide variety of TL settings and nomenclature have been proposed so far. During this grant we presented, together with a survey of the literature on the majority of TL methods, a unifying view of the many TL settings with a common nomenclature suitable to classification problems.

Another work that was conducted was related to the performance assessment of automatic Computer Vision (CV) tools for immunogold particles detection. Our approach based on Laplacian of Gaussian (LoG) filter was applied and compared against the Spot Detector (SD) publicly available as part of the Icy bioimaging software. Two datasets were created for this study: a first dataset for benchmarking purposes and a second dataset to validate our method. Our approach (LoG detector) outperformed in almost every scenario the SD approach. On the second dataset, LoG attained more than 86.9% of accuracy. The automatic CV tool significantly aids in the detection of immunogold particles, performing better than its counterpart and also has few parameters, making it intuitive for a researcher unfamiliar to CV to handle this tool. This procedure allowed us to pave the way for TL on biomedical applications with Stacked Denoising Autoencoders (SDAs).

Contents

List of Tables

Chapter 1

Introduction

1.1 Research Plan and Document Structure

For this grant we had as objectives the study of the reusability capacity of Deep Neural Networks (DNNs) (such as auto-encoders or Convolutional Neural Networks (CNNs)) known as Transfer Learning (TL). It was also purpose of this work the development of new TL strategies to improve accuracy on classification problems and to propose a clear framework for TL terms and methods.

Based on the material that was developed during this grant, and its achievements, this document is structured as follows: Chapter 2 presents the installation procedures that we have conducted for the setup of our High Performance Computer (HPC). Chapter 3 presents a survey on TL and a unifying view of the many TL settings with a common nomenclature suitable to classification problems. In Chapter 4 we delve the possibility of automatic detection of immunogold particles in Transmission Electron Microscopy (TEM) imaging. This preliminary work allowed us to determine the possibility of the exploration of Stacked Autoencoder (SAE) or Stacked Denoising Autoencoder (SDA) approaches for the recognition of organalles. Finally, in Chapter 5 we present a summary of our work and our future research plan.

1.2 Contributions and Scientific Indicators

The work conducted during this research grant can be summarized as follows:

- Submission of a journal paper to IEEE Transactions on Neural Networks and Learning Systems (Impact Factor (IF): 4.37) on TL terminology and its current status;

- A study of immunogold particle detection was conducted. This preliminary research paves the way for TL with biomedical applications.

In doing so, we were able to conduct the following scientific proposals:

- A clear presentation of the learning problem of TL and its implications;
- A survey on TL on its recent advancements;
- A system for the detection and quantification of immunogold particles.

These proposals are currently being evaluated by our peers.

Chapter 2

High Performance Computer Setting

Before starting reading this chapter it is expected that the reader is familiar with linux and its command line environment. If you do **not** feel comfortable with this, you should skip this Chapter or start by reading 'The Linux Documentation Project' (TLDP): `http://www.tldp.org/`.

2.1 HPC Configuration

About the system:

- Core i7 3770K, 3.50Ghz;
- 16Gb;
- 2Tb Disk;
- 2 $\times$ GPU Nvidia GTX 770.

And specifications of the Operating System (OS) and filesystem structure:

- debian, 7.5;
- filesytem structure:

```
Filesystem         Size  Used Avail Use% Mounted on
rootfs             909G  205G  658G  24% /
/dev/sda1          2.4G  4.0K  2.4G   1% /boot/efi
/dev/sda6          909G   24G  839G   3% /opt
```

- directory structure: /opt/datasets is the folder for repositories and /opt/sources for hand compiled system libraries.

Important NOTES:

- Truetype font (lstlisting) means command line instruction or result;
- A # symbol preceding a given sentence means that the command was executed with root privileges.
- Same compiler versions:

  ```
  g++ --version
  gcc --version
  gfortran --version
  ```

2.1.1 Users

Each user will have a specific account but will belong to the same group (`users`). In /etc/adduser.conf, USERGROUPS=yes was changed to USERGROUPS=no

2.2 Python, LAPACK, OpenBlas, and Theano Configuration

2.2.1 Python

Control version system software: `# apt-get install git subversion` and for coding purposes: `# apt-get install g++ gfortran`

Some requirements: `# apt-get install python-dev`

PIP is mandatory:

```
$ git clone https://github.com/pypa/pip
$ python setup.py build
# python setup.py install
```

Install now pyparsing, cython and pillow (other dependencies):

```
# pip install pyparsing
# pip install cython
# pip install pillow
```

2.2.2 OpenBLAS

You need basic linear algebra subprograms (blas) and other routines/libraries. We have opt to use openblas for three very strong reasons: 1) system binaries are architecture independent and therefore not very well optimized; 2) for some particular reason, atlas was not linking with numpy and other libraries; 3) openblas is far more easy to compile/install and is as fast as atlas.

```
# git clone https://github.com/xianyi/OpenBLAS.git
# cd OpenBLAS
# make FC=gfortran BINARY=64 USE_THREAD=1 DYNAMIC_ARCH=0
# make PREFIX=/usr/local/ install
# ldconfig
```

Do not forget to change your `$~/.bashrc` file (standard configuration at `/etc/bashrc.bashrc`):

```
export OPENBLAS_NUM_THREADS=8
export BLAS=/usr/local/lib/libopenblas.a
export LAPACK=/usr/local/lib/libopenblas.a
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib/

export PATH=$PATH:$HOME/usr/bin:/usr/local/cuda/bin
```

2.2.3 NumPy

```
git clone https://github.com/numpy/numpy
cd numpy
```

Edit site.cfg file so that:

```
[openblas]
libraries = openblas
library_dirs = /usr/local/lib
include_dirs = /usr/local/include
```

Do not forget to check your configurations, and only afterwards you should build and install.

```
$ python setup.py config
$ python setup.py build
# python setup.py install
```

test numpy with this:
https://gist.github.com/raw/3842524/df01f7fa9d849bec353d6ab03eae0c1ee68f1538/test_numpy.py

```
$ OPENBLAS_NUM_THREADS=1 python build/test_numpy.py
$ OPENBLAS_NUM_THREADS=8 python build/test_numpy.py
```

In this system, results should be around 0.07 and 0.02 seconds, respectively.

```
# OPENBLAS_NUM_THREADS=1 python numpy/build/test_numpy.py
FAST BLAS
version: 1.9.0.dev-f80ccb0
maxint: 9223372036854775807

dot: 0.0710162162781 sec

# OPENBLAS_NUM_THREADS=8 python numpy/build/test_numpy.py
FAST BLAS
version: 1.9.0.dev-f80ccb0
maxint: 9223372036854775807

dot: 0.0227558135986 sec
```

2.2.4 SciPy

We proceed now to the installation of the scipy. The installation instructions are as follows:

```
$ git clone https://github.com/scipy/scipy
$ cd scipy
```

Edit site.cfg file so that:

```
[DEFAULT]
library_dirs = /usr/local/lib/
include_dirs = /usr/local/include/
```

and then:

```
$ python setup.py config
$ python setup.py build
# python setup.py install
```

test scipy with this:
https://gist.github.com/osdf/3842524#file_test_scipy.py

```
# python test_scipy.py
cholesky: 0.0181890010834 sec
svd: 0.37500538826 sec
```

2.2.5 Nose

Proceed similarly for the installation of the nose package:

```
$ git clone https://github.com/nose-devs/nose/
$ python setup.py config
$ python setup.py build
# python setup.py install
```

2.2.6 Theano

Finaly, theano:

```
git clone https://github.com/Theano/Theano.git
git pull
python setup.py develop
```

One simple test:

```
$ python ‘python -c "import os, theano; print os.path.dirname(theano
    .__file__)"‘/misc/check_blas.py
```

And now the final tests. Do not compare with the results from theano website. With different numpy/scipy versions, new tests are continuously being added. You should obtain the following times for each package test.

```
NumPy (~30s): python -c "import numpy; numpy.test()"
SciPy (~132s): python -c "import scipy; scipy.test()"
Theano (~30m): python -c "import theano; theano.test()"
```

.theanorc configuration flags (/etc/theanorc.theanorc)

```
# if device is cpu, use floatX=float64
# otherwise leave it as floatX=float32

[global]
device   = gpu
floatX   = float32
cxx      = g++
mode     = FAST_RUN
optimizer = fast_run
linker   = cvm_nogc
allow_gc = False
base_compiledir = /tmp/theanocompiledir_rsousa/

[gcc]
cxxflags = -O3 -ffast-math

[unittests]
rseed   = 0
```

```
[cuda]
root = /usr/local/cuda/

[nvcc]
fastmath = True

# for cpu execution
[blas]
ldflags = -L/usr/local/lib/ -lopenblas -lgfortran -lpthread -lm
```

2.2.7 Matplotlib

It is time to install matplotlib, but first we need some libraries:
`# apt-get install libfreetype6-dev libpng12-dev`

Now we are ready to proceed with the installation of the matlaplotlib

```
$ git clone https://github.com/matplotlib/matplotlib
$ cd matplotlib
$ python setup.py build
# python setup.py install
```

We may need to install some dependencies:
`# pip install distribute`

2.3 BLAS and Theano Benchmark

We should check if our installation was conducted properly. For this, please conduct the following benchmarks. For theano with BLAS bindings:

```
python ‘python -c "import os, theano; print os.path.dirname(theano.
    __file__)"‘/misc/check_blas.py

... with GPU ...
Total execution time: 0.11s on GPU.
... CPU, with one thread ...
Total execution time: 5.41s on CPU (with direct Theano binding to
    blas).
... CPU, with eight threads ...
Total execution time: 1.47s on CPU (with direct Theano binding to
    blas).
```

floatX=float32 and with floatX=float64 makes an huge difference, so carefully set your settings for your experiments.

```
$ python check1.py
Using gpu device 0: GeForce GTX 770
[Elemwise{exp,no_inplace}(<TensorType(float64, vector)>)]
Looping 1000 times took 3.87985801697 seconds
Result is [ 1.23178032 1.61879341 1.52278065 ..., 2.20771815
    2.29967753
  1.62323285]
Used the cpu
$ python check1.py
Using gpu device 0: GeForce GTX 770
[GpuElemwise{exp,no_inplace}(<CudaNdarrayType(float32, vector)>),
    HostFromGpu(GpuElemwise{exp,no_inplace}.0)]
Looping 1000 times took 0.287859916687 seconds
Result is [ 1.23178029 1.61879349 1.52278066 ..., 2.20771813
    2.29967761
  1.62323296]
Used the gpu
```

The same behavior is obtained when you use CPU.

```
$ python check1.py
[Elemwise{exp,no_inplace}(<TensorType(float32, vector)>)]
Looping 1000 times took 44.3791029453 seconds
Result is [ 1.23178029 1.61879337 1.52278066 ..., 2.20771813
    2.29967761
  1.62323284]
Used the cpu
$ python check1.py
[Elemwise{exp,no_inplace}(<TensorType(float64, vector)>)]
Looping 1000 times took 4.33928704262 seconds
Result is [ 1.23178032 1.61879341 1.52278065 ..., 2.20771815
    2.29967753
  1.62323285]
Used the cpu
```

with floatX=float64 (float32 raises an error)

```
$ python check3.py
Using gpu device 0: GeForce GTX 770
time spent evaluating both values 0.170000 sec
time spent evaluating one value 0.080000 sec
```

Run check4.py and check5.py to check if the results are the same.

```
$ python check6.py
Using gpu device 0: GeForce GTX 770
[GpuElemwise{exp,no_inplace}(<CudaNdarrayType(float32, vector)>)]
```

```
Looping 1000 times took 0.169833898544 seconds
Result is <CudaNdarray object at 0x6033fb0>
Numpy result is [ 1.23178029 1.61879349 1.52278066 ..., 2.20771813
    2.29967761
  1.62323296]
Used the gpu
```

Change to **Borrow=True**

```
rsousa@nnigroup:~/shared_folder/theano_simple_tests$ python check6.
    py
Using gpu device 0: GeForce GTX 770
[GpuElemwise{exp,no_inplace}(<CudaNdarrayType(float32, vector)>)]
Looping 1000 times took 0.0119159221649 seconds
Result is <CudaNdarray object at 0x72aee30>
Numpy result is [ 1.23178029 1.61879349 1.52278066 ..., 2.20771813
    2.29967761
  1.62323296]
Used the gpu
```

```
python logistic_sgd.py
....
Optimization complete with best validation score of 7.500000 %,with
    test performance 7.489583 %
The code run for 74 epochs, with 28.682171 epochs/sec
The code for file logistic_sgd.py ran for 2.6s
```

First, download the dataset.

```
git clone https://github.com/yaoli/GSN
http://deeplearning.net/data/mnist/mnist.pkl.gz
```

```
$ python run_gsn.py
.... < some compilation messages > ...
1 Train : 0.60719 Valid : 0.366995 Test : 0.364948 time : 14.83045
    ...
2 Train : 0.304251 Valid : 0.277751 Test : 0.27737  time : 14.95894
    ...
3 Train : 0.285826 Valid : 0.267526 Test : 0.268914 time : 15.27624
    ...
4 Train : 0.268297 Valid : 0.264869 Test : 0.266583 time : 15.09365
    ...
5 Train : 0.266338 Valid : 0.264188 Test : 0.265794 time : 15.19850
    ...
```

2.4 Other resources

If you wish, you can use the up-to-date pytest scripts that will bind more properly with theano.

```
apt-get install mercurial-git
```

Clone pytest

```
hg clone https://bitbucket.org/hpk42/pytest/
python setup.py install
```

Chapter 3

Classifier Transfer Learning: A Survey Towards A Unifying View

3.1 Introduction

One common difficulty arising in practical machine learning applications is the need to redesign the classifying machines (classifiers) whenever the respective probability distributions of inputs and outputs change, even though they may relate to similar problems. For instance, classifiers that perform recommendations of consumer items for the Amazon website cannot be straightforwardly applied to the IMDB website [4, 11]. In the same way, text classification for the Wikipedia website may not perform appropriately when applied to the Reuters website, even though the texts of one and the other are written in the same idiom with only moderate changes of the text statistics. The *reuse* of a classifier designed for a given (*source*) problem on another (*target*) problem, presenting some similarities with the original one, with only minor operations of parameter tuning, is the scope of Transfer Learning (TL).

The following aspects have recently contributed to the emergence of TL:

– Considerable amount of unlabeled data: TL relaxes the necessity of obtaining large amounts of labeled data for new problems [1, 4]. TL can be advantageous since unlabeled data can have severe implications in some fields of research, such as in the biomedical field [32, 51];

– Good generalization: TL often produces algorithms with good generalization capability for different problems [4, 27];

– Less computational effort: TL provides learning models (classifier models) that can be applied with good performance results in different problems and far less computational effort (see e.g., [4, 24, 25]).

The following definitions clarify what we believe should be meant by TL. We use “model” as a general designation of a classifier or regressor, although in the present Chapter we restrict ourselves to the reusability of classifiers.

Definition 1 (MK): *Model Knowledge, or simply* knowledge *when no confusion arises, means the functional form of a model and/or a subset of its parameters.*

Definition 2 (TL): *TL is a Machine Learning (ML) research field whose goal is the development of algorithms capable of transferring knowledge from the source problem model in order to build a model for a target problem.*

Although (portions of) these ideas have been around in the literature [51], it has never been clearly defined as **Definition 2**. Most of the times the concept of TL has been mixed in the literature with active, online [7] and even sequential learning [7, 40]; also, concepts from classical statistical learning theory have not been used to properly define all possible TL scenarios. TL in fact encompasses ideas from areas such as *dataset shift* [40] where the distribution of the data can change over time, and to which sequential algorithms may be applied; or *covariate shift* [4] where data distributions of two problems differ but share the same decision functions. Overall, the aforementioned discrepancies (e.g., terminology mixture) contribute to obscuring the TL field and hindering its proper consideration.

The Chapter is structured as follows: Section 3.2 presents an historical overview on the developments of TL; Section 4.2 describes the fundamental concepts of TL and their derivations followed by our proposal towards the unification of TL. Section 3.4 concludes this manuscript by summing up and discussing all main ideas.

3.2 A Review on TL

TL has been around since the 80’s with considerable advancements since then (see e.g., [4, 13, 20, 33, 37, 46, 47] and references therein). Probably, the foremost work that envisioned the concept of TL was the one of Mitchel in [33] where the idea of bias learning was presented.

An early attempt to extend these ideas was soon performed by Pratt *et al.* in [39] where Neural Networks (NNs) were used for TL. In a simple way, layer weights trained for the source problem were reused and retrained to solve the target problem. At the time, Pratt and her collaborators [39] adopted entropy measures to assess the quality of the hyperplanes tailored for the target problem and to define stopping criteria for training NNs for

TL. Soon after, Intrator [23] derived a framework to use (abstract) internal representations generated by NNs on the source problem to solve the target problem.

After these pioneer works, a significant number of implementations and derivations of TL started to appear. In [45] a new learning paradigm was proposed for TL where one would incrementally learn concept after concept. Thrun [45] envisioned this approach to how humans learn: by stacking knowledge upon another (as building blocks) resulting in an extreme nested system of learning functions. At that time, a particular case of [45], coined Multi-Task Learning (MTL), was presented [3, 12]. However, this approach does not hold for our definition of TL (see **Definition 2**) since it learns a common representation for all problems (multiple target problems addressed simultaneously).

In the year 2000 the concept of *covariate shift* was introduced by Shimodaira [43]. Although initially not contextualized in the domain of TL, his theoretical conclusions on how to learn a regression model on a target problem based on a source problem had a significant impact later to be realized. Shimodaira described a weighted least squares regressor based on the prior knowledge of the densities of source and target problems. At the time, he only addressed the case of marginal distributions being different and equal posterior leading to what he termed *covariate shift*. Other authors [5, 13, 16] followed with different algorithms to address the limitations of Shimodaira's [43] work such as the estimation of data densities leading to the rise of the *domain adaptation*.[1] In the work of Sugiyama *et al.* [44] an extension of Shimodaira's work was presented so that it could cope with the leave-one-out risk. The success of *covariate shift* is mostly associated to solving many Natural Language Processing (NLP) problems [5, 8, 14, 18]. In fact, many other works were dedicated to this subject–see e.g., [9, 11, 21, 22, 29, 42, 53] and references therein. Recently, an overview on TL was presented by Pan *et al.* in [36] with a vast but horizontal analysis of the most recent works that tackle classification, regression and unsupervised learning for TL. Orabona and co-workers [27] provided fundamental mathematical reasonings for TL by devising: 1) generalization bounds for max-margin algorithms such as SVMs and 2) their theoretical bounds based on the leave-one-out risk [10]. This was afterwards extended by Ben-David *et al.* in [6]. The work of Orabona *et al.* [27] was the first to identify a gap in the literature of the theoretical limitations of algorithms on TL.

The majority of the aforementioned works assume equal number of classes

[1]Domain adaptation has similar principles to *covariate shift*. We stick to the latter designation.

both for source and target problems. A significant contribution to the unconstrained scenario of the class set was presented in [46] expanding the work of Thrun [45].

With the recent re-interest on NNs and the availability of more computational power along with new and faster algorithms, NNs with deep architectures started to emerge to tackle TL. In [20] a framework for *covariate shift* with deep networks was presented; In [1, 2, 24, 25] the research line of [39] was widened by addressing the following questions: How can one tailor Deep Neural Networks (DNNs) for TL? How does TL perform by reusing layers and using different types of data?

The immense diversity of TL interpretations and definitions gave rise to concerns on how to unify this area of research. To this respect, Patricia *et al.* proposed in [37] an algorithm to solve *covariate shift* and other types of TL settings. In what follows we present a theoretical framework that ties together most of the work presented so far on TL for classification problems.

3.3 Transfer Learning

3.3.1 Classification: Notation and Problem Setting

A dataset represented by a set of tuples $D = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$ is given to a classification learning machine. The set $\boldsymbol{X}_N = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\}$ contains N instances (realizations) of a random vector $\boldsymbol{X}$ whose codomain is $\boldsymbol{X} = \mathbb{R}^d$; it will be clear from the context if $\boldsymbol{X}$ denotes a codomain or a random vector. Any instance $\boldsymbol{x} \in \boldsymbol{X}$ is a d-dimensional vector of real values $\boldsymbol{x} = [x_1, x_2, \dots, x_j, \dots, x_d]^t$. Similarly, the set $\boldsymbol{Y}_N = \{y_1, \dots, y_N\}$ contains N instances of a one-dimensional random variable $\boldsymbol{Y}$ whose codomain is w.l.o.g. $\boldsymbol{Y} = \mathbb{Z}$, coding in some appropriate way the labels of each instance of $\boldsymbol{X}_N$ [31].

The $(\boldsymbol{x}_i, y_i)$ tuples are assumed to be drawn i.i.d. according to a certain probability distribution $\mathcal{P}(\boldsymbol{x}, y)$ on $\boldsymbol{X} \times \boldsymbol{Y}$ [15, 31, 48]. We also have a *hypothesis space* $\mathcal{H}$ consisting of functions $h : \boldsymbol{X} \to \boldsymbol{Y}$ and a loss function $L(y, h(\boldsymbol{x}))$ quantifying the deviation of the predicted value of $h(\boldsymbol{x})$ from the true value y. For a given loss function one is able to compute the average loss, or classification risk, $R(h)$, as an expected value of the loss. For absolutely continuous distributions on $\boldsymbol{X}$ the classification risk (a functional of

h) is then written as:

$$\begin{aligned} R(h) &\equiv \mathbb{E}_{\boldsymbol{X}\times\boldsymbol{Y}}[L(\boldsymbol{Y}, h(\boldsymbol{X}))] \\ &= \sum_{y\in\boldsymbol{Y}} \int_{\boldsymbol{X}} \mathcal{P}(\boldsymbol{x}, y) L(y, h(\boldsymbol{x})) d\boldsymbol{x}. \end{aligned} \tag{3.1}$$

For discrete distributions on $\boldsymbol{X}$ the classification risk is:

$$R(h) \equiv \mathbb{E}_{\boldsymbol{X}\times\boldsymbol{Y}}[L(\boldsymbol{Y}, h(\boldsymbol{X}))] = \sum_{y\in\boldsymbol{Y}} \sum_{\boldsymbol{x}\in\boldsymbol{X}} \mathcal{P}(\boldsymbol{x}, y) L(y, h(\boldsymbol{x})). \tag{3.2}$$

Our aim is to derive a hypothesis $h(\boldsymbol{x})$ that minimizes $R(h)$. In common practice $\mathcal{P}(\boldsymbol{x}, y)$ is unknown to the learner. Therefore, $R(h)$ would be estimated using Eq. (3.1) or Eq. (3.2) above using an estimate of $\mathcal{P}(\boldsymbol{x}, y)$. As an alternative, one could also opt to minimize an empirical estimate of the risk, $\widehat{R}(h)$, $\widehat{R}(h) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, h(\boldsymbol{x}_i))$. Note that if we use an indicator loss function, $L(y, h(\boldsymbol{x})) = \mathbb{1}_{y\neq h(\boldsymbol{x})}$, the risk given by Eq. (3.1) or Eq. (3.2) corresponds to the probability of error. Finding the function h that minimizes $R(h)$ corresponds then to finding the hypothesis — also known as decision function — that minimizes the probability of error. Similarly, $\widehat{R}(h)$ corresponds to an empirical estimate of the probability of error. Finally, when minimizing Eq. (3.1), h is given according to a parametric form $h(\boldsymbol{x}, \boldsymbol{w})$ with $\boldsymbol{w} \in A, A \subset \mathbb{R}^n$. Finding the appropriate function means finding its corresponding parameters [48]. When clear from the context we will omit $\boldsymbol{w}$ to define the hypothesis $h(\boldsymbol{x})$.

3.3.2 TL: A Unifying View

The first requirement is to set the scope of TL. To assess if we can perform TL we need to know the data distribution of our source, $\mathcal{P}_S$, and target, $\mathcal{P}_T$, problems. We will use subscript S and T to refer to the source and target problems, respectively. Suppose we have two functions, h_S^* and h_T^*, each one solving by Direct Learning (DL) the source and target problems, respectively. While DL uses a random initial parameterization, TL, by contrast, uses h_S^* as a seed to reach h_T^*. This is illustrated in Figure 3.1.

In the work of Shimodaira [43] a weighted maximum likelihood estimate is devised to handle different data distributions for regression problems. One can assess this issue theoretically by deriving the target risk w.r.t. the source hypothesis as follows. We start by writing down the risk of the source problem:

$$\mathbb{E}_S[L(\boldsymbol{Y}, h_S(\boldsymbol{X}))] = \sum_{y\in\boldsymbol{Y}} \int_{\boldsymbol{X}} \mathcal{P}_S(\boldsymbol{x}, y) L(y, h_S(\boldsymbol{x})) \mathrm{d}\boldsymbol{x}, \tag{3.3}$$

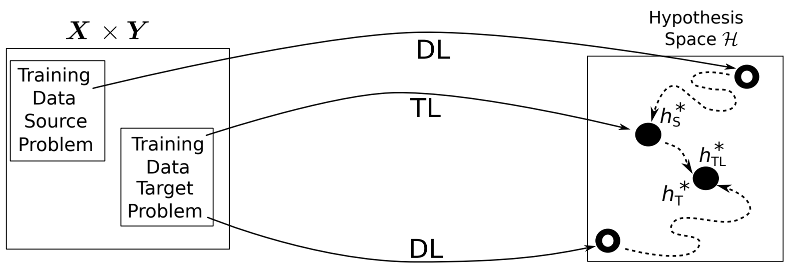
$\boldsymbol{X} \times \boldsymbol{Y}$
Training Data Source Problem
Training Data Target Problem
DL
TL
DL
Hypothesis Space $\mathcal{H}$
h_S^*
h_{TL}^*
h_T^*

Figure 3.1: A representation of Transfer Learning (TL) when we have the same number of classes on both source and target problems. When directly solving the source and target problems — Direct Learning (DL) — one usually starts with a random parameterization (open bullets) until attaining the optimal solution, h_S^* or h_T^*, respectively. TL uses h_S^* as a starting point to reach h_T^*, instead of a random parameterization.

where $\mathbb{E}_S[L(\boldsymbol{Y}, h_S(\boldsymbol{X}))]$, $\mathbb{E}_S$ for short, is the same as $\mathbb{E}_{\boldsymbol{X}\times\boldsymbol{Y}}$ under the distribution $\mathcal{P}_S(\boldsymbol{x}, y)$, and we assume that a one-to-one correspondence between source and target spaces exists, denoting the common space by $\boldsymbol{X} \times \boldsymbol{Y}$. The risk minimization process will select an optimal hypothesis, h_S^*, with a minimum risk:

$$\begin{aligned}
R_S(h_S^*) &\equiv \mathbb{E}_S[L(\boldsymbol{Y}, h_S^*(\boldsymbol{X}))] \\
&= \sum_{y\in\boldsymbol{Y}} \int_{\boldsymbol{X}} \mathcal{P}_S(\boldsymbol{x}, y) L(y, h_S^*(\boldsymbol{x}))\mathrm{d}\boldsymbol{x} \\
&= \sum_{y\in\boldsymbol{Y}} \int_{\boldsymbol{X}} \frac{\mathcal{P}_S(\boldsymbol{x}, y)}{\mathcal{P}_T(\boldsymbol{x}, y)} \mathcal{P}_T(\boldsymbol{x}, y) L(y, h_S^*(\boldsymbol{x}))\mathrm{d}\boldsymbol{x} \\
&= \mathbb{E}_T\left[\frac{\mathcal{P}_S(\boldsymbol{X}, \boldsymbol{Y})}{\mathcal{P}_T(\boldsymbol{X}, \boldsymbol{Y})} L(\boldsymbol{Y}, h_S^*(\boldsymbol{X}))\right] \equiv R_T^W(h_S^*), \qquad (3.4)
\end{aligned}$$

where W stands for weighted. The equations for discrete distributions are obtained by simple substitutions of the integrals by the appropriate summations. Assessing the TL advantage corresponds to assessing a "distance" between the initial solution with risk $h_S^*(\boldsymbol{x})$ and the optimal solution one would obtain by DL (see Figure 3.1). An appropriate "distance" is the deviation of the corresponding risks. How to assess TL has been intensively pursued in psychology. Of most interest is how to measure gains of TL. This phenomenon was addressed in [38] where concepts of positive and negative transfer are introduced.

Definition 3 (Transference): *Transference is a property that allow us to measure the effects of a TL framework (e.g., performance).*

– Positive Transference: *is a TL method that results on improving the performance on a target problem w.r.t. the best model obtained by DL on the target problem;*

– Negative Transference: *the opposite of positive transference that is, when a TL method results on degrading performance on a target problem.*

3.3.3 Categorizations of TL

Based on the joint probability and by the Bayes rule,

$$\mathcal{P}(\boldsymbol{x}, y) = \mathcal{P}(y|\boldsymbol{x})\mathcal{P}(\boldsymbol{x}), \tag{3.5}$$

for the source and target problems, we now generate all TL possibilities. This choice is related to the decision functions and data distribution changes that can occur on the source and target problems. Take the example of *covariate shift*. If we generate datasets for the source and target marginal distributions, $\mathcal{P}_S(\boldsymbol{x})$ and $\mathcal{P}_T(\boldsymbol{x})$, according to two Gaussian distributions with different means and covariances, and superimpose the same decision function such that a coding y for an instance, $\boldsymbol{x}$, $\boldsymbol{x} \in \mathbb{R}^2$, is assigned according to the rule $d = (x_1 - 0.5)(x_2 - 0.5)$ with $y = -1, d < -1$ and $y = +1$ otherwise, we are then able to obtain the source and target datasets shown in Figure 3.2.

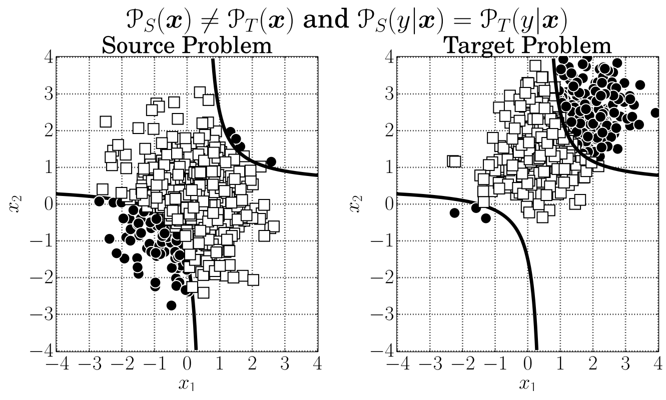

Figure 3.2: A possible scenario for *covariate shift*. Data for the marginal distributions $\mathcal{P}_S(\boldsymbol{x})$ and $\mathcal{P}_T(\boldsymbol{x})$ was generated from two Gaussian distributions with $\mu_S = (0, 0)^t$ and $\Sigma_S = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ and $\mu_T = (1, 2)^t$ and $\Sigma_T = \begin{bmatrix} 1 & 0.2 \\ 0.8 & 1 \end{bmatrix}$. The following decision rule was superimposed: $d = (x_1 - 0.5)(x_2 - 0.5), y = -1, d < -1$ and $y = +1$ otherwise.

Inspired by the covariate shift setting, we may then impose conditions on the marginal and posterior distributions (Eq. (3.5)), in order to arrive at all possible TL categories presented in Table 3.1. Note that under a practical perspective it makes sense to use marginal and posterior distributions for TL categorization. Marginal distributions are easy to estimate and histogram inspection may hint of whether or not the posteriors are the same. A practical assessment of the TL category at hand is then achievable.

Table 3.1: Multiple possibilities for having TL: based on the equality ($\mathcal{P}_S(\boldsymbol{x}, y) = \mathcal{P}_T(\boldsymbol{x}, y)$) or not ($\mathcal{P}_S(\boldsymbol{x}, y) \neq \mathcal{P}_T(\boldsymbol{x}, y)$) of the source and target problems.

	$\mathcal{P}_S(\boldsymbol{x}, y) = \mathcal{P}_T(\boldsymbol{x}, y)$	$\mathcal{P}_S(\boldsymbol{x}, y) \neq \mathcal{P}_T(\boldsymbol{x}, y)$
$\mathcal{P}_S(\boldsymbol{x}) = \mathcal{P}_T(\boldsymbol{x})$ and $\mathcal{P}_S(y\|\boldsymbol{x}) = \mathcal{P}_T(y\|\boldsymbol{x})$	**No TL**: everything is the same	**No TL**: Impossible
$\mathcal{P}_S(\boldsymbol{x}) \neq \mathcal{P}_T(\boldsymbol{x})$ and $\mathcal{P}_S(y\|\boldsymbol{x}) = \mathcal{P}_T(y\|\boldsymbol{x})$	**No TL**: Impossible	**TL**: Covariate Shift
$\mathcal{P}_S(\boldsymbol{x}) = \mathcal{P}_T(\boldsymbol{x})$ and $\mathcal{P}_S(y\|\boldsymbol{x}) \neq \mathcal{P}_T(y\|\boldsymbol{x})$	**No TL**: Impossible	**TL**: Response Shift
$\mathcal{P}_S(\boldsymbol{x}) \neq \mathcal{P}_T(\boldsymbol{x})$ and $\mathcal{P}_S(y\|\boldsymbol{x}) \neq \mathcal{P}_T(y\|\boldsymbol{x})$	**No TL**: Valid but TL only for a few particular cases	**TL**: Complete Shift

It is now clear that if $\mathcal{P}_S(\boldsymbol{x}, y) = \mathcal{P}_T(\boldsymbol{x}, y)$ there is no reason to perform TL. The interesting TL settings correspond to $\mathcal{P}_S(\boldsymbol{x}, y) \neq \mathcal{P}_T(\boldsymbol{x}, y)$ leading to three possible categories:

Covariate Shift: $\mathcal{P}_S(\boldsymbol{x})$ is different from $\mathcal{P}_T(\boldsymbol{x})$ and $\mathcal{P}_S(y|\boldsymbol{x})$ equal to $\mathcal{P}_T(y|\boldsymbol{x})$

Response Shift: $\mathcal{P}_S(\boldsymbol{x})$ is equal to $\mathcal{P}_T(\boldsymbol{x})$ and $\mathcal{P}_S(y|\boldsymbol{x})$ different from $\mathcal{P}_T(y|\boldsymbol{x})$

Complete Shift: $\mathcal{P}_S(\boldsymbol{x})$ is different from $\mathcal{P}_T(\boldsymbol{x})$ and $\mathcal{P}_S(y|\boldsymbol{x})$ different from $\mathcal{P}_T(y|\boldsymbol{x})$

Different attempts have been made to stratify these approaches [16, 36, 42]. While this stratification can lead to different terminologies, attempts to bring them together seem insufficient. In a recent work, Zhang *et al.* [52] have proposed to categorize Eq. (3.4) but based on the class prior, $\mathcal{P}(y)$, and likelihood, $\mathcal{P}(\boldsymbol{x}|y)$, distributions. Their rationale is not related to the decision surfaces as ours leading to a different analysis. Moreover, they require further assumptions on the data distributions [52] than the ones presented in this Chapter.

3.4 Conclusions

A survey towards a unifying formalism for TL has been presented in this Chapter. We conducted a review of classical and state-of-the-art work on TL by enumerating key aspects of each one. Based on the work in covariate shift we could devise three categories of TL: *covariate shift*, *response shift* and *complete shift*.

Chapter 4

Automatic Detection of Immunogold Particles from Electron Microscopy Images

4.1 Introduction

Immunogold electron microscopy is a high-resolution method for the selective localization of biological molecules at the subcellular level. Antibodies coupled to particles of colloidal gold, which are visible in the transmission electron microscope, can reveal the localization and distribution of the biological molecules of interest. In this particular work, this technique was used to determine the composition of cell wall uneven thickenings that ultimately differentiate into reticulate and flange ingrowths of maize (*Zea mays* L.) endosperm transfer cells [34]. These cells are essential for the assimilate flow into the endosperm, thus having a significant impact on kernel yield. Immunogold particle detection is a time-consuming task where a single image containing almost a thousand particles can take several hours to annotate [41] (Figure 4.1). An automatic detection tool can reduce the time consumed in such analysis and improve its accuracy.

In this Chapter we describe and compare two methods that permit the automatic detection of immunogold particles. We show that our method (Laplacian of Gaussian (LoG)) is tolerant to different sizes of immunogold particles and to noise that may occur. Our algorithm is compared to the well-known Spot Detector (SD) method available on the Icy bioimaging software [17] and outperforms it in most tested situations.

This Chapter is structured as follows: we start by briefly describing previous works on this subject immediately followed by the description of our

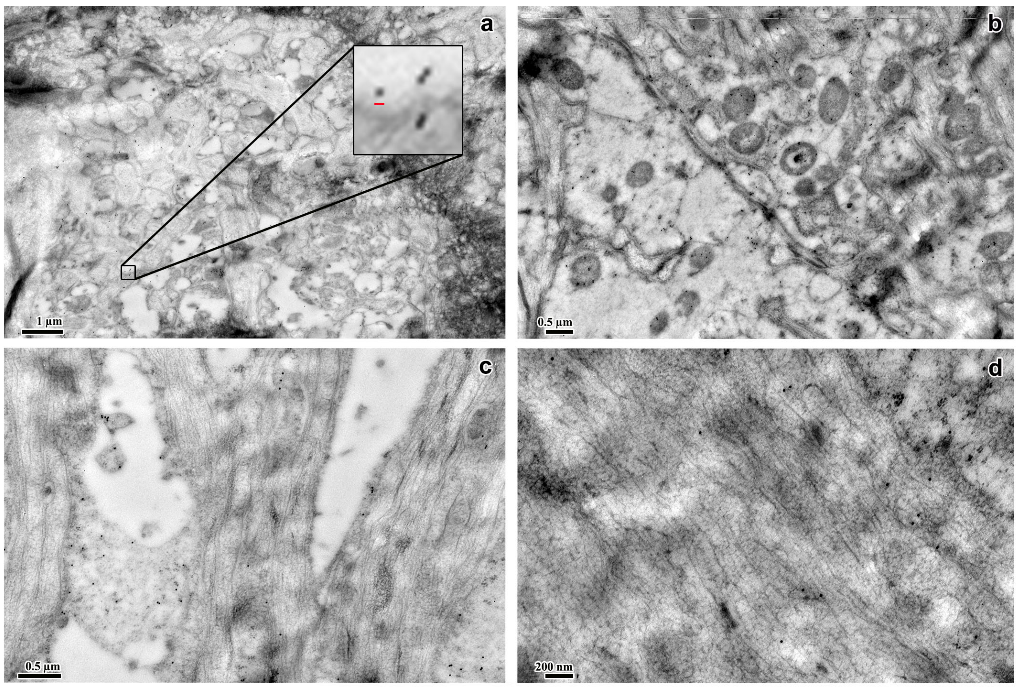

Figure 4.1: Representative images of our datasets illustrating different structures that can interfere in the detection of the immunogold particles due to: cellular overlapping, tissues and background noise. Each image has 4000×2600 pixels of dimension with particles *diameter* ranging from 8 to 20 pixels. **(a)** Example of a sample with a magnification of 15000 ($1\mu m$, particles with a diameter of 8 pixels—red line); **(b)** magnification of 20000 ($0.5\mu m$, 12 pixels diameter particles); **(c)** magnification of 30000 ($0.5\mu m$, 15 pixels diameter particles); and, **(d)** magnification of 50000 ($200nm$, 20 pixels diameter particles).

proposal (see Section 4.2).

4.2 Immunogold Particles Detection

Fisker *et al.* in [19] explores the possibility to automatically estimate particle sizes in immuno-microscopy imaging. Their approach is based on deformable models that can be fitted to the prior known shape of the particles. With the same goal as Fisker, a different approach was presented by Mallic *et al.* in [30] by using cascade of classifiers. Inspired in the conventional face detection challenges where recognition with ensemble of weak classifiers have proven to be effective, this straightforward strategy may be computationally complex due to the amount of immunogold particles that may occur in electron microscopy images. In [41] an insight review is presented by motivating the advantages of Computer Vision (CV) for the processing of images generated by electron microscopies. Finally, in [49, 50] a Difference of Gaussian (DoG) and LoG filters are applied to aid the detection of particles (e.g., organelles) on cryo-electron microscopy images. In these works DoG or LoG were used as a first step to detect more complex biological structures and were not tailored neither evaluated on immunogold particles.

A major difference is that in the aforementioned proposals all particles structures were shallow, irregular in shapes and intensities. Our work will be focused on the detection of immunogold particles with regular spherical shape, thus avoiding the adoption of a highly parameterized formalism for its detection.

4.2.1 Icy bioimaging software: Spot Detector

For the detection of biological structures, there is the publicly available SD [35] algorithm that is included in the well-known Icy bioimaging software [17]. Icy (in short) is an open source software with resources to visualize, annotate and quantify bio-imaging data. The SD enables the detection of spots that can organelles or other biological similar structures in noise images 2D or 3D and is based on the non-decimated wavelet transform [35]. This approach aggregates a response for each resolution and scale of the image providing detailed information of the objects. As a generic form of spot detection it includes a set of parameters that need to be defined for an appropriate detection. It requires the identification of a trade-off between particles and background; the definition of a scale and sensibility that controls both size of the particles to be detected and a threshold for noise removal.

4.2.2 Immunogold Particles Detection using LoG Filter

For the task of immunogold particles detection we used the LoG filter, which is based on the image scale-space representation to enhance the blob like structure as introduced by Lindeberg [28]. Given an input image $I(x, y)$, the Gaussian scale space representation at a certain scale t is:

$$L(x, y, t) = g(x, y, t) * I(x, y), \text{ where } g(x, y, t) = \frac{1}{2\pi t} e^{\frac{x^2+y^2}{2t}}, \tag{4.1}$$

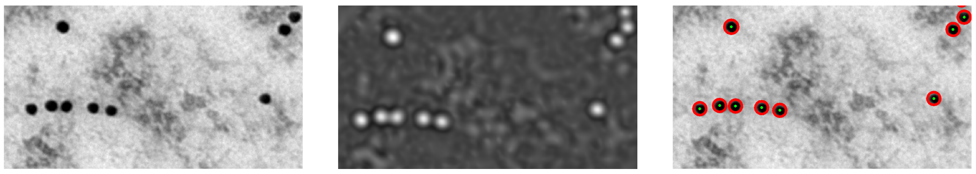

Figure 4.2: LoG based cell detection: left) Original image (crop from an image with magnification of 50000); center) LoG response; right) Detections overlaid in the original image.

where $*$ is the convolution operation. The scale normalized LoG operator is then defined as:

$$\nabla^2 L(x, y, t) = t^2(L_{xx}(x, y, t) + L_{yy}(x, y, t)), \tag{4.2}$$

where L_{xx} and L_{yy} are the second derivatives of the input image in x and y respectively, and t is the scale parameter so that $t = r/1.5$ for a particle radius r [28]. We set the scale of the filter (t) given the expected range of the immunogold particles radius (Figure 4.1). We perform detection of immunogold particles by detecting local maxima of LoG response (Figure 4.2 - center) in the input image (Figure 4.2 - left). The detected maxima enable us to estimate the position of immunogold particles (Figure 4.2 - right).

4.3 Conclusion

We have presented a study on different methodologies for automatic detection of immunogold particles in different magnifications. We found that the LoG automatic detection algorithm outperformed in the almost tested scenarios the SD approach. In the future we will study the applicability of Deep Neural Networks [1, 26] for nanoparticle detection.

Chapter 5

Conclusion and Future work

During this grant we have addressed two important subjects: (a) the proposal of a clear framework for Transfer Learning (TL) and (b) the study of the detection of immunogold nanoparticles.

This pivotal work allowed us to identify a significant amount of opportunities for Machine Learning (ML) and Computer Vision (CV) methodologies with applications on several biomedical fields. Thereby, although not initially foreseen, we had also the opportunity to propose a new research project that extends and widens the research lines that we have initiated. In summon, we have proposed to develop an intelligent system capable of processing and analyzing Transmission Electron Microscopy (TEM) images of the Nervous System (NS) and Plants (PTs). We have identified the importance of this system for helping in understanding the underlying biological principles and processing massive data storages. Cellular structure and ultrastructure (the latter only seen through TEM imaging) was identified to be explored in order to devise Deep Neural Networks (DNNs) capable of automatically extracting information that contribute into new approaches for the analysis of the NS in development and disease paradigms, improvement of plant yield and plant response to abiotic stress. Ultimately, this framework can provide a ubiquitous support for TEM technology by conveying a fundamental intelligent system to answer pressing biological matters with significant social and economic impact.

5.1 Future Work

In the future, we will continue to research new methodologies to improve accuracy of the DNNs on biomedical problems. More concretely, we will apply the knowledge gained so far:

1. To devise and assess the impact of TL algorithms for the recognition of immunogold particles in Maize images acquired through TEM with different magnifications;
2. Towards the development of new TL methodologies capable of coping new (bigger) datasets for the analysis of images of *Collagen*;
3. For the morphometric analysis of microglia cells using Stacked Denoising Autoencoders (SDAs);
4. To develop new image processing algorithms for the analysis of images of *C.elegans*.

Appendix A

Appendix

A.1 Acronyms

CNN	Convolutional Neural Network	3
CV	Computer Vision	27
DL	Direct Learning	
DNN	Deep Neural Network	27
DoG	Difference of Gaussian	25
HPC	High Performance Computer	3
IF	Impact Factor	3
LoG	Laplacian of Gaussian	23
ML	Machine Learning	27
MTL	Multi-Task Learning	16

NN Neural Network 15

NLP Natural Language Processing 16

SAE Stacked Autoencoder 3

SDA Stacked Denoising Autoencoder 28

SD Spot Detector 23

TEM Transmission Electron Microscopy 27

TL Transfer Learning 27

Bibliography

[1] Telmo Amaral, Luis M. Silva, Luis M. Alexandre, Chetak Kandaswamy, Joaquim Marques de Sá, and Jorge Santos. Improving Performance on Problems with Few Labelled Data by Reusing Stacked Auto-Encoders. In *ICMLA*, 2014. accepted.

[2] Telmo Amaral, Luis M. Silva, Luis M. Alexandre, Chetak Kandaswamy, Joaquim Marques de Sá, and Jorge Santos. Transfer learning using rotated image data to improve deep neural network performance. In *ICIAR*, 2014.

[3] Jonathan Baxter. A model of inductive bias learning. *J. Artif. Intell. Res.(JAIR)*, 12(1):149–198, 2000.

[4] Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine Learning*, 79(1-2):151–175, October 2009.

[5] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of Representations for Domain Adaptation. *Advances in Neural Information Processing Systems*, 19:137, 2007.

[6] Shai Ben-David and Ruth Urner. Domain Adaptation as Learning with Auxiliary Information. *NIPS*, 2013.

[7] C Bishop. Pattern Recognition and Machine Learning, 2007.

[8] John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *ACL*, volume 7, pages 440–447, 2007.

[9] John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing - EMNLP '06*, page 120, 2006.

[10] Olivier Bousquet and André Elisseeff. Stability and generalization. *The Journal of Machine Learning Research*, 2:499–526, 2002.

[11] Lorenzo Bruzzone and Mattia Marconcini. Domain adaptation problems: a DASVM classification technique and a circular validation strategy. *IEEE transactions on pattern analysis and machine intelligence*, 32(5):770–87, May 2010.

[12] Caruana. Multitask Learning. *Machine Learning*, 75:41–75, 1997.

[13] Olivier Chapelle, Bernhard Schölkopf, Alexander Zien, et al. *Semi-supervised learning*, volume 2. MIT press Cambridge, 2006.

[14] Zhiyuan Chen and Bing Liu. Topic Modeling using Topics from Many Domains, Lifelong Learning and Big Data. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 703–711, 2014.

[15] Vladimir Cherkassky and Filip M Mulier. *Learning from data: concepts, theory, and methods.* John Wiley & Sons, 2007.

[16] Hal Daumé III and Daniel Marcu. Domain Adaptation for Statistical Classifiers. *J. Artif. Intell. Res.(JAIR)*, 26:101–126, 2006.

[17] Fabrice de Chaumont, Stéphane Dallongeville, Nicolas Chenouard, Nicolas Hervé, Sorin Pop, Thomas Provoost, Vannary Meas-Yedid, Praveen Pankajakshan, Timothée Lecomte, Yoann Le Montagner, et al. Icy: an open bioimage informatics platform for extended reproducible research. *Nature methods*, 9(7):690–696, 2012.

[18] Lixin Duan, Ivor W Tsang, and Dong Xu. Domain transfer multiple kernel learning. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(3):465–479, 2012.

[19] Rune Fisker, Jens Michael Carstensen, Mikkel Fougt Hansen, Franz Bødker, and Steen Mørup. Estimation of nanoparticle size distributions by image analysis. *Journal of Nanoparticle Research*, 2(3):267–277, 2000.

[20] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 513–520, 2011.

[21] Arthur Gretton, Alex Smola, Jiayuan Huang, Marcel Schmittfull, Karsten Borgwardt, and Bernhard Schölkopf. Covariate shift by kernel mean matching. *Dataset shift in machine learning*, 3(4):5, 2009.

[22] Jiayuan Huang, Arthur Gretton, Karsten M Borgwardt, Bernhard Schölkopf, and Alex J Smola. Correcting sample selection bias by unlabeled data. In *Advances in neural information processing systems*, pages 601–608, 2006.

[23] Nathan Intrator. Making a Low-dimensional Representation Suitable for Diverse Tasks. *Connection Science*, 8(2):205–224, 1996.

[24] Chetak Kandaswamy, Luis M. Silva, L. M. Alexandre, Jorge Santos, and JP Marques de Sá. Improving Deep Neural Network Performance by Reusing Features Trained with Transductive Transference. In *ICANN*, 2014.

[25] Chetak Kandaswamy, Luis M. Silva, L. M. Alexandre, Ricardo Sousa, Jorge Santos, and JP Marques de Sá. Improving Transfer Learning Accuracy by Reusing Stacked Denoising Autoencoders. In *Proceedings of the IEEE SMC Conference*, 2014.

[26] Chetak Kandaswamy, Luis M. Silva, L. M. Alexandre, Ricardo Sousa, Jorge Santos, and Joaquim Marques de Sá. Improving transfer learning accuracy by reusing stacked denoising autoencoders. In *Proceedings of the IEEE SMC Conference*, 2014.

[27] Ilja Kuzborskij and Francesco Orabona. Stability and hypothesis transfer learning. In *Proceedings of The 30th International Conference on Machine Learning*, pages 942–950, 2013.

[28] T. Lindeberg. Scale-space theory: A basic tool for analyzing structures at different scales. *Journal of Applied Statistics*, 21(2):224–270, 1994.

[29] Xiao Ling, Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. Spectral domain-transfer learning. *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08*, page 488, 2008.

[30] Satya P Mallick, Yuanxin Zhu, and David Kriegman. Detecting particles in cryo-em micrographs using learned features. *Journal of Structural Biology*, 145(1):52–62, 2004.

[31] JP Marques de Sá, Luís MA Silva, Jorge MF Santos, and Luís A Alexandre. *Minimum Error Entropy Classification.* Springer, 2013.

[32] Vivien Marx. Biology: The big challenges of big data. *Nature*, 498(7453):255–260, 2013.

[33] Tom M. Mitchell. The need for biases in learning generalizations. In Jude W. Shavlik and Thomas G. Dietterich, editors, *Readings in Machine Learning*, pages 184–191. Morgan Kauffman, 1980. Book published in 1990.

[34] Paulo Monjardino, Sara Rocha, Ana C Tavares, Rui Fernandes, Paula Sampaio, Roberto Salema, and Artur da Câmara Machado. Development of flange and reticulate wall ingrowths in maize (*Zea mays* L.) endosperm transfer cells. *Protoplasma*, 250(2):495–503, 2013.

[35] Jean-Christophe Olivo-Marin. Extraction of spots in biological images using multiscale products. *Pattern Recognition*, 35(9):1989–1996, 2002.

[36] Sinno Jialin Pan and Qiang Yang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, October 2010.

[37] Novi Patricia and Barbara Caputo. Learning to Learn, from Transfer Learning to Domain Adaptation: A Unifying Perspective. In *Proceedings of the Computer Vision and Pattern Recognition*, 2014.

[38] David N Perkins and Gavriel Salomon. Transfer of learning. *International encyclopedia of education*, 2, 1992.

[39] Lorien Y Pratt, LY Pratt, SJ Hanson, CL Giles, and JD Cowan. Discriminability-Based Transfer between Neural Networks. In S J Hanson, J D Cowan, and C L Giles, editors, *Advances in Neural Information Processing Systems 5*, pages 204–211, 1992.

[40] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. *Dataset shift in machine learning.* The MIT Press, 2009.

[41] Eraldo Ribeiro and Mubarak Shah. Computer vision for nanoscale imaging. *Machine Vision and Applications*, 17(3):147–162, 2006.

[42] Gabriele Schweikert, Gunnar Rätsch, Christian Widmer, and Bernhard Schölkopf. An empirical analysis of domain adaptation algorithms for

genomic sequence analysis. In *Advances in Neural Information Processing Systems*, pages 1433–1440, 2009.

[43] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

[44] Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert Müller. Covariate shift adaptation by importance weighted cross validation. *The Journal of Machine Learning Research*, 8:985–1005, 2007.

[45] S. Thrun. Is Learning the n-th Thing Any Easier Than Learning the First? In D. Touretzky and M Mozer, editors, *Advances in Neural Information Processing Systems*, pages 640–646, Cambridge, MA, 1996. MIT Press.

[46] T Tommasi, F Orabona, and B Caputo. Learning Categories from Few Examples with Multi Model Knowledge Transfer. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PP(99):1, 2013.

[47] V N Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 10(5):988–999, January 1999.

[48] Vladimir Vapnik. *The nature of statistical learning theory.* springer, 1995.

[49] NR Voss, CK Yoshioka, M Radermacher, CS Potter, and B Carragher. Dog picker and tiltpicker: software tools to facilitate particle selection in single particle electron microscopy. *Journal of Structural Biology*, 166(2):205–213, 2009.

[50] David Woolford, Ben Hankamer, and Geoffery Ericksson. The laplacian of gaussian and arbitrary z-crossings approach applied to automated single particle reconstruction. *Journal of structural biology*, 159(1):122–134, 2007.

[51] Liu Yang, Steve Hanneke, and Jaime Carbonell. A Theory of Transfer Learning with Applications to Active Learning. *Mach. Learn.*, 90(2):161–189, February 2013.

[52] Kun Zhang, Bernhard Sch, and Zhikun Wang. Domain Adaptation under Target and Conditional Shift. 28, 2013.

[53] Erheng Zhong, Wei Fan, Jing Peng, Kun Zhang, Jiangtao Ren, Deepak Turaga, and Olivier Verscheure. Cross domain distribution adaptation via kernel mapping. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1027–1036. ACM, 2009.