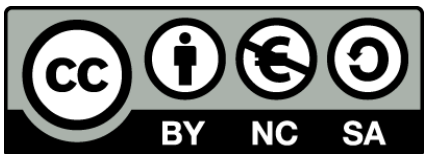
UNIVERSITAT DE BARCELONA

Assisting the training of deep neural networks with applications to computer vision

Adriana Romero



Assisting the training of deep neural networks with applications to computer vision

Adriana Romero

Department of Applied Mathematics and Analysis

University of Barcelona

A thesis submitted for the degree of

Doctor in Mathematics (PhD)

Doctoral advisor: *Dr. Carlo Gatta*

Tutor: *Dr. Petia Radeva*

September 2015

Abstract

Deep learning has recently been enjoying an increasing popularity due to its success in solving challenging tasks. In particular, deep learning has proven to be effective in a large variety of computer vision tasks, such as image classification, object recognition and image parsing. Contrary to previous research, which required engineered feature representations, designed by experts, in order to succeed, deep learning attempts to learn representation hierarchies automatically from data. More recently, the trend has been to go deeper with representation hierarchies. Learning (very) deep representation hierarchies is a challenging task, which involves the optimization of highly non-convex functions. Therefore, the search for algorithms to ease the learning of (very) deep representation hierarchies from data is extensive and ongoing.

In this thesis, we tackle the challenging problem of easing the learning of (very) deep representation hierarchies. We present a hyper-parameter free, off-the-shelf, simple and fast unsupervised algorithm to discover hidden structure from the input data by enforcing a very strong form of sparsity. We study the applicability and potential of the algorithm to learn representations of varying depth in a handful of applications and domains, highlighting the ability of the algorithm to provide discriminative feature representations that are able to achieve top performance.

Yet, while emphasizing the great value of unsupervised learning methods when labeled data is scarce, the recent industrial success of deep learning has revolved around supervised learning. Supervised learning is currently the focus of many recent research advances, which have shown to excel at many computer vision tasks. Top performing systems often involve very large and deep models, which are not well suited for applications with time or memory limitations. More in line with the current trends, we engage in making top performing models more efficient, by designing very deep and thin models. Since training such very deep

models still appears to be a challenging task, we introduce a novel algorithm that guides the training of very thin and deep models by hinting their intermediate representations. Very deep and thin models trained by the proposed algorithm end up extracting feature representations that are comparable or even better performing than the ones extracted by large state-of-the-art models, while compellingly reducing the time and memory consumption of the model.

Als meus pares i a la Valèria.

Acknowledgements

I would like to thank Dr. Carlo Gatta, my doctoral advisor, for taking me as his student and advising me over the past few years. It goes without saying that I learned a lot from him and that none of this would have been possible without his helpful advice and support. But it wouldn't be fair to stop here. I cannot be more grateful to Carlo for believing in me, for giving me the freedom to explore new ideas, for encouraging me to shift the initial direction of my research, and, most importantly, for sharing his contagious passion for research with me.

I would also like to show my appreciation to my tutor, Dr. Petia Radeva, for giving me the opportunity to pursue a PhD. Thank you Petia.

More broadly, I would like to thank many people at University of Barcelona and Computer Vision Center. Special thanks to Simeon Petkov for the many times we discussed utopian ideas; hopefully one day, the world will be a better place. But also thanks for digging deeper with me. Paradoxically, Insanity kept me sane over these years. Thanks to Camp Davesa, for being my unconditional gym partner. I would also like to thank many of my colleagues. Thanks to Víctor Ponce, Toni Hernández, Albert Clapés, Miguel Ángel Bautista, and Miguel Reyes for the mini-basket and the fun. Thanks to Carlos Domingo and Roc Oliver for the laughter and the gossips. It was a pleasure sharing the office with you. Thanks to Maya Aghaei for the great deal of time we spent together since our Masters and for being my friend; you will always find a yoga mat next to mine.

I would like to express my most sincere gratitude to Professor Yoshua Bengio, for warmly welcoming me to join his lab for a few months. My gratitude extends to all the members of LISA lab, who create an open and very intellectual atmosphere, where everyone is welcome to collaborate and discuss ideas. Although I would like to name all the amazing people I met there, I can only name a few. Special thanks to Dr. Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang and

Prof. Bengio for working closely with me and making FitNets possible. Thanks to Kyunghyun Cho for kindly providing some of the figures in this thesis. Thanks to Dustin Webb for the countless discussions and for being a friend. Thanks to Jason Yosinski, Yann Dauphin, Laurent Dinh, Zhouhan Lin, Harm de Vries, Vincent Michalski, Dustin Webb once again, and Samira Ebrahimi Kahou once again for all the fun. Thanks Frédéric Bastien and Pascal Lamblin for all the Theano and Pylearn2 support.

My friends deserve a nod as well. Special thanks to Laura for always finding the words to describe my research despite having a completely different background. But also to Natàlia, Sònia, Berta, Laia and Núria: thank you.

Thanks to my parents, Gilbert and Maria Antònia, for all the support, for the understanding, for raising me to achieve my goals and for giving me the best education I could ever dream of. But also thanks to my sister Valèria for the support, the occasional silliness, the laughter and the faith; I hope this thesis will encourage you to write yours in the near future.

Last but certainly not least, my gratitude goes for Michał, because most of my strength comes from you.

Contents

List of Figures

LIST OF FIGURES

List of Tables

Notation

The following list summarizes the basic notation that we will use in this thesis.

Architecture

$\mathbf{x}$	Input vector.
$\mathbf{w}$	Weights vector.
b	Bias.
l	Superscript indicating the l-th layer of a deep architecture.
L	Number of layers of the deep architecture.
$\mathbf{W}^l$	Weights of the l-th layer of a deep architecture: . For MLPs and patch-based training, $\mathbf{W}^l$ is a 2-D matrix. . For CNNs, $\mathbf{W}^l$ is a 4-D tensor.
$\mathbf{b}^l$	Biases vector of the l-th layer.
$\theta^l = \{\mathbf{W}^l, \mathbf{b}^l\}$	l-th layer parameters, composed of weights $\mathbf{W}^l$ and biases $\mathbf{b}^l$.
$\mathbf{a}^l$	Pre-activation vector of the l-th layer of a deep architecture.
$\mathbf{h}^l$	Activation vector of the l-th layer of a deep architecture.
$\mathbf{A}^l$	Activation map of l-th layer.
$\mathbf{H}^l$	Output feature map of l-th layer: . For patch-based training, $\mathbf{H}^l$ is a 2-D matrix. . For CNNs, $\mathbf{H}^l$ is a 3-D tensor.
R^l	In CNN, height of feature map $\mathbf{H}^l$.
C^l	In CNN, width of feature map $\mathbf{H}^l$.
$\mathrm{N}_{\mathrm{h}}^l$	Number of features in $\mathbf{H}^l$.
$\mathrm{w}_r^l \times \mathrm{w}_c^l$	Spatial size of layer l's filters.
$\star$	Convolution operator.
f	Non-linearity.
pool	Pooling operator.
R_{p}	Pooling region.

Feature Encoding

f_{enc}	Encoding function.
t	Threshold.
s	Sparse code.

Optimization

$\mathcal{D}$	Training set.
$\mathbf{x}^{(n)}$	Input sample indexed by n.
$\mathbf{y}^{(n)}$	Label of input sample indexed by n.
N	Number of samples.
$\mathrm{N_b}$	Number of samples in a mini-batch.
K	Number of classes.
$\mathcal{L}$	Loss function.
ϵ	Learning rate.

EPLS

c	EPLS inhibitor.
$\mathbf{T}$	EPLS sparse target.

Applications

$\mathrm{N_f}$	Number of features for methods that do not involve deep learning models.
O	Spatial prior.
GT	Ground truth.

FitNets

T	Teacher network.
S	Student network.
$\mathbf{a_T}$	Teacher softmax pre-activation.
$\mathbf{a_S}$	Student softmax pre-activation.
$\mathrm{P_T}$	Teacher softmax activation.
$\mathrm{P_S}$	Student softmax activation.
τ	Relaxation term.
$\mathrm{P_T^\tau}$	Teacher softmax activation.
$\mathrm{P_S^\tau}$	Student softmax activation.
$\mathbf{W_T}$	Weights of all layers of the teacher network.
$\mathbf{W_S}$	Weights of all layers of the student network.
$\mathbf{W_{Hint}}$	Weights up to the teacher's hint layer h.
$\mathbf{W_{Guided}}$	Weights up to the student's guided layer g.
$\mathbf{W_r}$	Weights of regressor.
u_h	Teacher deep nested function up to its hint layer.
v_g	Student deep nested function up to its guided layer.
r	Regressor function.

1

Introduction

1. INTRODUCTION

Building machines that see is a long lasting dream of many researchers. But what does it mean to see? According to David Marr [1], "... vision is the process of discovering from images what is present in the world, and where it is". Computer vision a discipline that aims to enable machines to see and interpret the visual world. But how can machines see? Computer vision is often approached from a learning perspective, where a large amount of visual information is presented to a machine in order to acquire knowledge and improve its own understanding of the visual world. The study of computer algorithms that can learn from data is referred to as machine learning. In the case of vision, machine learning algorithms use visual sensory data to acquire knowledge and improve their own understanding of the visual world. This learning is useful to accomplish many computer vision tasks, such as image classification, object recognition, object detection and scene understanding among other.

Data used for learning is composed of input observations, sometimes followed by a desired output value, called *label*. When desired output values are provided, the learning is said to be *supervised* and is directed by the desired output values that algorithms attempt to predict. When desired output values are not available, the learning is said to be *unsupervised* and aims to discover hidden structure in the input data.

No matter the kind of learning, capturing visual information, such as what is present in the world and where, directly from the raw input data is not trivial and often involves extremely complex and non-linear functions [2] that are hard to discover. Many machine learning problems become easier if the input data is transformed into a representation that emphasizes its most relevant characteristics.

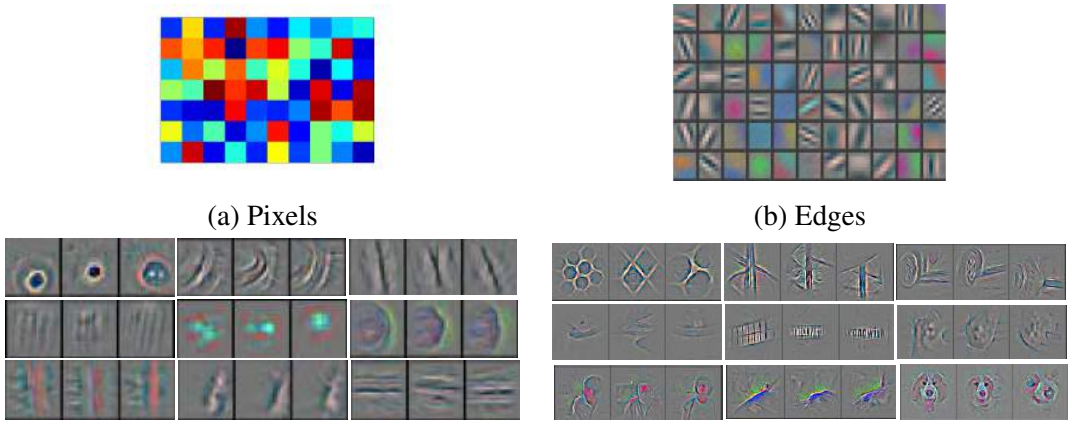
Data representation is a crucial step to many machine learning algorithms. Representations provide meaningful features, which constitute explicit pieces of relevant information to describe the observed data. There is no unique way to describe the data; outlining certain information comes at the expense of ignoring other information that may be hard to recover [1]. A good example to illustrate is presented by David Marr in [1] based on the Arabic and Roman numeral systems: The Arabic system represents an integer as a sum of multiples of powers of 10, whereas the Roman system represents an integer as a combination of letters. However, the usefulness of a representation relies on the task that we want to perform. Following the previous example, operations such as addition, subtraction or multiplication are easier to perform when using the Arabic representation of numbers.

In this thesis, we will focus on representations of visual data for computer vision tasks. There have been many attempts in the literature to handcraft representations, such as [3, 4].

Nevertheless, handcrafted representations require a fair amount of domain knowledge and it is not trivial how to evaluate whether such designed features are useful for a given task. Representation learning is a field of machine learning, which explores and learns relevant representations, also called *features*, from the data. In the recent years, learned features have proven to compete well or even outperform manually designed ones in many vision tasks such as image classification and object recognition [5, 6, 7]. The goal of representation learning is to seek and disentangle high level explanatory factors of variations in the observed data [8]. Factors of variations are high level concepts that influence the way we perceive the data. For example, if we consider an image of a dog, the factors of variations may include the position of the dog, its morphology, its color, the viewing angle, and the illumination, among others. Representations should capture the desirable factors of variations and discard the irrelevant ones. Therefore, discovering useful representations is a non-trivial task, which requires a high level of abstraction and understanding of the data.

Compositionality is a fundamental aspect of cognition [9, 10, 11]; we build complex concepts out of simpler ones, e.g. the concept "young man" is composed of the simpler concepts "young" and "man". Likewise, we can apply composition rules to visual data: pixels can be combined to form edges, which in turn can be combined to form more complex contours or objects. Hence, discovered representations should be able capture the hierarchical (compositional) nature of the visual world. Figure 1.1 shows a *representation* hierarchy, also called *feature hierarchy*, with increasing level of abstraction, following a natural progression from lower level (pixels) to higher level structures (object contours).

Deep learning is a branch of representation learning based on the idea of compositionality, which describes the world as a hierarchy of concepts, where more complex concepts are defined in terms of simpler ones [12]. Deep learning assumes distributed representations of the observed information and models high level concepts with increasing level of abstraction. The *depth* in deep learning models refers to the number of levels of abstraction we want to learn. The deeper we go in a representation hierarchy, the more abstract representations we expect to find. Learning such a representation hierarchy is a challenging task [13, 14]. Deep learning was popularized in 2006 with the introduction of successful approaches to learn deep feature hierarchies [15, 16] and has been receiving increasing attention ever since. In 2012, deep learning became a major breakthrough in the computer vision community by outperforming, by a large margin, classical computer vision methods on ILSVRC challenge [17]. Since then, deep learning models have been successfully applied to a large variety of computer vision tasks.



(a) Pixels (b) Edges

(c) Object contours (d) More complex object contours

Figure 1.1: Visual data representation hierarchy: (a) pixels are combined to form (b) edges, which are combined to form object contours (c), which in turn are combined to form complex object contours (d). Visualizations (b)-(d) excerpted from [20].

More recently, the trend has been to go *deeper* with representation hierarchies. A fair amount of work has been devoted to design *very* deep models that excel at many vision tasks. The top performers of ILSVRC 2014 challenge [18, 19] built deep models with 19 and 22 levels of abstraction respectively, highlighting the importance of depth. Along with the same trend, research has also been devoted to proposing new approaches to tackle the challenging problem of training *very* deep models.

In this thesis, we will study algorithms that attempt to learn deep representation hierarchies from visual data. We will start from the early ideas of [15, 16] to train deep models and proceed with a few steps in the same direction to tackle the problem of training deep models. Despite the existing literature on unsupervised learning, current state-of-the-art algorithms can often be unwieldy and, in many cases, are computationally intensive. We will attempt to address the just-mentioned concerns with the introduction of a novel algorithm to learn discriminative hierarchical features from the data. The main advantages of the proposed algorithm rely on extracting potentially relevant characteristics of the input data and simplifying the training process for practitioners, while being computationally efficient. We will exploit and extend the properties of the algorithm to learn deep representations on a variety of image types and problems, showing its ability to learn discriminative hierarchical features in different domains.

Yet, while emphasizing the usefulness of unsupervised methods when labeled data is scarce, the recent industrial success of deep learning has revolved around supervised learning. Com-

puter vision state-of-the-art top performing systems usually involve very large models, which are not well suited for applications with time or memory limitations. Therefore, more in line with the current trends, we will engage in making top performing models more efficient, by designing very deep models with a limited number of features per representation level. Since training the lower representation levels of very deep models still appears to be a challenging task, we will contribute with a novel algorithm that guides the training of very deep models by hinting their intermediate representations.

This thesis will proceed as follows:

Chapter 2: Background. In this chapter, we will cover the necessary background material to understand the contributions of this thesis. To that end, we will introduce the notation and terminology that we will use throughout the thesis.

Chapter 3: Meta-parameter free unsupervised sparse feature learning. In this chapter, we will present one of the main contributions of this thesis. The great majority of state-of-the-art algorithms involve tuning a few meta-parameters in order to train deep architectures and/or are computationally expensive. Therefore, we will introduce a meta-parameter free, off-the-shelf, simple and fast unsupervised learning algorithm, called Enforcing Population and Lifetime Sparsity (EPLS), which provides discriminative features that generalize well. Results will highlight the potential of the method when compared to other existing approaches.

Chapter 4: Training deep architectures by means of EPLS. In this chapter, we will build on top of the previous idea and will exploit the algorithm's properties to train deep learning models. To that end, we will choose two applications that could benefit from our approach. We will first consider the application of the algorithm to remote sensing data, which is characterized by its scarcity of labeled data. We will use this data to provide an in-depth analysis on the benefits of our method to extract hierarchical representations as well as the influence of the model's design on its performance. Second, we will tackle an image parsing problem and will extend our method to make it proficient at this task. Results will again highlight the potential of the method compared to other standard approaches.

Chapter 5: FitNets: Hints for Thin Deep Nets. In this chapter, we will present another important contribution of this thesis, which addresses the model compression problem, to mitigate the time and memory requirements of state-of-the-art models. We will design very deep models by limiting their number of features per representation level. In order to ease the training of such models, we will use hints from a top performing state-of-the-art model to assist the training of our compressed model. Results will show that compressed very deep models are able to extract feature representations that are comparable or even better performing than the ones extracted by state-of-the-art models, while compellingly reducing the time and memory consumption of the model.

Chapter 6: Conclusions. In this chapter, we will draw conclusions on the work presented in this thesis. To that end, we will emphasize the impact of the presented contributions, we will present possible future lines of research and will discuss some of the many problems that still remain unaddressed.

Much of the work presented in the above-described chapters has appeared in other publications. The results of Chapter 3 first appeared in [21]. Chapter 4 includes remote sensing results that came out in [22] and that are accepted for publication in [23]; and image parsing results that appeared in [24]. Finally, Chapter 5 was presented in [25].

2

Background

In this chapter, we will introduce the necessary notation and most relevant concepts that we will use throughout this thesis. We will start by describing how deep learning models are built in Section 2.1. Then, in Section 2.2, we will review the classical methods to train such models, discuss the difficulties encountered in this process and detail several state-of-the-art methods that attempt to overcome these difficulties, paying special attention to the ones that will show up later in the thesis. We will also outline how to use deep learning models to extract feature hierarchies in Section 2.3 and, finally, we will comment on commonly used pre-processing methods to prepare the data for training in Section 2.4.

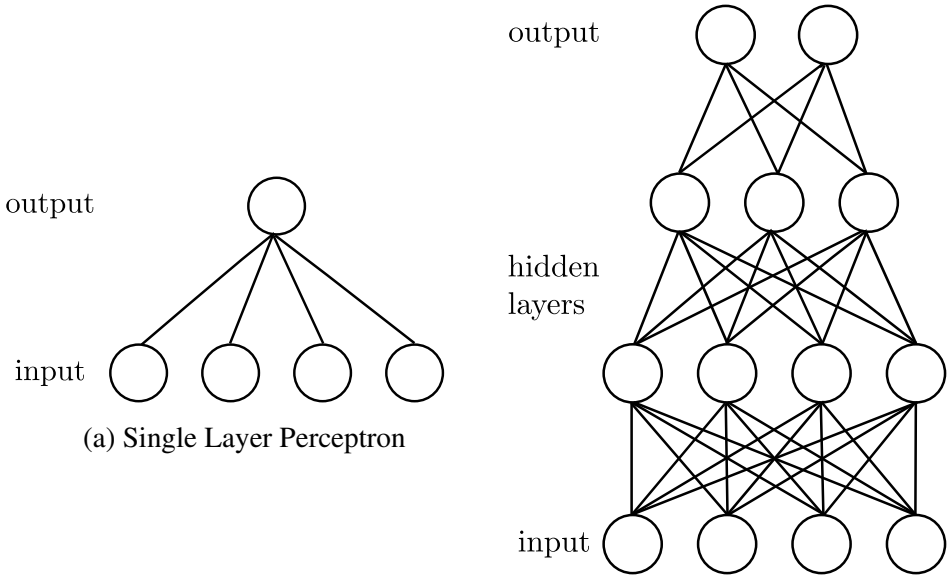
2.1 Deep Learning Models

As introduced in Chapter 1, deep learning seeks to describe the world as a hierarchy of abstractions. Deep learning refers to a type of computational models based on artificial neural networks. These networks are learning systems composed of interconnected processing units, called *neurons*, which communicate with each other. These units are organized in *layers*. Each layer corresponds to one level of abstraction in the feature hierarchy. Deep learning models are built by stacking together multiple layers. The design of a network, i.e. number of layers, number of units or connections among units, defines its *architecture*. The number of layers defines the *depth* of the architecture.

In this thesis, we will focus on a specific type of architectures, where there are no intra-layer connections between units and the information moves from the input units to the output units. Moreover, we will refer to architectures with one representation layer as *shallow* or *single layer* architectures, whereas we will refer to architecture with at least two representation layers as *deep* architectures.

2.1.1 Single layer Perceptron

A Single Layer Perceptron (SLP) [26] is the simplest kind of neural network trained as a binary classifier. It consists of a set of input units connected to an output unit by means of a set of learnable *weights*. The unit has an additional type of weight called *bias*, which allows to shift its output. Biases and weights form the *parameters* of a neural network. Figure 2.1(a) illustrates the SLP architecture.



(a) Single Layer Perceptron

(b) Multi-Layer Perceptron

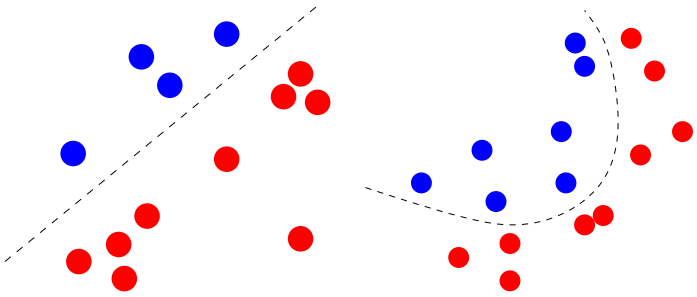
Figure 2.1: Difference between (a) Single Layer Perceptron and (b) Multi-Layer Perceptron architectures. For simplicity, the biases have been omitted without loss of generality.

The output of the network is computed as *learned affine transformation* followed by a threshold function *non-linearity*:

$$\mathbf{h} = \begin{cases} 1 & \text{if } \mathbf{xw} + b \geq 0 \\ 0 & \text{otherwise,} \end{cases} \tag{2.1}$$

where $\mathbf{x}$ is the input vector, $\mathbf{w}$ is the learnable weight vector, $\mathbf{xw}$ denotes the dot product, and b is a learnable bias. The threshold function determines the classifier output, i.e. whether or not an input belongs to a class.

Note that SLP can generalize to multi-class problems by changing its output unit for an output vector with as many output units as pre-defined classes. In this case, all input units are connected to all output units and each output unit has its own bias term. Given its nature, SLP can only learn *linear* classifiers and, therefore, cannot classify non-linearly separable input vectors properly. Figure 2.2 shows an example of linearly separable samples versus non-linearly separable samples.



(a) Linearly separable (b) Non-linearly separable

Figure 2.2: Difference between (a) linearly separable and (b) non-linearly separable samples. Image courtesy of Kyunghyun Cho.

2.1.2 Multi-layer Perceptron

A Multi-Layer Perceptrons (MLP) is a kind of neural network, which overcomes the SLP limitations by adding intermediate representation layers, called *hidden layers*, to build a feature hierarchy. MLP consists of an input layer, a number of hidden layers and an output layer. All these layers are stacked together such that the output of a layer is the input to the next layer. As in SLP, each unit within a layer is connected to each unit in the adjacent layers, i.e. layers are *fully-connected*. Connections between units are called *weights*. The weights connected to the same unit form a *filter*, also called *kernel*. MLPs' hidden and output units also have a *bias* associated to them. Figure 2.1(b) illustrates the architecture of an MLP with an input layer, two hidden layers and an output layer. The number of hidden layers and the number of units per hidden layer are *hyper-parameters* of MLP architectures.

Each layer of a MLP applies a non-linear transformation to the input data. Since MLP stack multiple layers together, its output can be computed as a function composition as

$$g^L(\cdots g^2(g^1(\mathbf{x};\theta^1);\theta^2)\cdots;\theta^L), \tag{2.2}$$

where $\mathbf{x}$ is an input vector, $\theta^l = \{\mathbf{W}^l, \mathbf{b}^l\}$ are layer l's parameters, g^l is layer l's non-linearity, $l \in [1 \cdots L]$ and L is the number of layers. The idea is to apply successive non-linear transformations to the input data in order to project it into a space, where it becomes (ideally) linearly separable. Figure 2.3 shows an example of non-linearly separable input data. The data is projected into a new space by each MLP layer until it becomes linearly separable.

MLPs' layers non-linear transformations consists in (1) computing a learned *affine transformation* of an input vector; and (2) applying an *element-wise non-linearity*. Section 2.1.2.1

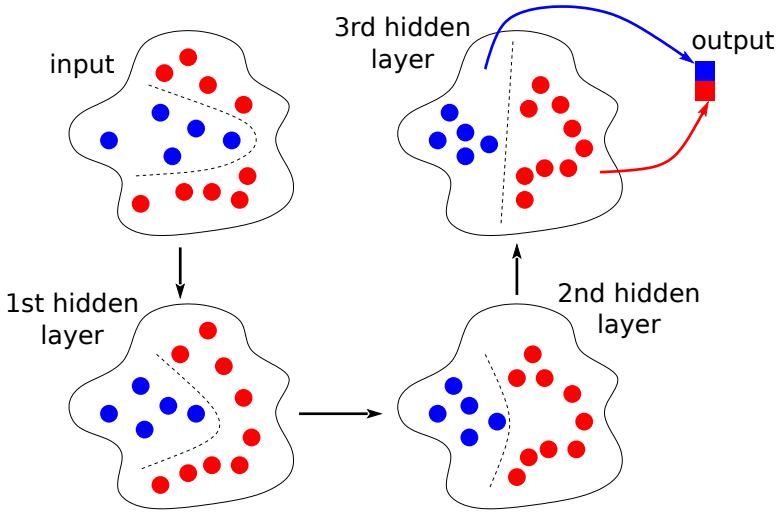


Figure 2.3: MLP non-linear projection of the data. Image courtesy of Kyunghyun Cho.

details how the affine transformation is performed and Section 2.1.2.2 enumerates the most commonly used non-linearities.

2.1.2.1 Affine Transformation

The learned *affine transformation* performed by each layer l of an MLP is computed as a matrix multiplication followed by a bias addition as

$$\mathbf{a}^l = \mathbf{h}^{l-1}\mathbf{W}^l + \mathbf{b}^l, \tag{2.3}$$

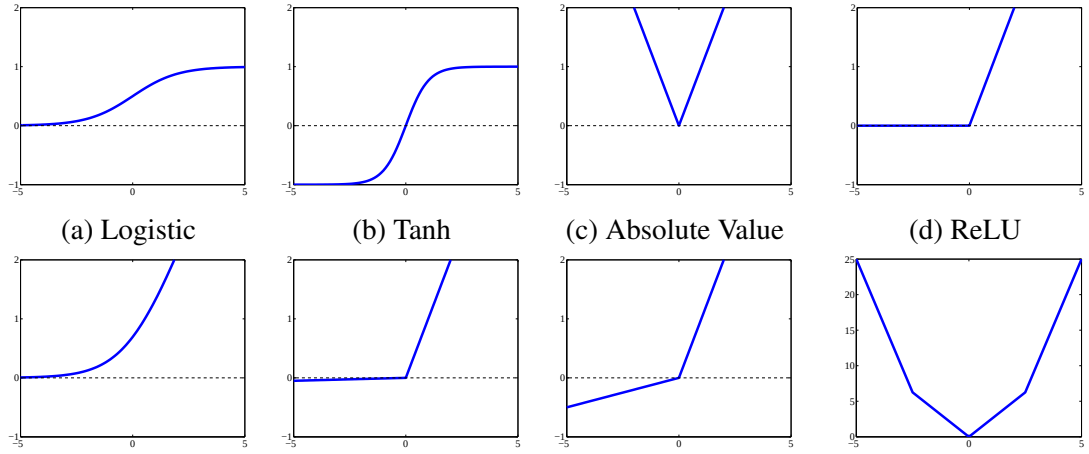
where $\mathbf{h}^{l-1} \in \mathbb{R}^{\mathrm{N}_\mathrm{h}^{l-1}}$ is the input of layer l, $\mathbf{W}^l \in \mathbb{R}^{\mathrm{N}_\mathrm{h}^{l-1} \times \mathrm{N}_\mathrm{h}^l}$ is the learnable weight matrix connecting input and output units within the layer, $\mathbf{b}^l \in \mathbb{R}^{\mathrm{N}_\mathrm{h}^l}$ is the layer's learnable bias vector and $\mathbf{a}^l \in \mathbb{R}^{\mathrm{N}_\mathrm{h}^l}$ is the output of the affine transformation, also called *pre-activation*. Note that $\mathrm{N}_\mathrm{h}^{l-1}$ and N_h^l are the input and the pre-activation dimensionality, respectively.

2.1.2.2 Non-linearity

An *element-wise* non-linearity f, also called *activation function*, is applied to the output of each layer's affine transformation $\mathbf{a}^l$ as

$$\mathbf{h}^l = f(\mathbf{a}^l), \tag{2.4}$$

where $\mathbf{h}^l \in \mathbb{R}^{\mathrm{N}_\mathrm{h}^l}$ is the output of layer l, also called *activation*. Note that both pre-activation and activation usually have the same dimensionality.



(a) Logistic (b) Tanh (c) Absolute Value (d) ReLU

(e) Softplus (f) LReLU (g) PReLU (h) Maxout approx. of quadratic function

Figure 2.4: Non-linearities typically applied after the affine transformation of a layer: (a) Logistic; (b) Tanh; (c) Absolute value; (d) ReLU; (e) Softplus; (f) Leaky ReLU; (g) Example of learned Parametric ReLU; (h) Example of how maxout can approximate a convex function.

Non-linearities are typically used to restrict the range of the output of the networks' layers. Many non-linearities have been successfully used in the literature; the most common ones are depicted in Figure 2.4 and listed below.

Logistic Squashing functions are among the most commonly used activation functions, since they allow to bound the output of units to a given range. The logistic function, shown in Figure 2.4(a), is a differentiable monotonically increasing function that asymptotes 0 as it approaches $-\infty$ and 1 as it approaches $+\infty$. It is defined as

$$\mathbf{h}^l = \sigma(\mathbf{a}^l) = \frac{1}{1 + e^{-(\mathbf{a}^l)}}. \tag{2.5}$$

This activation function can be used to model probabilities. However, its gradient vanishes as we increase/decrease $\mathbf{a}^l$.

Hyperbolic Tangent (tanh), depicted in Figure 2.4(b), is another commonly used squashing function, which is differentiable and monotonically increasing. It asymptotes -1 and 1 as it approaches $-\infty$ and $+\infty$, respectively. It is defined as

$$\mathbf{h}^l = \tanh(\mathbf{a}^l) = \frac{1 - e^{-2(\mathbf{a}^l)}}{1 + e^{-2(\mathbf{a}^l)}}. \tag{2.6}$$

Although tanh sigmoid has the same vanishing gradient problem as logistic sigmoid, it is preferred due to its symmetry w.r.t. the axes' origin, which makes the learning process converge faster (the average of its outputs is close to 0) [27]. In practice, a scaled version of the tanh non-linearity, $1.7159 \tanh(\frac{2}{3}\,\mathbf{a}^l)$ [27], is often used to further help convergence, by seeking the variance of the inputs transformed by the tanh to be close to 1.

Absolute Value (see Figure 2.4(c)) is a non-differentiable activation function, which has been used for object recognition tasks [28]. It is defined as

$$\mathbf{h}^l = \mathrm{abs}(\mathbf{a}^l) = |\mathbf{a}^l|. \tag{2.7}$$

As stated in [12], such non-linearity is useful for tasks "[...] where it makes sense to seek features that are invariant under a polarity reversal of the input illumination".

Rectifiers Rectifier Linear Unit (ReLU) non-linearity has been highly successfully applied to achieve state-of-the-art in many computer vision tasks and has proven to accelerate learning convergence [17, 29, 30, 31]. ReLU activation function is defined as

$$\mathbf{h}^l = \max(0, \mathbf{a}^l), \tag{2.8}$$

thus, the function has range $[0, +\infty)$. Such non-linearity encourages sparse representations by setting to 0 the negative values of $\mathbf{a}^l$. However, the function is non-differentiable and has zero gradient in the negative part of the rectifier (see Figure 2.4(d)).

Soft versions of ReLU have been proposed in the literature to deal with the zero gradient and/or its differentiability. Softplus [32], shown in Figure 2.4(e), is a smooth approximation of ReLU defined as

$$\mathbf{h}^l = \log(1 + e^{a^l}). \tag{2.9}$$

Leaky ReLU (LReLU) [33], shown in Figure 2.4(f), was introduced to allow a small, non-zero gradient when the unit is saturated and not active. LReLU is defined as

$$\mathbf{h}^l = \begin{cases} \mathbf{a}^l & \text{if } x > 0 \\ 0.01\mathbf{a}^l & otherwise. \end{cases} \tag{2.10}$$

Such alternatives have a negligible impact on the performance when compared to standard ReLU.

More recently, Parametric ReLU (PReLU) [34] was introduced as a generalization of ReLU. It has the same motivation as LReLU, i.e., avoid zero gradients. However, PReLU

non-linearity adaptively learns the coefficient β of the negative part of the rectifier, instead of fixing it to a small value (see Figure 2.4(g) for an example). PReLu is defined as

$$\mathbf{h}^l = \begin{cases} \mathbf{a}^l & \text{if } x > 0 \\ \beta\, \mathbf{a}^l & otherwise. \end{cases} \tag{2.11}$$

Maxout [35] is a cross-feature pooling activation function defined as

$$\mathbf{h}^l_k = \max_{j \in \mathcal{F}} \mathbf{a}^l_j \tag{2.12}$$

where $\mathbf{h}^l_k$ is a maxout unit and $\mathcal{F}$ is a set of consecutive, non-overlapping, pre-activation indices j. Maxout learns a convex activation function of each hidden unit by approximating it with a $|\mathcal{F}|$-piece piece-wise linear function, where $|\mathcal{F}|$ is the number of elements in $\mathcal{F}$. Note that maxout pre-activation and activation have different dimensionality. Figure 2.4(h) shows how maxout can approximate the quadratic function with a 5-piece piece-wise linear function.

Softmax is an activation function, which maps a vector to a categorical probability distribution and, which is typically used as output non-linearity. Each softmax output represents the probability of a class j and is computed as

$$\mathbf{h}^l_j = \mathrm{softmax}(\mathbf{a}^l) = \frac{e^{\mathbf{a}_j}}{\sum_k e^{\mathbf{a}_k}} \tag{2.13}$$

Note that softmax non-linearity is the generalization of the logistic non-linearity that allows to handle multiple classes.

2.1.3 Convolutional Neural Networks

As mentioned in Section 2.1.2, the full connectivity of an MLP requires each unit within a layer to be connected to each unit within the next layer. This design makes the number of learnable weights grow very rapidly with the input dimensionality, making it computationally unfeasible for large input dimensions. One way to deal with this problem is to remove full connectivity between adjacent layers.

A Convolutional Neural Network (CNN) [36, 37] is a type of network, which enforces a sparse connectivity among units of adjacent layers by restricting each hidden unit to be connected to only a small subset of contiguous input units. The region of contiguous units that defines the local connectivity of a hidden unit is called *receptive field*. Receptive fields can be

extracted at arbitrary input locations; we call *stride* the distance between consecutive receptive field centers. In addition, CNNs adopt a *weight sharing* scheme. By encouraging the re-use of weights, CNNs reduce significantly the number of parameters and memory requirements of the model. Note that, given their architecture, CNNs only capture local interactions and, thus, their design is valid when the input shares the same statistics at all locations, i.e. features learned at one location can be applied elsewhere.

A CNN is composed of a number of *convolutional* and *pooling* layers stacked together, optionally followed by fully-connected layers such as the ones described in Section 2.1.2. *Convolutional* layers consist of (1) a convolution of the input with a set of learnable *weights*, followed by a learnable *bias* addition, to extract local features; and (2) an *element-wise non-linearity* to allow deep architectures to learn non-linear representations of the input data. *Pooling* layers consist of a subsampling operation that aggregates the statistics of the features at nearby locations [12], to reduce the computational cost (by reducing the spatial size of the images), while providing a local translation invariance. The output of each layer is composed of *feature maps*. The output of a CNN is computed as a function composition

$$g^L(\cdots g^2(g^1(\mathbf{x}))), \tag{2.14}$$

where $\mathbf{x}$ is the input and g^l is a function that returns the output of either a convolutional, a pooling or, eventually, a fully-connected layer.

Figure 2.5 shows a classical CNN architecture with two convolutional layers and two subsampling (pooling) layers, followed by a fully-connected layer and an output layer. Section 2.1.3.1 to 2.1.3.3 detail the operations performed by convolutional and pooling layers. Note that CNN architectures have a significant number of *hyper-parameters*, such as the number of layers, the number of units per layer, the receptive field size, the stride and the spatial pooling size.

2.1.3.1 Convolution

A convolution is a mathematical operator, which applies a function repeatedly by translating it across the output of another function. By analogy, in the context of CNNs, the convolution operator replicates the weights of each layer over their entire input and computes the output of each feature at each input position.

Let $\mathbf{H}^{l-1} \in \mathbb{R}^{\mathrm{R}^{l-1} \times \mathrm{C}^{l-1} \times \mathrm{N}_\mathrm{h}^{l-1}}$ be the input to the l-th layer, which corresponds to the output of the previous layer and, which is composed of $\mathrm{N}_\mathrm{h}^{l-1}$ feature maps of size $\mathrm{R}^{l-1} \times \mathrm{C}^{l-1}$. Let

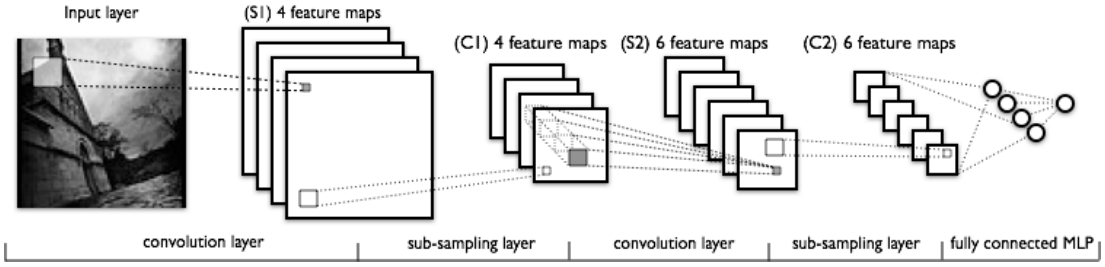


Figure 2.5: CNN architecture. Image from `http://deeplearning.net/tutorial/lenet.html`.

$\mathbf{b}^l \in \mathbb{R}^{\mathrm{N}_\mathrm{h}^l}$ be the learnable biases of the layer and $\mathbf{W}^l$ the learnable weights of the layer. As for MLP, the weights connected to the same output unit form a *filter*, also called *kernel*. These filters have a spatial size $\mathrm{w}_\mathrm{r}^l \times \mathrm{w}_\mathrm{c}^l$, which corresponds to the size of the receptive field of the units of the layer. Hence, $\mathbf{W}^l \in \mathbb{R}^{\mathrm{w}_\mathrm{r}^l \times \mathrm{w}_\mathrm{c}^l \times \mathrm{N}_\mathrm{h}^{l-1} \times \mathrm{N}_\mathrm{h}^l}$ is defined as a 4-D tensor

The result of applying a convolution to an input image (or feature map) is computed as

$$\mathbf{A}_j^l = \mathbf{H}^{l-1} \star \mathbf{W}_j^l + \mathbf{b}_j^l \; \forall j, \tag{2.15}$$

where the symbol $\star$ stands for the convolution operator and j is the index of the computed feature map. A convolution is performed for each filter $\mathbf{W}_j^l \in \mathbb{R}^{\mathrm{w}_\mathrm{r}^l \times \mathrm{w}_\mathrm{c}^l \times \mathrm{N}_\mathrm{h}^{l-1}}$. The output of the convolution $\mathbf{A}_j^l \in \mathbb{R}^{\mathrm{R}^l \times \mathrm{C}^l}$, called *pre-activation*, is a feature map indicating how well the filter matches the input at each location.

2.1.3.2 Non-linearity

As MLP affine transformations, CNNs' convolutions are followed by an element-wise non-linearity

$$\mathbf{H}_j^l = f(\mathbf{A}_j^l) \; \forall j, \tag{2.16}$$

where $\mathbf{H}_j^l$ is the output, also called *activation*, of a feature in the convolutional layer. Again, the feature's pre-activation and activation usually have the same dimensionality.

Non-linearities such as the ones summarized in Section 2.1.2.2 have been successfully used in conjunction with CNNs. Early works on CNNs used sigmoid non-linearities to build a feature hierarchy. In 2012, ReLU was shown to have significant benefits over squashing non-linearities and was successfully employed to learn the feature hierarchy that won ILSVRC2012 classification challenge [17]. Later on, maxout non-linearity was introduced to learn feature interactions and demonstrated to be able to extract powerful feature hierarchies for computer

vision tasks [35]. More recently, a CNN with PReLU non-linearity beat state-of-the-art results in computer vision applications [34]. Besides the above-mentioned non-linearities, Network In Network [38] was also used in conjunction with CNNs to learn the non-linearities to be applied.

Network In Network (NIN) [38] is a cascaded cross-feature pooling activation function, which adds an MLP within each layer of the network. Thus, the cross-feature pooling is performed as a weighted linear recombination of the output feature maps, followed by a ReLU non-linearity. Since MLPs are universal function approximators [39], using them as cross-feature pooling structures endows the model with more representation capability.

2.1.3.3 Pooling

Pooling layers, which are often stacked on top of convolutional layers, perform a subsampling operation, which reduces spatial dimensionality and achieves invariance to small translations by aggregating the statistics of the features at nearby locations. The purpose of the aggregation is to give invariance to the features by making them less sensitive to the exact location of the structures.

The result of applying a pooling operation to the output of a convolutional layer is computed as

$$\mathbf{H}_j^{l+1} = pool(\mathbf{H}_j^l)\ \forall j, \tag{2.17}$$

where $pool$ is the subsampling operation, which is applied to each feature map $\mathbf{H}_j^l$ of the previous convolutional layer. The resulting pooled feature maps $\mathbf{H}_j^{l+1} \in \mathbb{R}^{\mathrm{R}^{l+1} \times \mathrm{C}^{l+1}}$ are spatially smaller than their respective inputs $\mathbf{H}_j^l$. Note that in some particular cases, the number of features may also be reduced by the pooling operation. The most typically used pooling operations are listed below.

Average pooling is a conventional subsampling technique, which computes the average value of a feature j over a spatial region $\mathrm{R_p}$, i.e. $\frac{1}{|\mathrm{R_p}|} \sum_{i \in \mathrm{R_p}} \mathbf{H}_j^{l\,(i)}$, where $|\mathrm{R_p}|$ is the number of elements in the spatial region. In some cases, an equivalent form of pooling called **sum pooling** is used instead. Sum pooling computes the sum of a feature j over a spatial region $\mathrm{R_p}$, i.e. $\sum_{i \in \mathrm{R_p}} \mathbf{H}_j^{l\,(i)}$. In both pooling forms, we consider all elements of the region. This may lead to low pooled responses when ReLU is used as non-linearity, since strong responses are down-weighted by

the presence of zero outputs; or even worse, when tanh is used as non-linearity, since positive and negative responses might cancel each other out.

Max pooling is another conventional subsampling operation applied after the convolutional layer. It computes the maximum value of a feature j over a spatial region $\mathrm{R_p}$, i.e. $\max_{i \in \mathrm{R_p}} \mathbf{H}_j^{l\,(i)}$. Max pooling does not have the same problems as average/sum pooling, but is more prone to overfit the training set. However, this form of pooling is the most widely used in practice, combined with a plethora of regularization methods to avoid overfitting.

p-norm pooling is a natural generalization of the previous pooling forms. It computes the statistics of a feature j over a spatial region $\mathrm{R_p}$ under the p-norm, i.e. $(\sum_{i \in \mathrm{R_p}} |\mathbf{H}_j^{l\,(i)}|^p)^{1/p}$. For non-negative pooling inputs, if $p = 1$, the pooling is equivalent to average pooling, whereas if $p \to +\infty$ it is equivalent to max pooling.

Stochastic pooling [40] accounts for both strong and low outputs by applying a stochastic procedure to select a response within a pooling region. The procedure randomly picks an activation within the pooling region according to a multinomial distribution, given by the activations within the region. To do so, the probabilities p are computed for each location i within each region $\mathrm{R_p}$ as $p_i = \frac{\mathbf{H}_j^{l\,(i)}}{\sum_{k \in \mathrm{R_p}} \mathbf{H}_j^{l\,(k)}}$. After that, a location $s \in \mathrm{R_p}$ is picked by sampling $s \sim P(p_1, p_2, \cdots, p_{|R_p|})$ and the value $\mathbf{H}_j^{l\,(s)}$ of the selected location s is used as pooled response.

All Convolutional Net [41] analyze the possibility of replacing the conventional max-pooling operator by a convolutional layer with increased stride. Increasing the stride results in reducing the spatial resolution and, thus, acts as a subsampling technique. All Convolutional Nets achieve competitive results on several benchmark object recognition tasks. It is worth noticing that pooling operations usually have a feature-wise nature, i.e. they are applied to each feature independently. Therefore, substituting a pooling operation with a convolution operation adds inter-feature dependencies. This kind of cross-feature pooling is related to the maxout [35] and NIN [38] non-linearities (see Sections 2.1.2 and 2.1.3.1, respectively). In particular, all convolutional nets can be seen as a variant of NIN when only one 1×1 convolution is performed as substitute for spatial pooling.

2.2 Training Deep Learning Models

As introduced in Section 2.1, deep learning architures are built by stacking together computational modules, called *layers*. The output of a deep learning model is computed as a composition of L functions, where L is the number of layers in the architecture. Since each layer introduces a non-linearity, deep learning models result in highly non-linear functions. The parameters of these models are learned from the input data, such that the model's representation hierarchy is suitable for a given task. Finding a good parameter configuration with regard to some training criterion can be framed as an *optimization problem*.

In this section, we will describe how deep learning models are trained. In section 3.2.2, we will review standard optimization methods to train deep learning models and highlight the difficulties encountered while training these models. Then, in sections 2.2.2.1 to 2.2.4, we will present current state-of-the-art approaches to overcome the difficulties of training deep architectures.

2.2.1 Optimization

The *optimization problem* of training deep learning models consists in minimizing a function defined according to some training criterion. The function to be minimized is called *loss function*. The loss function (also called *objective* or *error* function) maps a set of inputs to a real number representing the error associated to them. The training process consists in tweaking the model parameters while minimizing the empirical average loss over the *training set*. An element of the training set is called *training sample*. The ultimate goal of training is to minimize the expected loss on unseen example, i.e. to generalize to new samples.

2.2.1.1 Loss Function

There are many loss functions used to train deep learning models.

In *supervised learning*, we have a training set

$$\mathcal{D} = \{(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}_{n=1}^{\mathrm{N}} \tag{2.18}$$

composed of N pairs of samples $\mathbf{x}^{(n)}$ and labels $\mathbf{y}^{(n)}$. Each pair $(\mathbf{x}^{(n)}, \mathbf{y}^{(n)})$ constitutes a *training sample*. The goal of supervised learning is to learn a function

$$\hat{\mathbf{y}} = g(\mathbf{x}), \tag{2.19}$$

which approximates as well as possible the mapping of an input sample $\mathbf{x}$, potentially not included in $\mathcal{D}$, to its corresponding label $\mathbf{y}$. The output $\hat{\mathbf{y}}$ of the learned function is called *prediction*. In order to learn the mapping g, we define a loss function $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$. One of the most popular choices for $\mathcal{L}$ is the Mean Squared Error (MSE), defined as follows

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{\mathrm{N}} \sum_{n \in [1...\mathrm{N}]} ||\hat{\mathbf{y}}^{(n)} - \mathbf{y}^{(n)}||^2. \tag{2.20}$$

However, when labels $\mathbf{y}$ are discrete and predictions $\hat{\mathbf{y}}$ contain probabilities, Average Cross-Entropy error (ACE) is the most widely used:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \mathcal{H}(\mathbf{y}, \hat{\mathbf{y}}) = -\frac{1}{\mathrm{N}} \sum_{n \in [1...\mathrm{N}]} \sum_{k \in [1...\mathrm{K}]} 1\{\mathbf{y}^{(n)} = k\} \log \hat{\mathbf{y}}_k^{(n)}, \tag{2.21}$$

where K is the number of classes and $1\{\cdot\}$ is the indicator function.

In *unsupervised learning*, we have a training set

$$\mathcal{D} = \{\mathbf{x}^{(n)}\}_{n=1}^{\mathrm{N}} \tag{2.22}$$

solely composed of N training samples; no labels are provided. Each $\mathbf{x}^{(n)}$ constitutes a *training sample*. The goal of unsupervised learning is to discover hidden structure in the unlabeled data. In this case, loss functions commonly follow sparsity or reconstruction constraints (see Section 2.2.2.1 for a detailed review of existing unsupervised criteria).

In both learning scenarios, it is important to have models that do well on unseen samples. *Regularization* is a technique that encourages the models to generalize beyond training samples. Regularizers often impose preferences, e.g. priors to the model parameters, such that some parameter configurations are more likely to be chosen than others. A common way to introduce regularization is by adding penalties to the loss function. One of the most frequent choices is the L_2 regularizer, also called *weight decay*, which penalizes large values of parameters. Large parameter values can potentially hurt generalization, since they may introduce excessive variance of the output. Weight decay is applied to the weights of each layer of a deep learning model and is defined as $\lambda||\mathbf{W}^l||_2^2$, where λ is a hyper-parameter controlling its strength. Following a similar spirit, L_1 regularizer has also been commonly used in the literature. As weight decay, L_1 regularizer also penalizes large weights. However, L_1 regularization results in numerous weights driven to zero.

Radically different regularization techniques, such as *dropout* [42], have been introduced in the literature in order to improve the way deep learning models learn and generalize. These

techniques do not rely on modifying the loss function. Dropout has become extremely popular and successful in recent years. It trains an ensemble of models that share their parameters. To do so, it randomly deactivates a set of hidden units in the network at each training iteration. This results in a set of weights and biases learnt under the conditions in which some hidden units are deactivated. Heuristically, it is like training many (sub-)models for one step using different subsets of data. Therefore, dropout effects are similar to the effects of averaging a very large number of different networks

2.2.1.2 Gradient-based Learning

Training deep learning models is an optimization problem that involves minimizing a loss function. Therefore, after defining the loss function, we need an optimization algorithm to find the model's parameters that minimize the empirical loss.

Gradient-based learning methods solve optimization problems by searching the direction of greatest increase rate of a loss function $\mathcal{L}$. The direction of greatest increase is given by the gradient of the function. Therefore, in order to minimize $\mathcal{L}$, one must move in the direction of the negative gradient (direction of greatest decrease). This procedure is known as *gradient descent*.

Gradient descent (GD) looks for a local minima of the loss function by iteratively taking steps along the negative direction of gradient and updating the parameters with the following rule

$$\theta' = \theta - \epsilon \nabla_\theta \mathcal{L}, \tag{2.23}$$

where $\theta = \{\mathbf{W}^l, \mathbf{b}^l\}_{l=1}^{L}$ are the model parameters and ϵ is the size of the step, called *learning rate*. Each GD update corresponds to one *iteration*. However, GD optimization is usually defined in terms of *epochs*. An *epoch* measures the number of times all samples in the training set have been presented to the model to update its parameters. The number of epochs in an optimization depends on the *stopping criterion*, also called *termination criterion*, which determines when to stop running the training algorithm.

The performance of GD optimization depends critically on the choice of the learning rate. There are different ways to determine the learning rate value: (1) set it to a small constant value and, optionally, apply an annealing schedule through time; (2) apply a *line search* strategy, i.e., try different values of ϵ and pick the one that minimizes the loss function the most; (3)

adopt adaptive learning rate schemes such as [43, 44]. Gradient descent converges when the parameter updates are barely noticeable, i.e., when all elements of the gradient are close to 0.

Gradient descent can be performed in a *batch* or in a *stochastic* fashion:

Batch Gradient Descent (BGD) computes the average gradient throughout the whole training set before updating the model parameters. Therefore, it has a smooth convergence, which reaches the local minimum lying in the basin of attraction of the parameter initialization [45].

Stochastic Gradient Descent (SGD) approximates the gradient with one single example of the training set. This results in a noisy gradient to update the model parameters, which is able to jump through different basins of attraction [45]. SGD is faster (it only computes the gradient of one sample per iteration) and often results in better solutions, since it can explore a broader range of possible solutions.

Mini-batch Stochastic Gradient Descent (mini-batch SGD) finds a compromise between BGD and SGD. It computes the average gradient of a subset of training samples in order to update the model parameters. Mini-batch SGD provides a slightly less noisy estimate of the gradient and, therefore, has a smoother convergence than SGD, while still being faster than BGD.

2.2.1.3 Back-propagation

Back-propagation [46] is an algorithm that efficiently computes the gradient of a loss function w.r.t. all the parameters (weights and biases) of a deep learning model. The algorithm is used in conjunction with gradient descent to find a good minimum of the loss function. Backpropagation takes advantage of the fact that deep learning models are function compositions (see Section 2.1.2) and applies the *chain rule* in order to compute the gradients w.r.t. the network's parameters. For simplicity, the bias terms will be omitted in the algorithm's description without loss of generality.

More precisely, the steps of the backpropagation algorithm can be summarized as

1. Perform a forward pass through the network, computing the activations of all the layers in the network. Figure 2.6(a) shows how the forward pass is performed and how the activation $\mathbf{h}_j^l$ of a unit j within each layer is computed.

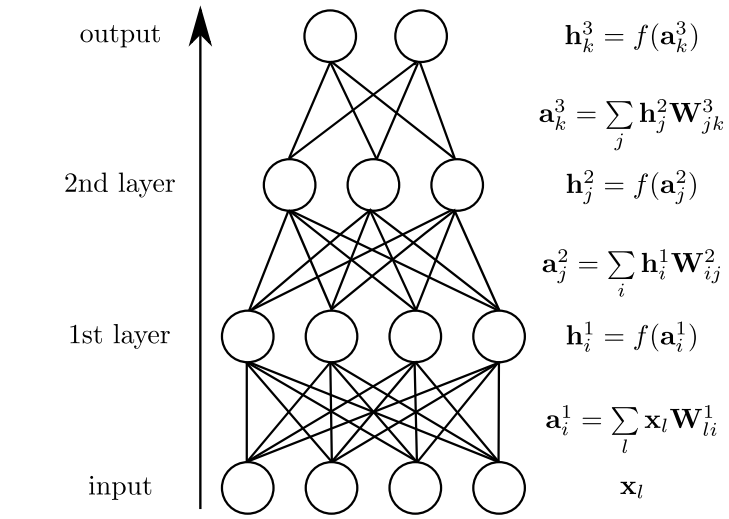
2. Perform a backward pass through the network, computing the partial derivatives of the loss function w.r.t. all parameters. Figure 2.6(b) shows how the backward pass is performed and how the partial derivatives w.r.t. all parameters are computed. First, the derivative of the loss function is computed w.r.t. each unit k of the output layer. Since the output units are a composition of two functions, the chain rule is applied as $\frac{\partial \mathcal{L}}{\partial \mathbf{a}_k^l} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_k^l} \frac{\partial \mathbf{h}_k^l}{\partial \mathbf{a}_k^l}$. Then, the derivatives of the loss function w.r.t. each hidden unit are computed. These derivatives involve a weighted sum of the derivatives of the loss function w.r.t. the pre-activation of the units in the layer above. As output units, hidden units compute a function composition. Therefore, the chain rule is applied analogously. Finally, the derivative of the loss function w.r.t. a weight $\mathbf{W}_{ij}^l$ is computed as $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{ij}^l} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_j^l} \frac{\partial \mathbf{a}_j^l}{\partial \mathbf{W}_{ij}^l} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_j^l} \mathbf{h}_i^{l-1}$.

Once the derivatives of the loss function w.r.t. each weight are obtained, the model parameters are updated by means of Eq. (2.23).

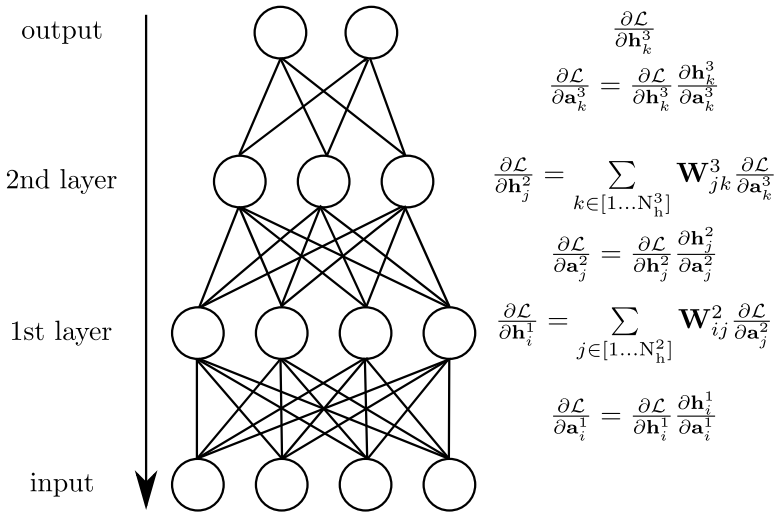
2.2.1.4 Optimization Difficulties

Training deep architectures has proven to be challenging [14, 47], since they are composed of successive non-linearities and, thus have highly non-convex and non-linear associated objective functions. These highly varying and non-linear objective functions present numerous non-optimal local minima as well as saddle points, which often make gradient-based optimization get stuck in poor solutions [48]. Moreover, random initialization of the parameters might lead to different solutions, which often do not provide equivalent solutions. Figure 2.7 shows how SGD follows different paths depending on the parameters initialization.

Moreover, gradient-based learning has the problem of vanishing/exploding gradients, when training deep architectures. Recall that deep architectures compute function compositions. Following the chain rule, the gradient of a function composition is computed as a product of terms of the derivatives. When the product terms are smaller than 1, the resulting gradient exponentially decreases. The tendency of the gradients to get smaller as we move to the bottom layers of the network is known as the *vanishing gradient* problem, which results in bottom layers learning more slowly than top layers. Analogously, there might be cases where the product terms are greater than 1. In this case, the resulting gradient blows up. The tendency of the gradients to get larger as we move to the bottom layers of the network is known as the



(a) Forward pass



(b) Backward pass

Figure 2.6: Backpropagation steps: (a) forward pass to compute activations, and (b) backward pass to propagate the gradients. For simplicity, bias terms have been omitted without loss of generality.

exploding gradient problem. In both cases, the network layers learn at different speeds and the optimization of the bottom layers gets hurt, leading to bad hidden representations of the data.

Significant effort has been devoted to alleviate the optimization problems of deep architectures. In the following sections, we will outline different approaches that have been introduced in the literature to assist the training of deep learning models.

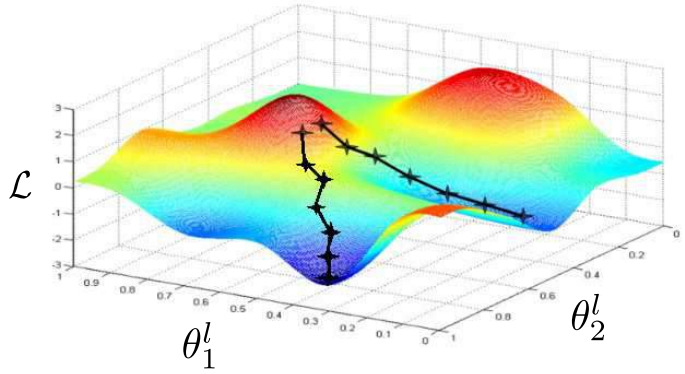


Figure 2.7: Given a loss function $\mathcal{L}$ with two parameters θ_1^l and θ_2^l to adjust, SGD finds different solutions depending on the parameter initialization. Image from `http://blog.datumbox.com/tuning-the-learning-rate-in-gradient-descent`.

2.2.2 Unsupervised Greedy Layer-wise (Pre-)training

Unsupervised pre-training strategies have been introduced in the literature to alleviate the optimization problems of training deep architectures. In [15, 16, 49, 50], unsupervised greedy layer-wise pre-training was presented to train deep architectures one layer at a time. Layers are trained in isolation by means of an unsupervised learning algorithm with the aim to discover hidden structure in the input data. These algorithms follow some criterion, e.g. sparsity[1] or reconstruction, to learn the layer's parameters. Layers are then stacked together to build a (pre-)trained deep architecture. The unsupervised pre-training strategy has the effect of leveraging the knowledge of the input data [51] and sets the network's parameters in a potentially good basin of attraction [48]. Note that in the case of CNNs, pooling layer may appear in between the convolutional layers. Whenever defined in the architecture, pooling operations are applied after pre-training the convolutional layer, and their output is used to pre-train the subsequent convolutional layer. Figure 2.8 shows the layer-wise pre-training of a 2-layer network. For simplicity, we have omitted the pooling layers, without loss of generality. The approach consists in training the first layer of the network from the input data. Then, use the output of the first layer as data to train the second layer of the network in isolation. Finally, the two layers are stacked together producing a (pre-)trained architecture.

[1] Sparsity is a desirable property of feature representations. Sparse features consist of a large amount of units, which respond rarely and provide high responses when they do respond. The concept of sparsity will be further discussed in Chapter 3.

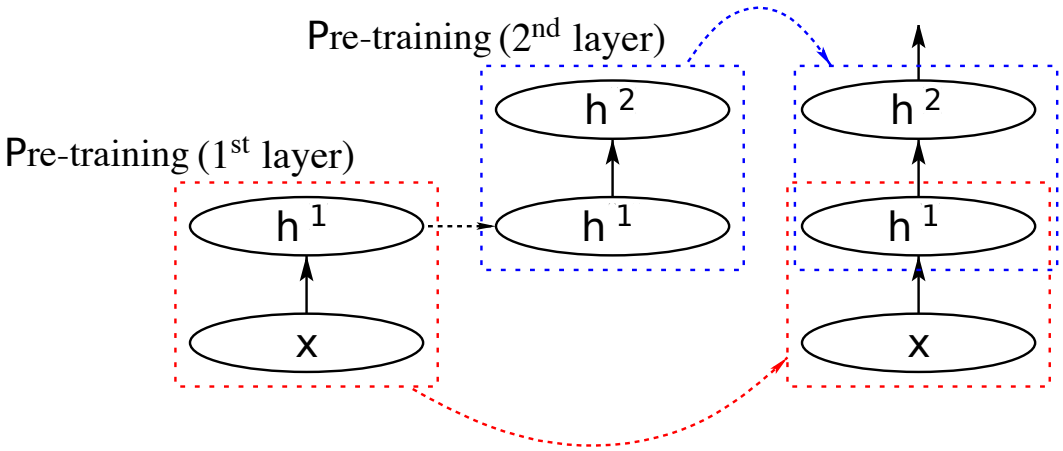


Figure 2.8: Greedy layer-wise pre-training of a deep architecture. For simplicity, we have omitted the pooling layers, without loss of generality. Image courtesy of Kyunghyun Cho.

In this section, we will review state-of-the-art unsupervised learning algorithms, which have been successfully applied to pre-train shallow and/or deep architectures in a greedy layer-wise fashion.

2.2.2.1 Unsupervised Learning Algorithms

Auto-encoders (AE) [13, 46, 52, 53] learn the mapping of the input data to a representation by means of an encoder of the following form

$$\mathbf{h}^l = f(\mathbf{h}^{l-1}{\mathbf{W_E}}^l + {\mathbf{b_E}}^l), \tag{2.24}$$

where ${\mathbf{W_E}}^l$ are the encoder weights and ${\mathbf{b_E}}^l$ its biases. The representation is then mapped back to the input space by means of a decoder of the following form

$$\tilde{\mathbf{h}}^{l-1} = f(\mathbf{h}^l{\mathbf{W_D}}^l + {\mathbf{b_D}}^l), \tag{2.25}$$

where ${\mathbf{W_D}}^l$ are the decoder weights and ${\mathbf{b_D}}^l$ its biases. In many cases, the decoder weight matrix is constrained to be the transpose of the encoder weight matrix, i.e. ${\mathbf{W_D}}^l = {{\mathbf{W_E}}^l}^{\mathrm{T}}$. This kind of auto-encoder is said to have *tied weights*. AE's parameters are optimized such that the reconstruction error is minimized. The most common way to measure the reconstruction error is the following

$$\mathcal{L}(\mathbf{h}^{l-1}, \tilde{\mathbf{h}}^{l-1}) = \sum_n ||{\mathbf{h}^{l-1}}^{(n)} - {\tilde{\mathbf{h}}^{l-1}}^{(n)}||_2^2. \tag{2.26}$$

Setting the reconstruction values $\tilde{\mathbf{h}}^{l-1}$ to be equal to the inputs $\mathbf{h}^{l-1}$ enforces the AE to learn an approximation of the identity function. When limiting the number of representation units, AE can discover interesting structure and properties of the input data, endowing them with compression capabilities. However, allowing the number of representation units to be *overcomplete* is also interesting. *Overcompleteness* implies the dimensionality of the representation to be greater than the dimensionality of the input.

Overcomplete AE are usually trained by enforcing sparsity constraints on the output representation. Sparse AE (SAE) [5] enforce representation units to have a small mean activation. To do so, a sparsity penalty is added to the classical auto-encoders' loss function:

$$\mathcal{L}(\mathbf{h}^{l-1}, \tilde{\mathbf{h}}^{l-1}) = \sum_{n} ||\mathbf{h}^{l-1^{(n)}} - \tilde{\mathbf{h}}^{l-1^{(n)}}||_2^2 + \lambda \sum_{j} KL(\rho||\rho_j), \tag{2.27}$$

where ρ defines target mean activation, ρ_j defines the mean activation of representation unit j, λ is a *hyper-parameter* that controls the weight of the sparsity penalty. KL denotes the Kullback-Leibler divergence between a Bernoulli random variable with mean ρ and a Bernoulli random variable with mean ρ_j. SAE seek to maintain similar activation statistics through all training samples among all representation units, thus ensure sparse representation of the data.

Other AE variants such as Denoising AE (DAE) [49] and Contractive AE (CAE) [54] have also been used in the literature to (pre-)train deep architectures [50, 54]. On one hand, DAE are trained to be robust in reconstructing the true input from a corrupted version of it. The algorithm encodes the corrupted input into a representation and, then, decodes the representation back to the input space, seeking to minimize the average reconstruction error between the output of the decoder and the true input. On the other hand, CAE add a penalty to the classical AE's reconstruction cost, which encourages local feature invariance (robust representations) and results in a space contraction. The penalty introduced by CAE is the squared Frobenius norm of the Jacobian of the activations w.r.t. the inputs.

Restricted Boltzmann Machines (RBM) [15, 55, 56] are energy based probabilistic models that learn non-linear representations of the input data. RBM are parameterized by weights connecting input (also called *visible*) and output (also called *hidden*) units and bias terms connected to each input/output unit. There is no interaction among units of the same layer. RBM's objective is to minimize the following energy function

$$E(\mathbf{h}^{l-1}, \mathbf{h}^{l}) = -\sum_{i \in visible} \mathbf{e}_i^l \mathbf{h}_i^{l-1} - \sum_{j \in hidden} \mathbf{b}_j^l \mathbf{h}_j^l - \sum_{i,j} \mathbf{h}_i^{l-1} \mathbf{h}_j^l \mathbf{W}_{ij}^l, \tag{2.28}$$

where $\mathbf{e}_i$ and $\mathbf{b}_j$ are the visible and hidden units' biases. In a RBM, units are random variables. The network assigns a probability to each state configuration; the joint probability of the model is defined as

$$P(\mathbf{h}^{l-1}, \mathbf{h}^l) = \frac{e^{-E(\mathbf{h}^{l-1}, \mathbf{h}^l)}}{Z}, \tag{2.29}$$

where Z is the partition function, i.e. sum over all possible $(\mathbf{h}^{l-1}, \mathbf{h}^l)$ pairs. Then, the marginal probability of $P(\mathbf{h}^{l-1})$

$$P(\mathbf{h}^{l-1}) = \sum_{\mathbf{h}^{l-1}} P(\mathbf{h}^{l-1}, \mathbf{h}^l), \tag{2.30}$$

In order to train a RBM, we aim to minimize the average negative log-likelihood (NLL)

$$\underset{\mathbf{W}^l, \mathbf{b}^l, \mathbf{c}^l}{\arg\min} - \sum_n \log P(\mathbf{h}^{l-1^{(n)}}). \tag{2.31}$$

Proceeding with gradient-based learning involves the computation of the expectation over the joint distribution (summing over $\mathbf{h}^{l-1}$ and $\mathbf{h}^l$), which is computationally intractable (see [13] for a detailed explanation). However, RBM's hidden units $\mathbf{h}^l_j$ are conditionally independent of each other when conditioning on $\mathbf{h}^{l-1}$. Likewise, visible units $\mathbf{h}^{l-1}_i$ are conditionally independent of each other when conditioning on $\mathbf{h}^l$. These properties ease the training of RBM, since the conditionals can be used as basis of Gibbs sampling to recover samples from the joint distribution [13]. Contrastive Divergence (CD-k) [57] approximates the expectation over the joint distribution with a point estimate with a single observation $\tilde{\mathbf{h}}^{l-1}$ by means of a k steps Gibbs sampling starting at a state configuration $\mathbf{h}^{l-1}$ corresponding to a true training sample, where k is chosen to be small. Nevertheless, starting each Gibbs chain with a true training sample limits the region of the energy function that we are going to explore. Persistent CD-k (PCD-k) [58] overcomes this limitation and allows to move further away from the training samples by starting each Gibbs sampling with the $\tilde{\mathbf{h}}^{l-1}$ of the previous training iteration. Training RBM with PCD-k obtains a better model of the probability distribution that generated the training samples, which provides better performance in terms of NLL on new examples.

RBM have also been trained with sparsity constraints [59] such as the one defined in SAE.

Sparse Coding (SC) [60, 61, 62] learns sparse over-complete representations of the input data as a linear combination of a set of filters $\sum_j \mathbf{h}^l_j \mathbf{W}^l_j$, where $\mathbf{h}^l_j$ are the coefficients applied

to each filter. In order to obtain sparse codes, SC sets most of the coefficients $\mathbf{h}_j^l$ to be close to 0. SC's optimization problem is defined as

$$\underset{\mathbf{h}^{l(n)},\mathbf{W}^l}{\arg\min} \sum_n ||\mathbf{h}^{l-1(n)} - \sum_j \mathbf{h}_j^{l\,(n)} \mathbf{W}_j^{l\,\mathrm{T}}||_2^2 + \lambda \sum_j S(\mathbf{h}_j^{l\,(n)}), \\ \text{subject to } ||\mathbf{W}_j^l||_2^2 = 1, \forall j, \tag{2.32}$$

where S is a sparsity penalty, e.g. L_1 penalty. Therefore, SC training optimizes over $\mathbf{h}_j^l$ for each sample and, then, optimizes over all $\mathbf{W}_j^l$. Note that, after training, SC still has to optimize the coefficients $\mathbf{h}_j^l$ to obtain the representation of each test example. Therefore, the method has a very computationally expensive inference.

Predictive Sparse Decomposition (PSD) [63] is a successful variant of sparse coding, which alleviates the expensive inference by learning a predictor to approximate a sparse representation. The method jointly learns the SC set of filters and an efficient predictor function. The learned predictor function will then be used as encoder. PSD's optimization problem is defined as

$$\underset{\mathbf{h}^{l(n)},\mathbf{W}^l,{\theta_\mathbf{E}}^l}{\arg\min} \sum_n ||\mathbf{h}^{l-1(n)} - \mathbf{h}^{l(n)} {\mathbf{W}^l}^{\mathrm{T}}||_2^2 + \lambda \sum_n ||\mathbf{h}^{l(n)}||_1 + ||\mathbf{h}^{l(n)} - f(\mathbf{h}^{l-1(n)}, {\theta_\mathbf{E}}^l)||_2^2 \tag{2.33}$$

where the first term of the loss function is the reconstruction penalty, the second term is the sparsity penalty and the third the encoder's penalty, i.e. it measures the discrepancy between the code predicted by the encoder and the sparse code. In this case, f represents the encoder function, with its respective parameters ${\theta_\mathbf{E}}^l$ and λ is a *hyper-parameter* controlling the weight given to the sparsity penalty.

Orthogonal Matching Pursuit (OMP-k) [64, 65, 66] learns the mapping of the input data to a sparse representation. The sparse representation is defined such that it contains at most k non-zeros, thus seeking sparsity. OMP-k is an iterative algorithm, which greedily picks a unit of the representation to be made non-zero in order to minimize the residual reconstruction

error. OMP-k's optimization problem is defined as

$$\begin{aligned} \underset{\mathbf{W}^l, \mathbf{h}^{l(n)}}{\arg\min} \sum_n ||\mathbf{h}^{l-1(n)} - \mathbf{h}^{l(n)} \mathbf{W}^{l\mathrm{T}}||_2^2, \\ \text{subject to } ||\mathbf{W}_j^l||_2^2 = 1, \forall j \\ \text{and } ||\mathbf{h}^{l(n)}||_0 \leq k, \forall i \end{aligned} \tag{2.34}$$

Therefore, it applies an alternating minimization to learn the weights $\mathbf{W}^l$: at each iteration, it selects a unit to be made non-zero and updates the representation accordingly to minimize the loss function. Units are selected based on $\arg\max_j |\mathbf{h}^{l-1(n)} \mathbf{W}_j^l|$.

Sparse Filtering [67] is an unsupervised feature learning method that optimizes for sparsity in the feature distribution. The method ensures (1) each sample to be represented by only a few active features and (2) features to have similar activity. To do so, the features are computed as

$$\mathbf{h}_j^l = f(\mathbf{h}_j^{l-1}; \theta^l), \tag{2.35}$$

and then normalized to be equally active across all samples

$$\tilde{\mathbf{h}}_j^l = \frac{\mathbf{h}_j^l}{||\mathbf{h}_j^l||_2}. \tag{2.36}$$

After that, features are normalized to lie in the ℓ_2 ball and optimized to be sparse as follows

$$\underset{\theta^l}{\arg\min} \sum_n \left|\left| \frac{\tilde{\mathbf{h}}^{l(n)}}{||\tilde{\mathbf{h}}^{l(n)}||_2} \right|\right|_1. \tag{2.37}$$

The output of each layer is computed as usual, by applying a non-linearity after the convolution operator or the affine transformation.

Independent Component Analysis (ICA) [68, 69] learns a set of weights $\mathbf{W}^l$ to map the input data to the representation space. ICA's most typical constraint is to enforce the filters to be orthonormal [69] to prevent degenerate solutions [70]. ICA's loss function is defined as [70]

$$\begin{aligned} \underset{\mathbf{W}^l}{\arg\min} \sum_n f(\mathbf{h}^{l-1(n)} \mathbf{W}^l), \\ \text{subject to } \mathbf{W}^l \mathbf{W}^{l\mathrm{T}} = \mathbf{I}, \end{aligned} \tag{2.38}$$

where $\mathbf{I}$ is the identity matrix. In order to provide sparse representation, the non-linearity f is usually chosen to be the L_1 norm penalty.

As mentioned in [70], ICA is sensitive to whitening (see Section 2.4 for a detailed explanation of whitening). In addition, its optimization problem involves an expensive orthonormalization at each iteration (after updating $\mathbf{W}^l$). Moreover, the orthonormality constraint limits the method to map input data into a representation space with lower or equal dimension, i.e. overcomplete representations cannot be learned. For all these reasons, ICA scales poorly to large and high-dimensional inputs and feature representations.

Reconstruction ICA (RICA) [70] was introduced to overcome the previously mentioned limitations. The algorithm substitutes the orthonormality constraint by a reconstruction penalty as non-degeneracy control measure. RICA's optimization problem is defined as [70]

$$\arg\min_{\mathbf{W}^l} \lambda \sum_n ||\mathbf{h}^{l-1(n)}\mathbf{W}^l\mathbf{W}^{l\mathrm{T}} - \mathbf{h}^{l-1(n)}||_2^2 + \sum_n \sum_j f(\mathbf{h}^{l-1(n)}\mathbf{W}_j^l), \tag{2.39}$$

where the first term corresponds to a reconstruction penalty, the second term corresponds to ICA's objective and λ balances both terms.

Other variants such as Topographic ICA (TICA) [69] and Tiled CNN [71] based on TICA have also been used in the literature to pre-train deep architectures.

2.2.2.2 Convolutional vs. Patch-based Training

The simplest way to learn convolutional layers' parameters by means of unsupervised learning algorithms is patch-based training [8]. Patch-based training consists in using a set of randomly extracted patches from input images (or feature maps) to train each layer. After that, the layer weights are applied to each input location to obtain output convolutional feature maps that will serve as input to the next layer. Pre-trained convolutional deep architectures are then built by stacking the unsupervised patch-based trained layers.

However, convolutional layers' parameters can also be learnt by means of unsupervised criteria in a convolutional fashion [8]. Convolutional training of convolutional layers has been shown to reduce the redundancy among learnt filters. Convolutional versions of the methods introduced in Section 2.2.2.1 have been proposed in the literature, such as Convolutional RBM [72, 73], Convolutional PSD [74], Convolutional SC [63, 75] or Convolutional AE [76].

2.2.3 Semi-supervised Learning

Semi-supervised strategies have been introduced in the literature to alleviate the optimization problems of training deep architectures. The goal of semi-supervised learning is to use unlabeled data to improve generalization of a supervised task. First, an unsupervised criterion is employed to transform the input data into a representation, such that points with the same label lie within the same structure [12, 77]. Then, the extracted representations are fed to a supervised criterion to learn the task at hand. Both unsupervised and supervised criteria can be optimized sequentially or jointly.

In the context of deep learning models, semi-supervised methods are either trained sequentially by performing an unsupervised greedy layer-wise pre-training (see Section 2.2.2.1), followed by a supervised *finetuning* [15, 16, 78] or jointly by training a trade-off of the unsupervised and supervised criteria simultaneously [77, 79]. Although the literature on this subject is much broader, in this section, we will describe only a subset of the most relevant methods to the full understanding of this thesis.

Unsupervised pre-training followed by supervised finetuning [15, 16] is a commonly used strategy to train deep networks in a semi-supervised fashion. The networks are trained in two sequential stages: (1) using some unsupervised criterion on unlabeled samples to pre-train the network (as described in Section 2.2.2.1) and (2) *finetune* the whole network according to a supervised criterion. The role of unsupervised pre-training is to find a parameter configuration that helps supervised learning yield better performance in terms of generalization. As argued in [51], unsupervised pre-training acts as a regularizer that minimizes the variance and introduces bias towards more useful and more effective parameter configurations [51]. The idea is to define a particular initialization of parameters, in a potentially better basin of attraction, for supervised finetuning to follow. Initializing the network's parameters with unsupervised pre-training has shown to improve generalization w.r.t. random initialization of the parameters [15, 16, 51]. The role of supervised finetuning is to adjust all layer's parameters together by means of backpropagation in order to approximate as well as possible the mapping of an input sample to its corresponding label. This strategy has been applied to a diversity of tasks with varying amounts of labeled data and has shown to be successful, especially when labeled data is scarce [15, 16, 78].

Semi-supervised embedding [77] is a semi-supervised strategy to improve the training of deep architectures. It consists in adding an unsupervised embedding criterion to any (or all) layers of the network and optimize for both the embedding and the standard supervised criteria simultaneously. The method reveals itself to be a compelling strategy to improve the training of deep architectures, highlighting its benefits when dealing with complex tasks.

Discriminative RBM [79] are a kind of hybrid RBM, in which a discriminative component is introduced to make the RBM a stand-alone non-linear classifier. In the semi-supervised setting, the deep architecture is jointly optimized to be a good generative model of unlabeled data as well as a good hybrid (generative-discriminative) model of the labeled data.

2.2.4 Supervised Guidance

Supervised algorithms have also been used in the literature to alleviate the optimization problem of deep architectures. In this section, we will briefly review a few state-of-the-art supervised learning techniques, which assist the training of deep architectures.

Supervised pre-training [16] is a method to pre-train deep architectures in a *supervised* greedy layer-wise fashion. Layers are trained one after another as the hidden layer of a supervised network. To do so, a supervised output layer is stacked on top of the layer to be trained. After optimization, the supervised output layer is thrown away and the trained parameters are used to extract the input representation of the subsequent layer. Using a supervised criterion to pre-train deep architectures in a greedy layer-wise fashion has proven to be helpful in assisting the training of deep architectures. However, as stated in [16], such pre-training might not be able to capture all relevant information about the target using a single hidden layer neural network at each stage and, therefore, might yield poorer solutions than unsupervised pre-training.

Knowledge matters [80] guides the training of supervised deep learning models by introducing prior information for optimization. It is motivated by the fact that some tasks appear to be extremely difficult for state-of-the-art methods. The idea is to provide context information on the task at hand to the intermediate layer of the deep architecture. The ultimate task is then divided into two presumably simpler subtasks during training. The first task consists in optimizing the first half of the deep architecture w.r.t. the context information. The output of

the first half of the network is then used as input to the second half, which is in turn optimized w.r.t. the supervised end-task.

Auxiliary supervision strategies [19] have been used in the literature to assist the training of supervised deep learning models. The idea consists in adding an auxiliary supervised layer to any intermediate layer of the network in order to make the output of the layer more discriminative w.r.t. the true labels and provide regularization. The whole architecture is then trained by jointly optimizing the general supervised loss of the network as well as the losses of the auxiliary supervised layers.

Deeply-supervised nets [81] propose to assist the training of deep networks by introducing a *companion objective* to each individual hidden layer, in addition to the overall network objective. The companion objective is defined as a source of supervision that encourages highly discriminative intermediate feature maps. The deep network is trained to jointly optimize its output objective as well as the companion objective of each hidden layer.

2.2.5 Curriculum Learning

Curriculum Learning strategies [82] have been introduced in the literature to alleviate the optimization problem of learning deep architectures. Curriculum learning is a radically different approach, which aims to ease the optimization problem by organizing the data/concepts presented to the learner in a meaningful way. To do so, the method re-weights the training distribution according to some criterion, such that the learner network gradually receives examples of increasing and appropriate difficulty w.r.t. the already learned concepts. This strategy provides guidance during the training process of deep architectures. As a result, curriculum learning can dramatically speed up and improve the quality of the learning process, achieving a faster convergence and finding potentially better local minima of highly non-convex loss functions. Therefore, curriculum learning leads to a potentially better model generalization by properly choosing the sequence of training samples/concepts seen by the learner: from simple samples/concepts to more complex ones.

It is worth mentioning that the design of the curriculum tends to be problem-specific. For example, in [82], the simplicity of an example in a sequence is measured by means of hand-designed heuristics. However, in the Knowledge Matters approach [80] presented in Section

2.2.4, the curriculum is built by decomposing the end-task into simpler subtasks and introducing the subtasks to the learning in increasing order of complexity. In this case, each subtask requires some prior knowledge on the end-task. Moreover, providing a meaningful ranking of the samples is not trivial. Attempts to alleviate this problem have been introduced in the literature. For instance, self-paced learning [83] defines a measure of "easiness" that adapts to the learner's abilities. This measure has been extended in [84] to account for diversity in the samples shown to the learner.

2.3 Feature Encoding

After training the parameters of a shallow/deep architecture (see Section 2.2) in an unsupervised fashion, we can proceed to use the model to extract feature representations. To do so, we must choose an *encoder* to map the input of the model to its representation. The encoder is defined as the *non-linearity* f_{enc} to be applied as follows

$$\mathbf{h}^l = f_{enc}(\mathbf{h}^{l-1}\mathbf{W}^l + \mathbf{b}^l). \tag{2.40}$$

As discussed in [66], the encoding choice can critically affect the performance of a system.

Moreover, the use of *polarity split* has shown to further improve the performance of many encoding strategies [66]. Polarity splitting takes into account the positive and negative components of a code in the following way:

$$\begin{aligned} \mathbf{h}^l_+ &= f_{enc}(\mathbf{h}^{l-1}\mathbf{W}^l + \mathbf{b}^l) \\ \mathbf{h}^l_- &= f_{enc}(\mathbf{h}^{l-1}(-\mathbf{W}^l) + \mathbf{b}^l) \\ \mathbf{h}^l &= [\mathbf{h}^l_-, \mathbf{h}^l_+], \end{aligned} \tag{2.41}$$

where $\mathbf{h}^l$ is the concatenation of $\mathbf{h}^l_-$ and $\mathbf{h}^l_+$. Note that this is usually applied to the last representation layer of the network. As a result, the number of features in the last representation layer is doubled. Note that, the same encoding procedure can be applied to convolutional architectures. In that case, the encoding might be followed by a pooling operation.

In the remaining part of this section, we will present the most commonly used encoding methods in the literature.

Natural encoding is the straightforward choice of encoding. Natural encoding corresponds to whichever non-linearity is associated to the learning algorithm.

Soft-threshold applies a fixed threshold non-linearity as

$$\mathbf{h}_j^l = \max\{0, \mathbf{h}_j^{l-1}\mathbf{W}_j^l - t\}, \tag{2.42}$$

where t is the threshold to be tuned. When applying polarity split, the encoding of each feature is computed as

$$\begin{aligned} \mathbf{h}_{j+}^l &= \max\{0, \mathbf{h}_j^{l-1}\mathbf{W}_j^l - t\} \\ \mathbf{h}_{j-}^l &= \max\{0, \mathbf{h}_j^{l-1}(-\mathbf{W})_j^l - t\}. \end{aligned} \tag{2.43}$$

Sparse coding computes the sparse code s associated with the input by optimizing Eq. 2.32 over $\mathbf{h}^l$, holding $\mathbf{W}^l$ fixed. When applying polarity split, the encoding of each feature is computed as in [66] as follows

$$\begin{aligned} \mathbf{h}_{j+}^l &= \max\{0, s\} \\ \mathbf{h}_{j-}^l &= \max\{0, -s\}. \end{aligned} \tag{2.44}$$

2.4 Data Pre-processing for Images

Data pre-processing is crucial to many deep learning methods. In this section, we will briefly describe the most commonly used pre-processing methods in practice when dealing with images.

2.4.1 Contrast and Brightness Normalization

This normalization is commonly applied to images. Each input image is normalized by subtracting its mean and dividing by its standard deviation

$$\mathbf{x_{norm}}_j^{(n)} = \frac{\mathbf{x}_j^{(n)} - \mu^{(n)}}{\sigma^{(n)}}, \tag{2.45}$$

where $\mu^{(n)}$ and $\sigma^{(n)}$ are the mean and the standard deviation accros the elements $\mathbf{x}_j^{(n)}$ of the sample $\mathbf{x}^{(n)}$, respectively.

2.4.2 Feature Standardization

This normalization is widely used in both machine learning and computer vision literature. It consists in normalizing each feature (data dimension) independently such that it has zero mean and unit variance

$$\mathbf{x_{std}}_j^{(n)} = \frac{\mathbf{x}_j^{(n)} - \mu_j}{\sigma_j}, \tag{2.46}$$

where μ_j and σ_j are the mean and the standard deviation of each feature j throughout the whole training set.

2.4.3 Whitening

Besides standard normalizations such as contrast normalization and/or feature standardization, whitening is often applied to help deep learning algorithms obtain better representations and speed up their training. Whitening consists in decorrelating a set of features, since raw input images exhibit a high redundance (adjacent pixels are highly correlated). To do so, it transforms an input set of features with covariance matrix $\mathbf{\Sigma}$ into a set of features with identity covariance matrix.

PCA Whitening allows us to decorrelate the features and optionally reduce the data dimensionality. To do so, we first center the data to ensure that it has zero mean. Then, we compute the covariance matrix $\mathbf{\Sigma}$ of the input data. After that, we compute the eigenvectors and eigenvalues of $\mathbf{\Sigma}$. The eigenvectors are stored one per column in a matrix $\mathbf{U}$ and the eigenvalues are stored in a diagonal matrix $\mathbf{V}$. Finally, the output of PCA whitening is computed as

$$\mathbf{x}_{\mathrm{PCAw}}^{(n)} = \frac{\mathbf{U}^{\mathrm{T}}\mathbf{x}^{(n)}}{\sqrt{\mathrm{diag}(\mathbf{V}) + \eta}} \tag{2.47}$$

As a result, the features are uncorrelated and have unit variance. Optionally, the dimensionality of the projected data can be reduced by keeping k components of $\mathbf{x}_{\mathrm{PCAw}}^{(n)}$.

ZCA Whitening computes a different transformation to get the set of features to have identity covariance matrix and does not perform any dimensionality reduction. More precisely, it computes its output as

$$\mathbf{x}_{\mathrm{ZCAw}}^{(n)} = \mathbf{U}\mathbf{x}_{\mathrm{PCAw}}^{(n)} = \mathbf{U}\frac{\mathbf{U}^{\mathrm{T}}\mathbf{x}^{(n)}}{\sqrt{\mathrm{diag}(\mathbf{V}) + \epsilon}} \tag{2.48}$$

3

Meta-parameter free unsupervised sparse feature learning

As discussed in Chapter 2, training deep architectures is known to be a challenging optimization problem, which has received much attention in the last decade. Unsupervised greedy layer-wise pre-training [15, 16] was introduced as a successful procedure to learn deep feature hierarchies. The method trains the layers of a deep architecture one after another by means of an unsupervised learning algorithm, which usually follows some sparsity and/or reconstruction criterion.

One of the main criticism to state-of-the-art unsupervised feature learning methods is that they require a significant amount of hyper-parameters. Note that, in this chapter, the term *hyper-parameter* refers to any parameter of an unsupervised learning algorithm. The tuning of these hyper-parameters is a laborious task that requires expert knowledge, rules of thumb or extensive search and, whose setting can vary for different tasks [85]. Therefore, there is great interest for hyper-parameter free methods [67] and automatic approaches to optimize the performance of learning algorithms [85]. Nevertheless, little effort has been devoted to address this problem (see Table 3.1 for a comparison of hyper-parameters required by various unsupervised learning algorithms). To the best of our knowledge, work in this direction includes ICA [68, 69] and sparse filtering [67].

Computational complexity is also a major drawback of many state-of-the-art methods. ICA [68, 69] requires an expensive orthogonalization to be computed at each iteration. Sparse coding [60, 61, 62] has an expensive inference, which requires a prohibitive iterative optimization. Significant amount of work has been done in order to overcome this limitation [63, 86]. Predictive Sparse Decomposition (PSD) [63] is a successful variant of sparse coding, which uses a predictor to approximate the sparse representation and solves the sparse coding computationally expensive encoding step.

In this chapter, we will present one of the main contributions of this thesis [21], namely, a hyper-parameter free, off-the-shelf, simple and fast unsupervised feature learning algorithm, which exploits a new way of optimizing for sparsity and provides discriminative features that generalize well. In Section 3.1, we will define the concept of *sparsity* and explain how it relates to various state-of-the-art unsupervised feature learning algorithms. Then, in Section 3.2 we will describe the method that we propose. After that, Section 3.3 will show the results achieved by the proposed method on benchmark datasets. Sections 3.4 and 3.5 will be devoted to highlighting the importance of sparsity and discussing the computational complexity, respectively. In Section 3.6, we will discuss the obtained results. Finally, we will summarize the contribution in Section 3.7.

Table 3.1: Hyper-parameters to tune of various state-of-the-art unsupervised feature learning methods.

Sparse RBM	target activation, sparsity penalty
Sparse Auto-encoder	target activation, sparsity penalty
Sparse Coding (SC)	sparsity penalty
RICA	reconstruction penalty
PSD	sparsity penalty, prediction penalty
OMP-k	k (non-zero elements)
ICA	-
Sparse Filtering	-

3.1 Sparsity

As mentioned in Chapter 2, many state-of-the-art unsupervised learning algorithms follow sparsity criteria. Sparsity is among the desirable properties of a good feature representation [5, 8, 59, 60, 67, 70, 87]. Sparse features consist of a large amount of units, which respond rarely and provide high responses when they do respond. Sparsity can be described in terms of *population* sparsity and *lifetime* sparsity [88]. Both lifetime and population sparsity are important properties of the feature distribution. On one hand, *lifetime* sparsity plays an important role in preventing bad solutions such as numerous *dead features*. A feature is said to be *dead* when it does not activate for any training data. There seems to be a consensus to overcome such degenerate solutions, which is to ensure similar statistics among features [5, 67, 87, 88]. On the other hand, *population* sparsity helps providing a simple interpretation of the input data such as the ones found in early visual areas. To the best of our knowledge, the proportion of features to be active per sample remains ambiguous.

A common way to enforce sparsity in the feature representation is by adding an L_1 penalty term in the objective function. The great majority of state-of-the-art methods described in Chapter 2 optimize either for one or both sparsity forms. On one hand, sparse auto-encoders and sparse RBM seek lifetime sparsity by optimizing for a target activation. This target activation requires tuning and does not explicitly control the level of population sparsity. On the other hand, OMP-k and sparse coding solely seek population sparsity. OMP-k defines the level of population sparsity by setting k to the maximum expected number of non-zero elements per feature representation, whereas sparse coding [60] ensures population sparsity by constraining the number of active features per sample to be small.

Although different in the sparsity form imposed to their features, the above-mentioned methods share the objective of explicitly modeling the data distribution by minimizing the reconstruction error. A completely different perspective is introduced by sparse filtering [67]. Rather than attempting to explicitly model the input distribution, the method focuses on the properties of the feature distribution, by optimizing for population sparsity while trying to maintain similar activation among features. As a result, lifetime sparsity is somehow achieved in the features.

Although learning a good approximation of the data distribution may be desirable, approaches such as sparse filtering [67] highlight that this seems not so important if the goal is to build a discriminative sparse system.

3.2 Method

In this section, we will describe how the proposed method learns a sparse feature representation of the data in terms of population and lifetime sparsity. The method iteratively builds a sparse target and optimizes the parameters of a system, namely a layer of a deep architecture, by minimizing the error between the system output and the sparse target. In section 3.2.1, we will highlight the algorithm to enforce lifetime and population sparsity in the sparse target. Then, in Section 3.2.2 we will provide implementation details on the system and optimization strategies used to minimize the error between the system output and the sparse target.

3.2.1 Enforcing Population and Lifetime Sparsity by Defining a Target

We define population and lifetime sparsity as properties of a sparse feature representation. Given N training samples and their corresponding feature representations of dimensionality N_h^l, we define the first property of a feature representation as:

1. **Strong Lifetime Sparsity:** Each feature representation must be composed solely of active and inactive units (no intermediate values between two fixed scalars are allowed) and all units must activate for an equal number of inputs. Activation is exactly distributed among the N_h^l features.

Our Strong Lifetime Sparsity definition is a more strict requirement than the *high dispersal* concept introduced in [67], which only requires that "the mean squared activations of each feature [...] should be roughly the same for all features". While high dispersal attempts to

diversify the learned filters, it does not guarantee the feature distribution, in the lifetime sense, to be composed of only a few activations. Furthermore, our definition ensures the absence of dead features.

Given our definition of Strong Lifetime Sparsity, we define population sparsity as:

2. **Strong Population Sparsity:** For each training sample only one feature must be active.

The rationale of our approach is to appropriately generate a sparse target that fulfills properties (1) and (2), and then learn the parameters of the system by minimizing the L_2 error between the sparse target and the feature representation generated by the system during training. In this way, *we seek a system optimized for both population and lifetime sparsity in an explicit way.*

The key component of our approach is how to define the sparse target based on the above-mentioned properties. However, to ensure that the optimization of the system parameters converges, we add a third property:

3. **Minimal Perturbation:** The sparse target should be defined as the best approximation of the system's generated feature representation, as measured by L_2 error, that fulfills properties (1) & (2).

Creating the sparse target that ensures the above-mentioned properties is analogous to solving an assignment problem. The Hungarian method [89] is a combinatorial optimization algorithm, which solves the assignment problem. However, its computational cost $\mathcal{O}((\mathrm{NN}_\mathrm{h}^l)^{3/2})$ is prohibitive. Therefore, in the next subsection we propose a simple and fast $\mathcal{O}(\mathrm{NN}_\mathrm{h}^l)$ algorithm to generate the sparse target, which ensures sparsity properties (1) & (2) and provides an approximate solution for minimal perturbation property (3).

3.2.1.1 Sparse Target Generation: the Enforcing Population and Lifetime Sparsity (EPLS) Algorithm

Let us assume that we have a system, which produces a row feature vector $\mathbf{h}^l \in \mathbb{R}^{\mathrm{N}_\mathrm{h}^l}$. We use the notation $\mathbf{h}_j^l$ to refer to one element of $\mathbf{h}^l$. We define a feature matrix $\mathbf{H}^l$ (e.g., built as the output of a network's layer l, see Section 3.2.2 for details), composed of N feature vectors. We define $\mathbf{H}^{l(b)}$ as a matrix containing a subset of N_b rows of $\mathbf{H}^l$, where $\mathrm{N}_\mathrm{b} \leq \mathrm{N}$. Likewise, we define a sparse target matrix $\mathbf{T}^{(b)}$ of the same size. Algorithm 1 details the EPLS algorithm

Algorithm 1 EPLS

Input: $\mathbf{H}^{l(b)}$, $\mathbf{c}$, N
Output: $\mathbf{T}^{(b)}$, $\mathbf{c}$
1: $\mathbf{T}^{(b)} = 0$
2: $\mathbf{H}^{l(b)} = \frac{\mathbf{H}^{l(b)} - \min(\mathbf{H}^{l(b)})}{\max(\mathbf{H}^{l(b)}) - \min(\mathbf{H}^{l(b)})}$
3: **for** $n = 1 \rightarrow \mathrm{N_b}$ **do**
4: $\quad \mathbf{h}^l_j = \mathbf{H}^{l\ (b)}_{n,j}\ \forall j \in \{1, 2, \ldots, \mathrm{N}^l_\mathrm{h}\}$
5: $\quad k = \arg\max_j \left(\mathbf{h}^l_j - \mathbf{c}_j\right)$
6: $\quad \mathbf{T}^{(b)}_{n,k} = 1$
7: $\quad \mathbf{c}_k = \mathbf{c}_k + \frac{\mathrm{N}^l_\mathrm{h}}{\mathrm{N}}$
8: **end for**
9: Remap $\mathbf{T}^{(b)}$ to active/inactive values

to generate the sparse target $\mathbf{T}^{(b)}$ from $\mathbf{H}^{l(b)}$. For the sake of simplicity, every step of the algorithm where the subscript j (feature index) appears must be applied $\forall j \in \{1, 2, \ldots, \mathrm{N}^l_\mathrm{h}\}$.

Starting with no activation in $\mathbf{T}^{(b)}$ (line 1) and the feature matrix $\mathbf{H}^{l(b)}$ normalized between [0,1] to adapt the method to any active/inactive values (line 2), the algorithm proceeds as follows. A row vector $\mathbf{h}^l$ from $\mathbf{H}^{l(b)}$ is processed at each iteration (line 4). The crucial step is performed in line 5: the feature k that has to be activated in the n^{th} row of $\mathbf{T}^{(b)}$ is selected as the one that has the maximal activation value $\mathbf{h}^l_j$ minus the inhibitor $\mathbf{c}_j$. The inhibitor $\mathbf{c}_j$ can be seen as an accumulator that "counts" the number of times a feature j has been selected, increasing its inhibition progressively by $\mathrm{N}^l_\mathrm{h}/\mathrm{N}$ until reaching maximal inhibition. This prevents the selection of a feature that has already been activated $\mathrm{N}/\mathrm{N}^l_\mathrm{h}$ times. The rationale behind the equation in line 5 is that, while selecting the maximal responses in $\mathbf{H}^{l(b)}$, we have to take care to distribute them evenly among all features (in order to ensure Strong Lifetime Sparsity). In line 6, the algorithm activates the k^{th} element of n^{th} row of the sparse target $\mathbf{T}^{(b)}$. By activating the "relative" maximum, we approximate property (3). After that, the inhibitor $\mathbf{c}$ is updated in line 7. Finally, in line 9, the complete sparse target $\mathbf{T}^{(b)}$ is remapped to the active/inactive values of the chosen activation function, e.g. $\{-1, 1\}$ for tanh and identity function (see Section 3.2.2).

3.2.2 System and Optimization Strategies

Let us assume that we have a system composed of the l-th layer of a deep architecture parameterized by $\theta^l = \{\mathbf{W}^l, \mathbf{b}^l\}$, with activation function f, which takes as input an input vector

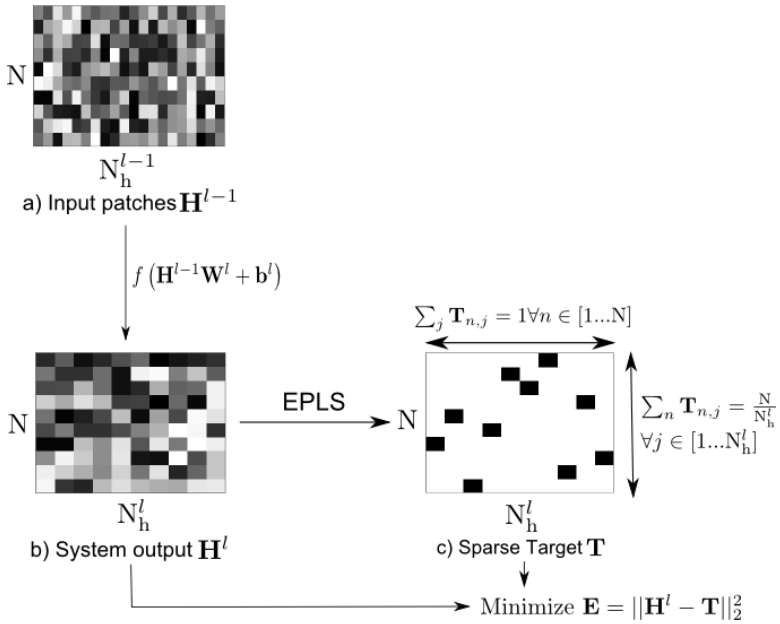


Figure 3.1: EPLS pipeline (see text for details). For simplicity, we assumed $\mathrm{N_b} = \mathrm{N}$ and omitted the super-script (b) without loss of generality.

$\mathbf{h}^{l-1}$ and produces a feature vector $\mathbf{h}^l = f(\mathbf{h}^{l-1}, \theta^l)$. We use the same notation as in Section 3.2 and define a data matrix $\mathbf{H}^{l-1}$ composed of N rows and $\mathrm{N_h^{l-1}}$ columns, where $\mathrm{N_h^{l-1}}$ is the input dimensionality.

The feature matrix $\mathbf{H}^l \in \mathbb{R}^{\mathrm{N} \times \mathrm{N_h^l}}$ is computed as

$$\mathbf{H}^l = f\left(\mathbf{H}^{l-1}\mathbf{W}^l + \mathbf{b}^l\right), \tag{3.1}$$

where f is an element-wise non-linearity. The proposed training algorithm is independent of the activation function f. To compare our training strategy to previous well known systems, we tested our algorithm using the logistic, scaled tanh (as suggested in [27]) and identity functions as element-wise non-linearity. When f is the identity function, an extremely fast algorithm can be used to train the system, as detailed in subsection 3.2.2.2.

3.2.2.1 Standard Optimization Strategy

The system is trained by means of an off-the-shelf mini-batch Stochastic Gradient Descent (SGD) method with adaptive learning rates such as variance-based SGD (vSGD) [44]. Algorithm 2 details the latter training process. The mini-batch size $\mathrm{N_b}$ can be set to any value, in all the experiments we have set $\mathrm{N_b} = \mathrm{N_h}$. Starting with θ^l set to small random numbers

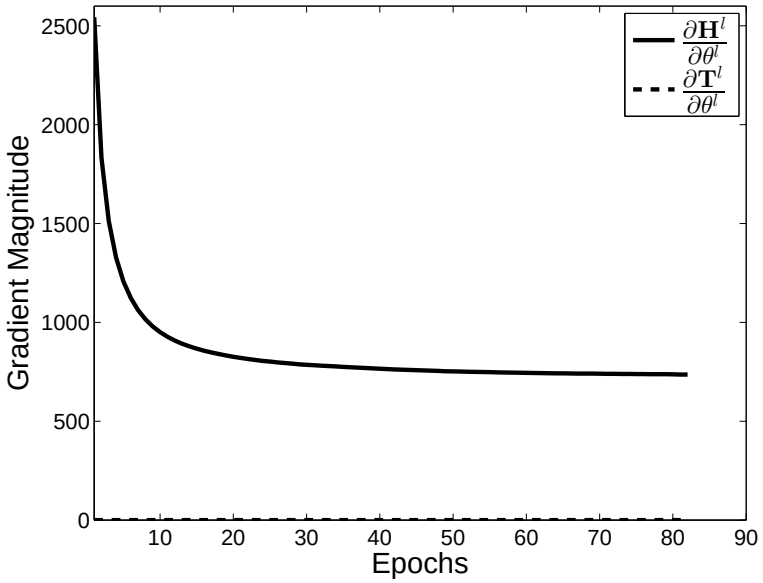


Figure 3.2: Magnitude of $\frac{\partial \mathbf{H}^l}{\partial \theta^l}$ and $\frac{\partial \mathbf{T}}{\partial \theta^l}$ throughout the training epochs.

as in [27] (line 1), at each epoch we shuffle the samples of the training set (line 3), reset the EPLS inhibitor $\mathbf{c}$ to zero (line 4) and process all mini-batches. For each mini-batch b, samples $\mathbf{H}^{l-1(b)}$ are selected (line 6); the feature matrix $\mathbf{H}^{l(b)}$ is computed (line 7); and the EPLS is invoked to compute the sparse target $\mathbf{T}^{(b)}$ and update the inhibitor $\mathbf{c}$ (line 8). After that, the gradient of the error is computed (line 9) and the learning rate ϵ is estimated as in [44] (line 10). The system parameters are then updated to minimize the L_2 error $E = ||\mathbf{H}^l - \mathbf{T}||_2^2$ (line 11). Finally, the weights $\mathbf{W}^l$ in θ^l are limited to have unit norm (line 13) to: (1) avoid saturation of non-linear activation functions, (2) help generating smoother mappings and, (3) avoid numerical issues due to large weights. This procedure is repeated until a stop condition is met; in our experiments, when the relative decrement error between epochs is small ($< 10^{-6}$).

When updating the system parameters, we assume that $\mathbf{T}^{(b)}$ does not depend on θ^l, thus $\frac{\partial \mathbf{T}^{(b)}}{\partial \theta^l} = 0$; we carried out experiments that show that this approximation does not significantly influence the gradient descent convergence nor the quality of the minimization. Moreover, this assumption makes the algorithm much faster, since we remove the need of computing the numerical partial derivatives of $\mathbf{T}^{(b)}$. Figure 3.2 shows how $\frac{\partial \mathbf{H}^{l(b)}}{\partial \theta^l}$ predominates over $\frac{\partial \mathbf{T}^{(b)}}{\partial \theta^l}$ throughout the training epochs.

Note that this optimization strategy is valid for any activation function, since the feature matrix $\mathbf{H}^{l(b)}$ is normalized within the EPLS algorithm (see Algorithm 1). Moreover, the mini-

batch vSGD allows to scale the algorithm easily, especially with respect to the number of samples N.

Algorithm 2 Standard EPLS training

Input: $\mathbf{H}^{l-1}$
Output: θ^l
$\theta =$ small random values
repeat
  Shuffle $\mathbf{H}^{l-1}$ randomly
  $\mathbf{c} = 0$
  for $b = 1 \rightarrow \lfloor \mathrm{N}/\mathrm{N_b} \rfloor$ **do**
    Select mini-batch samples $\mathbf{H}^{l-1\,(b)}$
    $\mathbf{H}^{l\,(b)} = f(\mathbf{H}^{l-1\,(b)}, \theta^l)$
    $(\mathbf{T}^{(b)}, \mathbf{c}) = EPLS(\mathbf{H}^{l\,(b)}, \mathbf{c}, \mathrm{N})$
    $\mathbf{G} = \nabla_{\theta^l} ||\mathbf{H}^{l\,(b)} - \mathbf{T}^{(b)}||_2^2$
    Estimate learning rate ϵ as in [44]
    $\theta^l = \theta^l - \epsilon \mathbf{G}$
  end for
  Limit the weights $\mathbf{W}^l$ in θ to have unit norm
until stop condition verified

Figure 5.1 summarizes the steps of the proposed method. Given an input $\mathbf{H}^{l-1} \in \mathbb{R}^{\mathrm{N} \times \mathrm{N_h^{l-1}}}$ (a), the algorithm computes its feature output $\mathbf{H}^l \in \mathbb{R}^{\mathrm{N} \times \mathrm{N_h^{l-1}}}$ by applying the activation function f (b), invokes EPLS to generate the sparse target $\mathbf{T}$ (c) and then learns its parameters θ^l by minimizing $E = ||\mathbf{H}^l - \mathbf{T}||_2^2$.

3.2.2.2 Special Optimization Case: Identity Activation

An interesting case appears when the activation function is the identity, since the linear system $\mathbf{H}^{l-1\,(b)}\mathbf{W}^l = \mathbf{T}^{(b)}$ can be solved by alternately fixing $\mathbf{W}^l$ and $\mathbf{T}^{(b)}$ for $\mathrm{N} \geq \mathrm{N_h^{l-1}}\mathrm{N_h^l}$, where the bias $\mathbf{b}^l$ of each feature is incorporated in $\mathbf{W}^l$ as an additional weight and a column of ones is attached at the end of $\mathbf{H}^{l-1\,(b)}$. In this context, we set $\mathrm{N_b} = \mathrm{N}$; thus, in the remaining part of the explanation, we will drop the super-script (b).

In this particular case, at each epoch, $\mathbf{W}^l$ can be estimated in closed form for a fixed $\mathbf{T}$. Hence, the system can be trained by means of an extremely fast algorithm. The optimization process alternatively fixes $\mathbf{W}^l$ and $\mathbf{T}$ to find $\mathbf{H}^l$, which better approximates $\mathbf{T}$. Starting with a random initialization of $\mathbf{W}^l$ as in [27], we compute the pseudo-inverse $\mathbf{H}_{\mathrm{inv}}^{l-1}$ of the input data $\mathbf{H}^{l-1}$. Note that this operation is performed **only once** before the optimization process. Then,

the optimization proceeds as follows: Starting from a flat activation in the EPLS inhibitor $\mathbf{c}$, (1) the output of the system is computed using a fixed $\mathbf{W}^l$; (2) EPLS algorithm is invoked to generate the sparse target $\mathbf{T}$ and; (3) the parameters $\mathbf{W}^l$ are updated keeping $\mathbf{T}$ fixed. The procedure is repeated until a stop condition is met. Note that the described algorithm is especially useful when the training set is relatively small, since the optimization process becomes extremely fast. However, when the training set is large, and computing the pseudo-inverse of $\mathbf{H}^{l-1}$ becomes critical, the standard optimization process in Algorithm 2 can be used with the identity activation.

3.3 Experiments on Benchmark Datasets

The performance of training and encoding strategies in single layer networks has been extensively analyzed in the literature [7, 66, 90] on CIFAR-10[1], STL-10[2] and UCMerced[3] datasets. CIFAR-10 [91] dataset provides a fully labeled training set used for both unsupervised and supervised learning, whereas STL-10 [7] provides a large unlabeled training set for unsupervised learning and a small number of labeled samples for supervised learning. UCMerced [92] presents a less heterogeneous dataset, with highly overlapping classes and fewer images per class. In this section, we will evaluate the proposed unsupervised feature learning algorithm on the three benchmark datasets.

3.3.1 Test with a Few Supervised Examples: STL-10

STL-10 dataset consists of 96×96 pixels color images belonging to 10 different classes. The dataset is divided into a large unlabeled training set containing 100K images and smaller labeled training and test sets, containing 5K and 8K images, respectively. The test set is divided into 10 folds of 1K images. It has to be considered that in STL-10, the primary challenge is to make use of the unlabeled data (100K images), which is 100 times bigger than the labeled data used to train the classifier (1K per fold). In this case, the supervised training must strongly rely on the ability of the unsupervised method to learn discriminative features. Moreover, since the unlabeled dataset contains other types of animals (bears, rabbits, etc.) and vehicles (trains,

[1] http://www.cs.toronto.edu/$\sim$kriz/cifar.html
[2] http://www.stanford.edu/$\sim$acoates/stl10/
[3] http://vision.ucmerced.edu/datasets/landuse.html

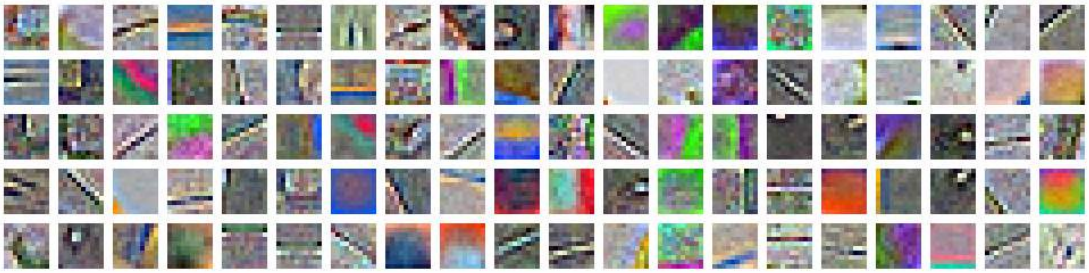


Figure 3.3: Random subset of filters learned by EPLS using the logistic activation function, a receptive field of 10×10 pixels and $\mathrm{N}_{\mathrm{h}}^{l} = 1600$ (better seen in color).

buses, etc.) in addition to the ones in the labeled set, the unsupervised method should be able to generalize well.

To validate our method, we follow the experimental pipeline of [7]. We first extract random patches and normalize them for local brightness and contrast. Note that EPLS does not require any whitening of the input data. Then, we train the system in a patch-based fashion. After that, we apply the system to retrieve sparse features of patches covering the input image, sum-pool them into 4 quadrants and finally train a L_2 Support Vector Machine (SVM) for classification. We tune the SVM parameter using 5-fold cross-validation. As in [67], we use a receptive field of 10×10 pixels and a stride of 1. The number of features is set to $\mathrm{N}_{\mathrm{h}}^{l} = 1600$ for fair comparison with state-of-the-art methods. We also provide the results of our method with polarity split ($\mathrm{N}_{\mathrm{h}}^{l} = 1600 \times 2$, using $\mathbf{W}^{l}$ and $-\mathbf{W}^{l}$ for encoding as in [66]) and using the sparse coding (SC) encoder, which [66] found to be the best when small number of labeled data is available. For this encoder, we searched over the same set of parameter values as [66], i.e., $\lambda = \{0.5, 0.75, 1.0, 1.25, 1.5\}$. Note that the parameter λ is part of the encoder and, thus, does not belong to the unsupervised learning method that we propose.

Table 3.2 summarizes the results obtained on this dataset compared to state-of-the-art methods at the time we carried out the research. When pairing each training method with its associated natural encoding, EPLS outperforms all other methods, independently of the activation function. When pairing the training methods with sparse coding, EPLS outperforms the state-of-the-art best performer in single layer networks as well, achieving 61.0% (0.58%) accuracy when the logistic function is used to train the system's parameters. Moreover, the standard deviation of the folds is lower than the one provided by OMP-1 with sparse coding encoding.

[4] We used a rectified linear unit as natural encoding of tanh and linear activations to avoid sum pooling cancelations.

Table 3.2: EPLS Classification accuracy on STL-10. Natural and Sparse Coding (SC) refer to the type of encoding used after training the network (see Chapter 2). Complete denotes a system with equal number of inputs and outputs.

Algorithm			**Accuracy**
*Single-Layer **with** hyper-parameters*			
RICA [70] (1600/Natural)			52.9%
OMP-k ($k = 1$) (1600/Natural)			51.8% (0.47%)
OMP-k ($k = 1$) (whitening, 1600/Natural)			53.1% (0.52%)
OMP-k ($k = 1$) (whitening, 1600 $\times$ 2/Natural)			54.5% (0.66%)
OMP-k ($k = 1$) (whitening, 1600 $\times$ 2/SC)			59.0% (0.80%)
*Single-Layer **without** hyper-parameters*			
Raw pixels			31.8% (0.62%)
ICA (whitening, Complete/Natural)			48.0% (1.47%)
K-means-tri (whitening, 1600)			51.5% (1.73%)
Sparse Filtering (1600/Natural)			53.5% (0.53%)
EPLS			
Enc. / f	Natural (1600)	Natural (1600x2)	SC (1600x2)
tanh[4]	55.4% (0.70%)	55.4% (0.62%)	59.2% (0.64%)
logistic	56.6% (0.66%)	56.9% (0.50%)	**61.0% (0.58%)**
identity[4]	55.4% (0.76%)	55.6% (0.59%)	58.7% (0.85%)

Results are even more impressive if we compare them to hyper-parameter free algorithms.

Figure 3.3 shows a subset of 100 randomly selected filters learned by our method using the logistic activation function, 10×10 pixel receptive field and a system of $\mathrm{N}_{\mathrm{h}}^{l} = 1600$ features. As shown in the figure, the method learns not only common filters such as oriented edges/ridges in many directions and colors but also corner detectors, tri-banded colored filters, center surrounds and Laplacian of Gaussians among others. This suggests that enforcing lifetime sparsity helps the system to learn a set of complex, rich and diversified bases.

3.3.2 Test with All Supervised Examples: CIFAR-10

CIFAR-10 dataset consists of 32×32 pixel color images belonging to 10 different classes. The dataset has a large amount of labeled data (50K images) to be used for training. We follow the experimental pipeline of [7], with a stride of 1 pixel, a receptive field of 6×6 pixels and a data normalization for local brightness and contrast. We do not perform any whitening of the input images. The number of features is again set to $\mathrm{N}_{\mathrm{h}}^{l} = 1600$, consistent with [7]. We also provide the results of our method with polarity split ($\mathrm{N}_{\mathrm{h}}^{l} = 1600 \times 2$) and using the soft-thresholding, which [66] found to be the best encoder on CIFAR-10. We searched over the same set of values as [66] to tune the value of the soft-threshold $t = \{0.1, 0.25, 0.5, 1.0\}$. Note that the parameter t is part of the encoder and, thus, does not belong to the unsupervised learning method that we propose.

Table 3.3 presents the results obtained on this dataset compared to state-of-the-art methods at the time we carried out the research. The results of sparse filtering [67] were computed following the same procedure, using the code provided by the authors[5]. As reported in Table 3.3, the performance of our training strategy matches the contemporary state-of-the art, while requiring no hyper-parameter tuning. When compared to training methods using their natural encoding, our approach provides results that outperform the state-of-the-art accuracy. When considering soft-threshold encoding, OMP-1 provides slightly higher performance. When the hyper-parameter k is properly tuned, OMP-k achieves better results. However, as suggested by the results, OMP-k strongly relies on data whitening previous to the parameters learning (decreasing its performance by 5.9% when no whitening is used and $k = 1$). The whitening pre-processing might be critical, since it can not always be computed exactly for high dimensional data [70]. Moreover, when setting the number of features to $\mathrm{N}_{\mathrm{h}}^{l} = 6000$, EPLS matches OMP-$k$ performance, with a performance gain of 3.29% w.r.t. $\mathrm{N}_{\mathrm{h}}^{l} = 1600$. Sparse coding also provides competitive results when properly combining its training and inference parameters.

In [66], authors do not report the results of sparse coding using its natural encoding (i.e., using the same λ for both training and encoding). It has to be noted that the method involves an expensive training (with very slow convergence), an expensive inference (with an extra loop of optimization) as well as hyper-parameter tuning to achieve best performance. When compared to single layer training methods without hyper-parameters, EPLS paired with its natural encoding outperforms state-of-the-art results, except for k-means (tri), whose performance is significantly higher than the others. It has to be considered that the CIFAR-10 dataset provides a large set of labeled data, so that the pipeline performance depends on the proper combination of unsupervised and supervised learning algorithms. As stated in [7], k-means coupled with a triangle activation and whitening especially benefits from the large amount of labeled data that CIFAR-10 provides. This is stressed by its significantly lower ranking on STL-10 (-5.1% w.r.t. our method paired with its natural encoding and the same number of features).

3.3.3 Test with Less Heterogeneous Dataset: UCMerced

UCMerced dataset consists of 256×256 pixels color images belonging to 21 aerial scene categories. The dataset contains highly overlapping classes and therefore, is less heterogeneous than both CIFAR-10 and STL-10. The dataset has only 100 images per class. We follow the experimental pipeline described in [90] and randomly select 80 images from each class to be used for training and leave the 20 remaining ones for testing. We report the mean accuracy obtained over five runs.

To validate our method, we reproduce the experimental setting of [90] and use a receptive field of 16×16 pixels, a stride of 8 pixels and $\mathrm{N}_{\mathrm{h}}^{l} = 1000$ features. We first extract random patches from the training images, normalize them for local brightness and contrast and use them to train the system. We train the system by means of EPLS with logistic activation function. After that, we apply the system to retrieve sparse features using natural encoding with polarity split, to be consistent with the number of features in [90]. We sum-pool the features into 4 quadrants and, finally, train a linear SVM for classification. As in the previous experiments, we tune the SVM parameter using 5-fold cross-validation.

[5] lhttp://cs.stanford.edu/$\sim$jngiam/papers/NgiamKoh\ChenBhaskarNg2011_Supplementary.pdf

[6] We used a rectified linear unit as natural encoding of tanh and linear activations to avoid sum pooling cancelations.

Table 3.3: EPLS Classification accuracy on CIFAR-10.

Algorithm			**Accuracy**
*Single-Layer **with** hyper-parameters*			
Sparse RBM (1600/Natural)			72.4%
Sparse auto-encoder (1600/Natural)			73.4%
Sparse coding (1600x2/different λ)			78.8%
OMP-k ($k = 1$) (whitening, 1600x2/Natural)			71.4%
OMP-k ($k = 1$) (1600x2/t)			73.5%
OMP-k ($k = 1$) (whitening, 1600x2/t)			79.4%
OMP-k ($k = 10$) (whitening, 1600x2/t)			80.1%
OMP-k ($k = 10$) (whitening, 6000x2/t)			**81.5%**
*Single-Layer **without** hyper-parameters*			
Raw pixels			37.3%
K-means-hard (whitening, 1600/Natural)			68.6%
K-means (whitening, 1600/tri)			77.9%
Sparse Filtering (1600/Natural)			71.2%
EPLS			
Enc. / f	Natural (1600)	Natural (1600x2)	t (1600x2)
tanh[6]	77.1%	77.2%	78.9%
logistic	74.7%	75.8%	77.3%
identity[6]	76.4%	76.5%	78.4%
tanh (6000x2/t)			**81.5%**

Table 3.4: EPLS Lifetime and Population Sparsity.

Method	Lifetime (KL)	Population (Kurtosis)
Sparse Coding[7]	0.0371	45.44
Sparse Auto-encoders	0.0002	18.24
OMP-1[7]	0.0869	45.55
Sparse Filtering	0.0189	3.48
EPLS (logistic)	0.0006	27.42
EPLS (tanh)[7]	0.0081	7.47
EPLS (linear)[7]	0.0068	10.11

We achieve an accuracy of $74.34 \pm 3.0\%$, which is significantly higher than the $62.7 \pm 1.72\%$ reported in [90] when pairing OMP-1 training with soft-threshold encoding (tuned to achieve maximum performance) on raw pixels. Note that when considering the result reported in [90] using OMP-1 with its natural encoding on normalized raw pixels ($13.86 \pm 1.31\%$), our method's results are even more impressive.

The reported results suggest that the EPLS is able to learn good representations even when training from fewer images and exhibits an especially remarkable performance on less heterogeneous datasets.

3.4 Analysis of Population and Lifetime Sparsity

It is interesting to analyze how well our method can achieve population and lifetime sparsity, and compare it to state-of-the-art methods, which claim to seek a sparse feature representation of the data. We trained single layer systems by means of sparse coding, sparse auto-encoders, OMP-1, sparse filtering and EPLS on $\mathrm{N} = 400\mathrm{K}$ color patches of 6x6 pixels of CIFAR-10 dataset, and only $\mathrm{N}_{\mathrm{h}}^{l} = 100$ features. The majority of state-of-the-art methods ensure lifetime sparsity by enforcing similar mean activation among features [5, 67]. To evaluate how well these methods achieve their objective, we take the training set and generate the output features of the system. After that, we compute the mean activation of each feature and normalize them (dividing by the sum of all feature's mean activations) in order to obtain a probability distribution. Then, we compute the Kullback-Leibler divergence (KL) between the resulting probability distribution and an uniform (flat) distribution with equal probability $\frac{1}{\mathrm{N}_{\mathrm{h}}^{l}}$. Note that this is not a strict measure of lifetime sparsity, but a measure of the activation statistics among

[7] We applied a rectified linear unit to the features extracted by these methods.

features. The lower the value, the closer to the objective of maintaining similar mean activation among features. Likewise, we want to evaluate all these methods in terms of population sparsity. To ensure population sparsity, we want a small subset of features to be strongly active at the same time. Therefore, we compute the population kurtosis, as proposed in [88], to measure the infrequency of strong neural response. The population kurtosis increases as the population sparsity increases. Table 3.4 summarizes the obtained results. OMP-1 and sparse coding typically enforce population sparsity and, thus, exhibit the highest population sparsity level among methods. However, these methods do not achieve good lifetime sparsity, e.g. OMP-1 potentially leaves a significant amount of dead (or almost dead) features. Sparse filtering claims to ensure both population and lifetime sparsity, but seems to achieve the lowest level of both sparsities when compared to other state-of-the-art methods. Sparse auto-encoders explicitly enforce lifetime sparsity. Results show that the method achieves one of the highest lifetime sparsity levels when tuned to achieve maximum performance. Moreover, the method also provides reasonable population sparsity. As shown in the table, the EPLS achieves a good compromise between lifetime and population sparsity by imposing sparsity in the objective function in a very strict way. Note that the extreme form of quantization in the EPLS sparse target will never be reached while training, since we constrain the filters to have at most unit norm (see Algorithm 2 line 13).

To highlight the benefits of encouraging Strong Lifetime Sparsity, we analyze the impact of allowing each feature to activate for a different number of inputs. To do so, we apply a coefficient $\lambda \leq 1$ to the ratio $\mathrm{N_h^l/N}$ in line 7 of Algorithm 1 to decrease the inhibition of each feature and allow each feature to activate $\frac{1}{\lambda}$ times more than the others, while maintaining Strong Population Sparsity property. For each λ, we trained a network of $\mathrm{N_h^l} = 100$ features and measured the accuracy of the network as well as the achieved lifetime sparsity.

Figure 3.4 shows the accuracy as a function of the lifetime sparsity (KL). We also evaluated the percentage of dead features of each network, reported between brackets. Enforcing Strong Lifetime Sparsity ($\lambda = 1$) achieves maximum accuracy and lowest KL. The farther we move from the Strong Lifetime Sparsity definition (λ decreases), the higher the KL and the lower the accuracy. Moreover, dead features do not appear until $\lambda \leq \frac{1}{32}$. Therefore, the reduced lifetime sparsity is the only responsible for the performance drop when λ is slightly smaller than 1. Note that, for $\lambda \leq \frac{1}{32}$, the performance is also affected by the presence of dead features.

This experiment suggests that enforcing Strong Lifetime Sparsity is indeed crucial to achieve good results.

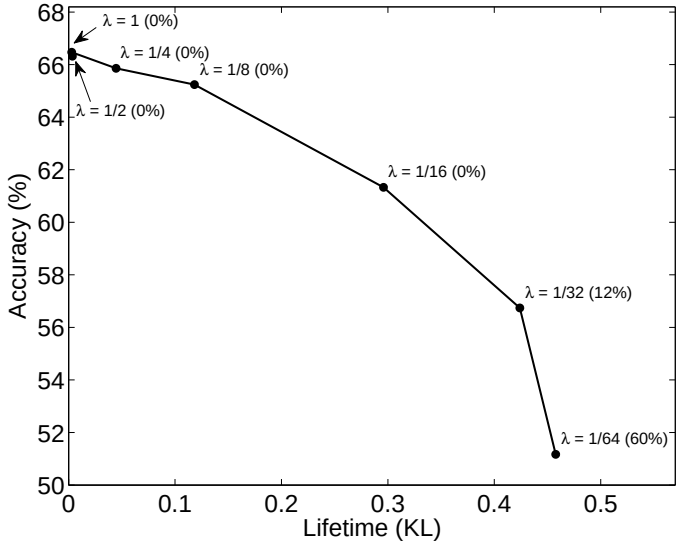


Figure 3.4: Accuracy changing lifetime sparsity in a single layer network of $\mathrm{N}_\mathrm{h}^l = 100$. The percentage of dead features of each network is reported between brackets

3.5 Computational Complexity

The EPLS algorithm requires the computation of $\mathbf{T}$, which has $\mathcal{O}(\mathrm{N}\mathrm{N}_\mathrm{h}^l)$ cost, and therefore scales linearly on both N and N_h^l. Since we can use vSGD for optimization, the method scales linearly on N given a fixed number of epochs. Finally, applying the activation function, the cost of computing the derivative is linear with $\mathrm{N}_\mathrm{h}^{l-1}$, since we use the analytical expression of $\frac{\partial E}{\partial \theta^l}$, with $\frac{\partial \mathrm{T}}{\partial \theta^l} = 0$.

The memory complexity is related to the mini-batch size N_b. Consequently, the method can scale gracefully to very large datasets: theoretically, it requires to store in memory the mini-batch input data $\mathbf{H}^{l-1(b)}$ ($\mathrm{N}_\mathrm{b}\mathrm{N}_\mathrm{h}^{l-1}$ elements), output $\mathbf{H}^{l-1(b)}$ ($\mathrm{N}_\mathrm{b}\mathrm{N}_\mathrm{h}^{l}$ elements), target $\mathbf{T}^{(b)}$ ($\mathrm{N}_\mathrm{b}\mathrm{N}_\mathrm{h}^{l}$ elements) and the system parameters to optimize θ ($\mathrm{N}_\mathrm{h}^{l}\left(\mathrm{N}_\mathrm{h}^{l-1}+1\right)$ elements); a total amount of $\mathrm{N}_\mathrm{h}\left(\mathrm{N}_\mathrm{h}^{l-1}+1\right)+\mathrm{N}_\mathrm{b}\left(\mathrm{N}_\mathrm{h}^{l}+2\mathrm{N}_\mathrm{h}^{l}\right)$ elements.

3.6 Discussion

Our results show that simultaneously enforcing both population and lifetime sparsity helps in learning discriminative filters, which translates into better performance, especially when compared to other hyper-parameter free methods [67, 70]. Experiments suggest that our algorithm

is able to extract features that generalize well on unseen data. When comparing the performance on CIFAR-10 and STL-10 datasets, our algorithm ranks much better on the STL-10 than on CIFAR-10. Note that on STL-10, we outperform contemporary state-of-the-art best performers, whereas on CIFAR-10, we match the best performer at the time this research was published (OMP-10 in conjunction with a soft-threshold encoding). Results on STL-10 suggest that our algorithm helps the classifier in generalizing with a few training examples (1% of the dataset), gaining 2% accuracy w.r.t. the state-of-the art best performer (OMP-1 paired with sparse coding) with a lower standard deviation across folds, suggesting more robustness to variations in the training folds. Moreover, on UCMerced, the EPLS exhibits extremely compelling results when compared to OMP-1. This suggests that sufficiently heterogeneous datasets might help state-of-the-art algorithms to implicitly achieve lifetime sparsity.

It is important to highlight that OMP-1 can be seen as a special case of our algorithm, where the activation function is $|\mathbf{H}^{l-1}\mathbf{W}|$ and lifetime sparsity is not taken into account in the optimization process (potentially leading to dead features). Our algorithm has several advantages over OMP-1: (1) It can use any activation function; (2) by enforcing lifetime sparsity it does not suffer from the dead feature problem, thus not requiring ad-hoc tricks to avoid it; (3) it does not require whitening, which can be a problem if the input dimensionality is large [70].

With our proposal, we advance in the hyper-parameter free line of ICA [68] and sparse filtering [67]. It is clear that the advantage of sparse filtering over ICA comes from removing the orthogonality constraint, and imposing some sort of "competition" between features, which also permits overcomplete representations. Following this spirit, our algorithm imposes an even more strict form of competition to prevent dead features by means of Strong Lifetime Sparsity and confirms the trend of [67, 68] that data reconstruction seems not so important if the goal is to have a discriminative sparse system.

Last and most importantly, it is worth highlighting five interesting properties of the EPLS algorithm. First, the method is hyper-parameter free, which highly simplifies the training process for practitioners, especially when used as a greedy pre-training method in deep architectures. Second, the method is fast and scales linearly with the number of training samples and the input/output dimensionalities. Third, EPLS is easy to implement. We implemented the EPLS in Algorithm 1 in less than 50 lines of C code. The mini-batch vSGD is a general purpose optimizer; our Matlab implementation of vSGD plus the EPLS mex source are publicly available[8]. Fourth, the proposed learning strategy is not limited to perceptrons. Fifth, there is an interest in the literature in avoiding redundancy in the image representation by performing

a convolutional training [63] instead of a patch-based one. For this purpose, the EPLS can be slightly modified to apply the procedure to a whole image at once and consider the mini-batch size to be the image divided into patches. Furthermore, combining the EPLS in transformation-invariant frameworks such as [93] is potentially interesting. These aspects are not considered in the thesis and are left for future investigation.

3.7 Summary

In this chapter, we presented a **hyper-parameter free, off-the-shelf, simple and computationally efficient** approach for unsupervised sparse feature learning. The method seeks both lifetime and population sparsity in an explicit way in order to learn discriminative features. We evaluated the method on single layer systems following the standard state-of-the-art pipeline. Experiments were aimed at highlighting the importance of explicitly enforcing lifetime sparsity to achieve good results. Finally, results on benchmark datasets showed that the method significantly outperforms hyper-parameter free best performers, as well as many other state-of-the-art methods, even when dealing with fewer samples and less heterogeneous datasets.

[8] https://sites.google.com/site/adriromsor/epls

4

Training deep architectures by means of EPLS

In Chapter 3, we introduced EPLS, an unsupervised learning algorithm to extract sparse feature representations. The algorithm was evaluated on single layer models, demonstrating its potential to provide discriminative features and achieving state-of-the-art results. Yet the question of training deep architectures with the proposed unsupervised algorithm remains unanswered. In this chapter, we will exploit the properties of the EPLS to train deep architectures in a greedy layer-wise fashion [15, 16]. To that end, we choose two different applications that could benefit from our approach. The first application, that will be discussed in Section 4.1, is remote sensing image and pixel classification. We will use this context to perform a detailed analysis of both (1) the benefits of our method to train deep architectures; and (2) the influence of several deep architecture's hyper-parameters on the system's performance. The second application, that will be tackled in Section 4.2, is image parsing. We will use this application to extend the method and make it proficient at the image parsing task.

4.1 Application to Remote Sensing Data

Earth observation through remote sensing techniques is a research field where a huge variety of physical signals is measured from instruments on-board space and airborne platforms. A wide diversity of sensor characteristics is nowadays available, ranging from medium and very high resolution (VHR) multispectral imagery to hyper-spectral images that sample the electromagnetic spectrum with high detail. These myriad of sensors serve to particularly different objectives, focusing either on obtaining quantitative measurements and estimations of geo-bio-physical variables, or on the identification of materials by the analysis of the acquired images [94, 95, 96]. Among all the different products that can be obtained from the acquired images, segmentation maps are perhaps the most relevant ones. The remote sensing image classification and pixel classification problems are very challenging and ubiquitous because land cover and land use maps are mandatory in multi-temporal studies and constitute useful inputs to other processes.

Despite the high number of advanced, robust and accurate existing classifiers [97], the field faces very important challenges:

1. *Complex statistical characteristics of images:* The statistical properties of the acquired images pose important difficulties for automatic classifiers. The analysis of these images turns out to be very challenging, especially because of the high dimensionality of the pixels, the specific noise and uncertainty sources observed, the high spatial and spectral

redundancy and collinearity, and their potentially non-linear nature[1]. Beyond these well-known data characteristics, we should highlight that spatial and spectral redundancy also suggest that the acquired signal may be better described in *sparse* representation spaces, as recently reported in [97, 99, 100, 101].

2. *High computational problems involved.* We are witnessing the advent of a Big Data Era, especially in remote sensing data processing. The upcoming constellations of satellite sensors will acquire a large variety of heterogeneous images of different spatial, spectral, angular and temporal resolutions. In fact, we are witnessing an ever increasing amount of data gathered with current and upcoming earth observation satellite missions, from multispectral sensors like Landsat-8 [102], to VHR sensors like WorldView-III [103], the super-spectral Copernicus' Sentinel-2 [104] and Sentinel-3 missions [105], as well as the planned EnMAP [106], HyspIRI [107] and ESA's candidate FLEX [108] imaging spectrometer missions. This data flux will require computationally efficient classification techniques. The current state-of-the-art SVM [109, 110] is not, however, able to cope with more than some few thousands of labeled data points.

3. *Scarcity of labeled data.* Remote sensing data is typically characterized by the insufficiency of labeled data. As in many application domains, collecting labeled data has a very high cost in terms of time and human resources, whereas unlabeled data is relatively easier to obtain.

A convenient way to alleviate these problems is to extract relevant, potentially useful, non-redundant, non-linear features from images in an unsupervised fashion in order to facilitate the subsequent classification step. The bottleneck would then be the *unsupervised feature learning* step. Thus, learning expressive *spatial-spectral features* from hyper-spectral images in an *efficient* way is of paramount relevance.

Therefore, we propose to train convolutional deep architectures in a greedy layer-wise fashion [15, 111] by means of EPLS unsupervised learning algorithm [21] introduced in Chapter 3. Convolutional deep architectures allow us to address the first challenge in remote sensing, given (1) their highly non-linear nature, well-suited to cope with the difficulties of non-linear spatial-spectral image analysis; and (2) their ability to capture local interactions, appropriate

[1]Factors such as multi-scattering in the acquisition process, heterogeneities at sub-pixel level, as well as atmospheric and geometric distortions lead to distinct non-linear feature relations, since pixels lie in high dimensional curved manifolds [97, 98].

when the input shares similar statistics at all location, i.e. when there is high redundancy. Moreover, the EPLS unsupervised learning algorithm allows to handle both the second and third previously mentioned remote sensing challenges, since (1) it provides a *purely unsupervised* pipeline to train each layer of a deep architecture, not suffering from labeled data scarcity; and (2) it is able to handle large numbers of high dimensional data efficiently, alleviating the high computational problems. In addition to that, EPLS enforces a sparse representation of the data, which is supposed to be convenient to describe remote sensing data [97, 99, 100, 101]. The proposed pipeline is applied to retrieve hierarchical sparse representations of remote sensing data, which are then used for image classification and pixel classification. Experimental results outline the applicability and potential of the method introduced in the previous chapter to extract potentially useful hierarchical representations of hyper-spectral, very high resolution and multispectral images.

4.1.1 Related Work

Given the typically high dimensionality of remote sensing data, feature extraction techniques have been widely used in the literature to reduce the data dimensionality. While the classical Principal Component Analysis (PCA) [112] is still one of the most popular choices, a plethora of non-linear dimensionality reduction methods, manifold learning and dictionary learning algorithms have been introduced in the last decade.

State-of-the-art *manifold learning* methods [113] include: local approaches for the description of remote sensing image manifolds [114]; kernel-based and spectral decompositions that learn mappings optimizing for maximum variance, correlation, entropy, or minimum noise fraction [115]; neural networks that generalize PCA to encode non-linear data structures via autoassociative/autoencoding networks [116]; as well as projection pursuit approaches leading to convenient Gaussian domains [117]. In remote sensing, auto-encoders have been widely used [118, 119, 120, 121]. However, a number of (critical) free parameters are to be tuned; regularization is an important issue, which is mainly addressed by limiting the network's structure heuristically; and only shallow (or not very deep) structures are considered due to the limitations on computational resources and efficiency of the training algorithms. On top of this, very often, auto-encoders employ only the spectral information, and in the best of the cases, spatial information is naively included through stacking hand-crafted spatial features.

In recent years, *dictionary learning* has emerged as an efficient way to learn sparse image features in unsupervised settings, which are eventually used for image classification and

object recognition: discriminative dictionaries have been proposed for spatial-spectral sparse-representation and image classification [122], sparse kernel networks have been recently introduced for classification [123], sparse representations over learned dictionaries for image pansharpening [124], saliency-based codes for segmentation [125, 126], sparse bag-of-words codes for automatic target detection [127], and unsupervised learning of sparse features for aerial image classification [90]. These methods describe the input images in sparse representation spaces but do not take advantage of the high non-linear nature of deep architectures.

Finally, there is some evidence of the good performance of deep architectures in remote sensing image classification: [128] introduces a deep learning algorithm for classification of (low-dimensional) VHR images; [129] explores the robustness of deep networks to noisy class labels for aerial image classification; and [130] introduces hybrid Deep Neural Networks to enable the extraction of variable-scale features to detect vehicles in satellite images; [131] proposes a hybrid framework based on Stacked Auto-Encoders for classification of hyper-spectral data. Although deep learning methods can cope with the difficulties of non-linear spatial-spectral image analysis, the issues of sparsity in the feature representation and efficiency of training algorithms are not obvious in state-of-the-art frameworks.

4.1.2 Feature Learning Pipeline

The system we use to learn and extract features for this application is based on a convolutional deep architecture trained in a greedy layer-wise fashion. Each convolutional layer is trained by feeding random patches from their input feature maps to the EPLS algorithm. The input images of the first layer are normalized for contrast and brightness. Moreover, the input of each layer is standardized to help the training process. The hyper-parameters of the deep architecture are selected according to the experiment at hand. After learning the feature hierarchy, the higher level features (the ones coming from the last representation layer) are fed to a simple classifier, such as SVM or k-Nearest Neighbor with $k = 1$ (1-NN). Figure 4.1 shows how the trained deep architecture is built. In this case, we have a network with two convolutional layers and two pooling layers. We train the first convolutional layer by feeding random patches from the normalized input images to the EPLS. Then, we use the layer's parameters to extract the activation feature maps. After that, we apply the pooling layer. From the pooled feature maps, we extract random patches and feed them to the EPLS to train to the second convolutional layer. Finally, we compute the activation feature maps from the second layer, pool them and use them for classification purposes.

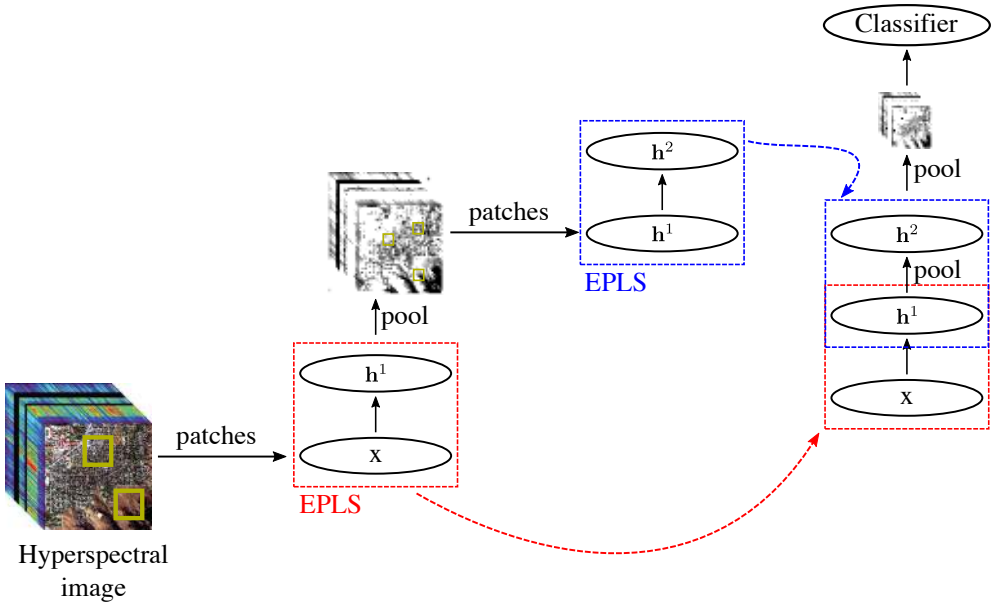


Figure 4.1: Remote sensing feature learning pipeline: Convolutional layers are trained in isolation one after another by means of EPLS and then stacked together with the pooling layers to extract the feature maps to be fed to the classifier.

4.1.3 Experiments

In this section, we will provide an in-depth analysis on the trends of the presented feature learning pipeline in different problems of image classification and pixel classification of remote sensing images. On one hand, in Section 4.1.3.1, we will extend the land use image classification problem of Chapter 3 by evaluating the effects of varying the number of features and the architecture's depth on a validation set. Moreover, we will analyze whether the benefits of sparsity are still applicable to deep scenarios. On the other hand, we will tackle the pixel classification problem in a wide diversity of scenarios differing in their input dimensionality, number of classes and amount of data available. In Section 4.1.3.2, we will consider a challenging hyper-spectral pixel classification problem to assess the impact of many architecture's hyper-parameters, such as the number of features, the number of layers, the size of the receptive field and the pooling regions. We will also investigate the robustness of the extracted features to the number of labeled samples used for training. Finally, we will explore the applicability of the method to segment very high spatial resolution images in Section 4.1.3.3 and multispectral images in Section 4.1.3.4.

4.1.3.1 Aerial Scene Classification

In this section, we will extend our experiments on UCMerced dataset[2]. Recall that UCMerced dataset contains 256×256 pixels color images from 21 aerial scene categories, with resolution one foot per pixel and highly overlapping classes. The images were manually extracted from the USGS National Map Urban Area Imagery collection. In order to validate our method, we follow the experimental setup described in [90] and randomly select 80% of the images per class for training and leave the remaining 20% ones for validation. As in [90], we report the mean accuracy obtained over five runs. From the training images, we train deep architectures as shown in Figure 4.1. Specifically, we (1) extract random patches and normalize them for contrast and brightness; (2) train a network by means of EPLS with logistic activation; (3) use the trained network parameters and an encoding strategy to retrieve sparse representations; (4) pool the upper-most feature map into four quadrants via sum-pooling; and (5) feed the pooled features to a linear SVM classifier. Following this pipeline, we define the following experimental settings to highlight the competitiveness of our method.

In the first setting, we aim to analyze the influence of varying the number of features $\mathrm{N}_{\mathrm{h}}^{1}$ in single layer networks. To do so, we use a receptive field of 15×15 pixels with stride 1 pixel and follow the previously described pipeline to train the system, extract the features and classify them. Figure 4.2 (left) shows the classification performance of our single layer approach (solid red line), compared to the best results reported in [90] (solid green line), for different $\mathrm{N}_{\mathrm{h}}^{1}$ values. As shown in the figure, our method outperforms the method in [90] for all $\mathrm{N}_{\mathrm{h}}^{1}$ in terms of average performance in the 5 runs, while having no training nor encoding meta-parameters to tune. Moreover, [90] requires to train the single layer network on SIFT features to achieve such performance, whereas we train all our networks on raw image patches normalized for contrast and brightness, i.e. we do not require any prior feature extraction.

To strengthen the results, we report the per class users and producers for the single layer network with $\mathrm{N}_{\mathrm{h}}^{1} = 1000$ and compare the producer's accuracy to the best results reported in [90, 92]. Figure 4.3 (left) shows the users and producers obtained by our approach. The proposed method achieves a very high sensitivity and specificity for most of the classes, in particular for chaparral, harbor, parking lot and runway. Errors are mainly in scenes with similar spatial structures, such as buildings and residential areas, for which the database contains three similar subclasses (medium, sparse and dense residential areas). We also compare in Figure 4.3

[2] http://vision.ucmerced.edu/datasets/landuse.html

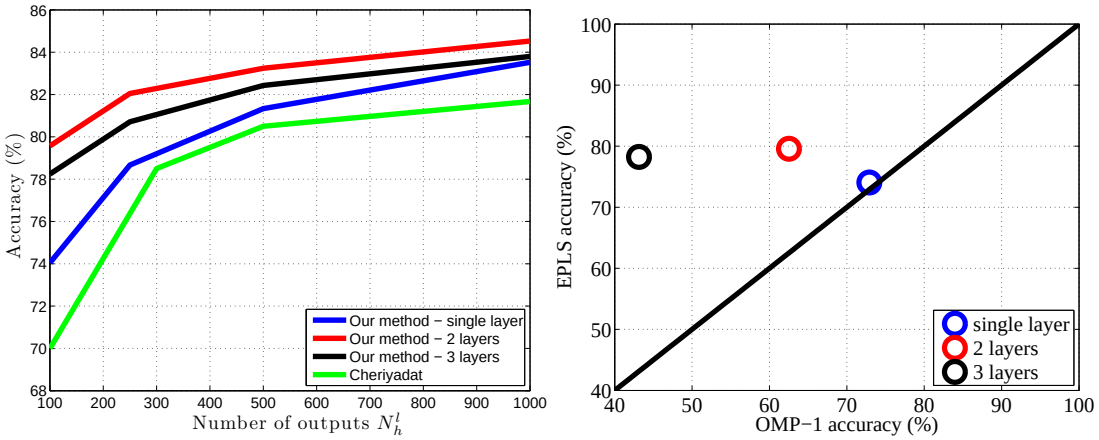


Figure 4.2: Left: UCMerced validation accuracy given different number of features and different network architectures (best seen in color). Right: Comparison of EPLS training against OMP-1 for $\mathrm{N}_\mathrm{h}^l = 100$ on architectures of increasing depth (best seen in color).

(right) our approach to previously reported results [90, 92]. The producers' accuracy is in general favorable for our method: in 14 out of 21 classes, we obtained better results than in [92], and in 15 out of 21 in [90]. These results encourage the use of the method, and the exploitation of combined approaches in future research.

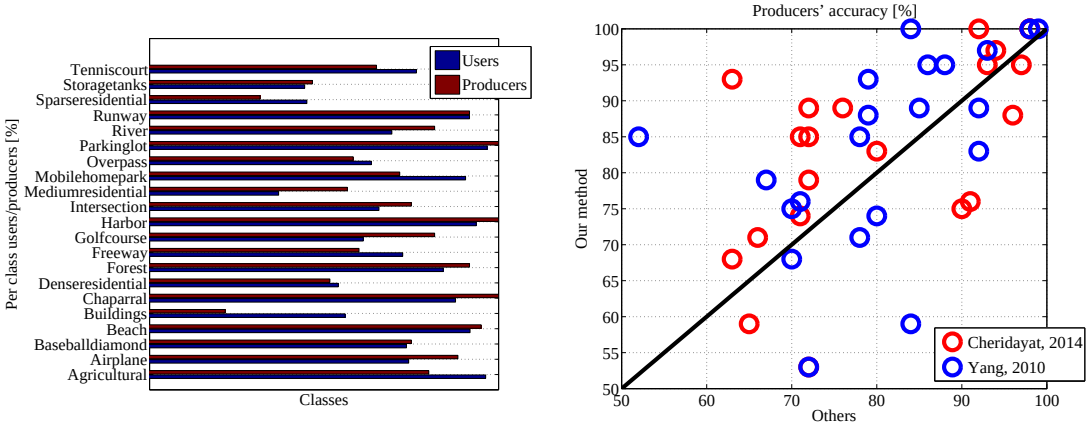


Figure 4.3: UCMerced per class users and producers results. Left: per class users and producers of the proposed method. Right: comparison of our method to previous works in the literature in terms of producers (per class accuracy) [90, 92]. We took the results for the best algorithms in both [92] (their Figure 8, 'color' model) and [90] (his Fig. 13, dense SIFT descriptors) (best seen in color).

In the second setting, we analyze deep architectures to further exploit the possibilities of our

method. In this case, we use a receptive field of 7×7 pixels, with a stride of 1 pixel and train deep architectures composed of two and three layers, respectively. We apply a max-pooling layer on non-overlapping regions of size 2×2 pixels after each convolutional layer, except for the last max-pooling layer, which divides the activation feature maps into four quadrants and applies sum-pooling for fair comparison with single layer architectures. We train each layer by means of EPLS with a logistic non-linearity using 100K patches per layer. After training, we apply a linear encoding (i.e. identity activation function) to each hidden representation layer and a rectifier encoding with polarity split to the last representation layer. As in the first experimental setting, we train deep architectures with varying number of features $\mathrm{N}_{\mathrm{h}}^{l}$. For simplicity, all layers within a deep architecture have the same number of features. Figure 4.2 (left) shows the classification performance of both the 2-layer (solid blue line) and 3-layer architectures (solid black line), for different $\mathrm{N}_{\mathrm{h}}^{l}$ values. As shown in the figure, 2-layer networks improve the previous single layer results, for all $\mathrm{N}_{\mathrm{h}}^{l}$. The 2-layer network with $\mathrm{N}_{\mathrm{h}}^{l} = 500 \; \forall l \in \{1, 2\}$ outperforms the single layer network with $\mathrm{N}_{\mathrm{h}}^{l} = 1000$, whereas the 2-layer network with $\mathrm{N}_{\mathrm{h}}^{l} = 1000 \; \forall l \in \{1, 2\}$ outperforms all previous results. However, when increasing the number of layers to 3, the accuracy starts dropping. We argue whether the UCMerced dataset could benefit from a higher level of abstraction in its feature representation, given the (high) amount of texture present in its images. Even if it could benefit from the higher abstraction, the layer-wise pre-training might be too greedy and a fine-tuning step might be required to achieve better performance as we increase the depth of the network. Furthermore, the receptive field size could be appropriately tuned to improve the results; note that the image region considered in the 3rd layer is much bigger than the ones considered by the 1st and 2nd layers. Finally, it is worth noticing that as we increase the number of layers, the number of parameters increases dramatically and the model becomes more prone to overfit.

Finally, after highlighting the impact of stacking hierarchical (deep) representations, we would like to assess the importance of sparsity to achieve good hierarchical representations. In order to highlight the relevance of lifetime sparsity in deep scenarios, we use OMP-1 as a substitute of EPLS to reproduce the experiments in Figure 4.2 (left) for $\mathrm{N}_{\mathrm{h}}^{l} = 100$. Note that OMP-1 does not promote any kind of lifetime sparsity but enforces a very strict form a population sparsity instead (see Chapter 2 for details). Figure 4.2 (right) reports the obtained results. In the case of the single layer network (red circle), EPLS achieves slightly better results than OMP-1. However, as shown in the figure, OMP-1 seems not to be able to take advantage of depth. When adding a second layer to the architecture (blue circle), OMP-1 experiences

a performance drop of 10.38%, whereas ELPS improves its performance by 5.52%. When adding a third layer to the architecture (green circle), both OMP-1 and EPLS decrease their performances. OMP-1's performance drop is particularly dramatic (from 72.90% in the single layer architecture to 43.14% in the 3-layer architecture). We argue this dramatic performance drop is related to the OMP-1's lack of lifetime sparsity, which makes the algorithm suffer from dead features. While increasing the network's depth, the dead features' effect becomes more significant and impacts the performance of the method. Therefore, enforcing lifetime sparsity is crucial for EPLS to achieve good performance.

4.1.3.2 Hyper-spectral Image Classification

This section illustrates the performance of the feature learning pipeline on a challenging hyper-spectral pixel classification problem: the well-known AVIRIS Indiana's Indian Pines test site acquired in June 1992[3]. A subset of the original image (145×145 pixels) has been extensively used as a benchmark for comparing classifiers[4]. However, we consider the whole image, which consists of 614×2166 pixels and 220 spectral bands, with a moderate spatial resolution of 20 m. This dataset represents a very ambitious land cover classification scenario.

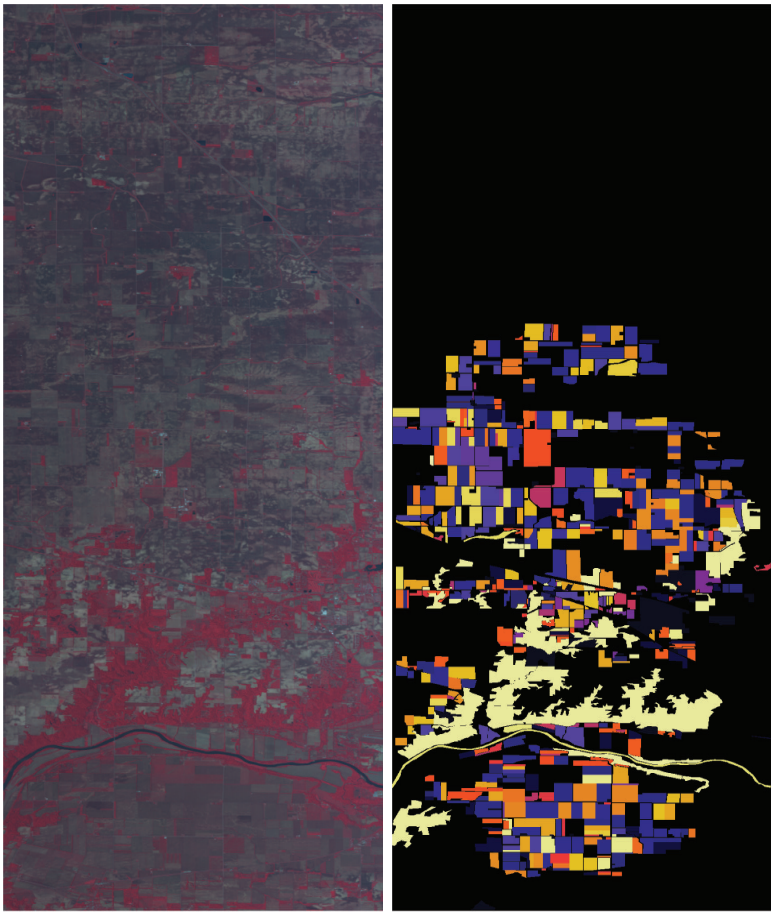
The image presents some crops at early stages of growth, covering a very small portion of the image. Therefore, from the 58 different land cover classes available in the original ground truth, we discard 20 classes given the insufficient number of training samples available[5].The background pixels are not considered for classification. As is standard practice, we remove several bands due to noise and water absorption phenomena, finally working with 200 spectral bands. Figure 4.4 shows a RGB composite of the AVIRIS Indian Pines image (left) along with its labeled ground truth (right).

In this experiment, we aim to analyze the influence of various hyper-parameters, in order to better understand the potential of deep architectures. To do so, we design different architectures. For each deep architecture, we choose different number of layers ($L \in \{1 \cdots 7\}$) to verify the benefits of having increasing levels of abstraction and different receptive field sizes (1×1, 3×3 or 5×5), using the same receptive field for all layers, to study the relevance of

[3]We would like to thank Prof. Antonio Plaza from the University of Extremadura, Spain, for kindly providing the AVIRIS dataset.

[4]`ftp://ftp.ecn.purdue.edu/biehl/MultiSpec/92AV3C.lan`

[5]less than 1000 samples



RGB Composition Ground truth

Figure 4.4: Color composition (left) and the available reference data (right) for the hyper-spectral AVIRIS Indian Pines dataset.

spatial information. Moreover, we build architectures both with and without pooling layers to assess the effect of the downscaling factor.

We train all deep architectures as shown in Figure 4.1. We employ the logistic non-linearity during training and, after that, we apply a natural encoding without polarity split to extract the hierarchical features. Finally, we compare the extracted features in terms of performance on the validation set and robustness to the number of labeled examples to the ones extracted by PCA and kPCA, which are the standard dimensionality reduction techniques employed in remote sensing. More precisely, we extract different numbers of features $\mathrm{N_f} = \{5, 10, 20, 50, 100, 200\}$ by means of PCA and kPCA and design networks with the

same number of features per layer $\mathrm{N}_\mathrm{h}^l = \{5, 10, 20, 50, 100, 200\}\ \forall l$. For simplicity, all layers have the same number of features. We evaluate the feature extraction in all scenarios varying the rates of training samples per class, $\{1\%, 5\%, 10\%, 20\%, 30\%, 50\%\}$. Then, we feed the extracted features to a 1-NN classifier with Euclidean distance in order to measure the performance of each system by means of Cohen's kappa statistic, κ, in an independent validation set made of the remaining samples.

For kPCA, we use an RBF kernel defined as

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \frac{1}{(2\pi\sigma^2)^{\mathrm{N}_\mathrm{h}^0/2}} e^{-\left(||\mathbf{x}^{(i)}-\mathbf{x}^{(j)}||_2^2\right)/(2\sigma^2)}, \tag{4.1}$$

where $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ are two input samples of dimensionality N_h^0 and σ is the scale parameter. We set σ to the average distance between all training samples as a reasonable estimate; note that the feature extraction is unsupervised, so there are no labels at this stage to tune kernel parameter.

Figure 4.5(a) shows the κ statistic for several numbers of extracted features using PCA, kPCA and single layer networks (N_f and N_h^1, respectively). Both kPCA and the networks yield poor results when a low number of features are extracted, and drastically improve their performance for more than 50 features. Single layer networks stick around $\kappa = 0.3$ for pixel classification, even with increased number of features. Nevertheless, there is a relevant gain when spatial information is considered. The best results are obtained for $\mathrm{N}_\mathrm{h}^1 = 200$ features and 5×5 receptive fields. With these encouraging results, we train deeper networks using 30% of the available training samples per class, $\mathrm{N}_\mathrm{h}^l = 200$ features per layer, and receptive fields of 3×3 at each layer. Results with and without the max-pooling layers are shown in Figure 4.5(b). Two main observations can be made: first, deeper networks improve the performance enormously (the 6-layer network reaches the highest performance with $\kappa = 0.84$), and second, including the max-pooling layers after each convolutional layer reveals to be extremely beneficial.

Another question to be addressed is the robustness of the features in terms of number of training samples. Figure 4.5(c) highlights that using a few supervised samples to train a deep architectures can provide better results than using far more supervised samples to train a single layer one (see Figure 4.5(a)). Note, for instance, that the 6-layer network using 5% samples/class outperforms the best single layer network using 30% of the samples/class.

Special attention should be devoted to the 7-layer network. In this case, the performance decreases since the potential contribution of an additional layer is strongly counterbalanced by the heavily reduced spatial resolution of the additional max-pooling. To corroborate this

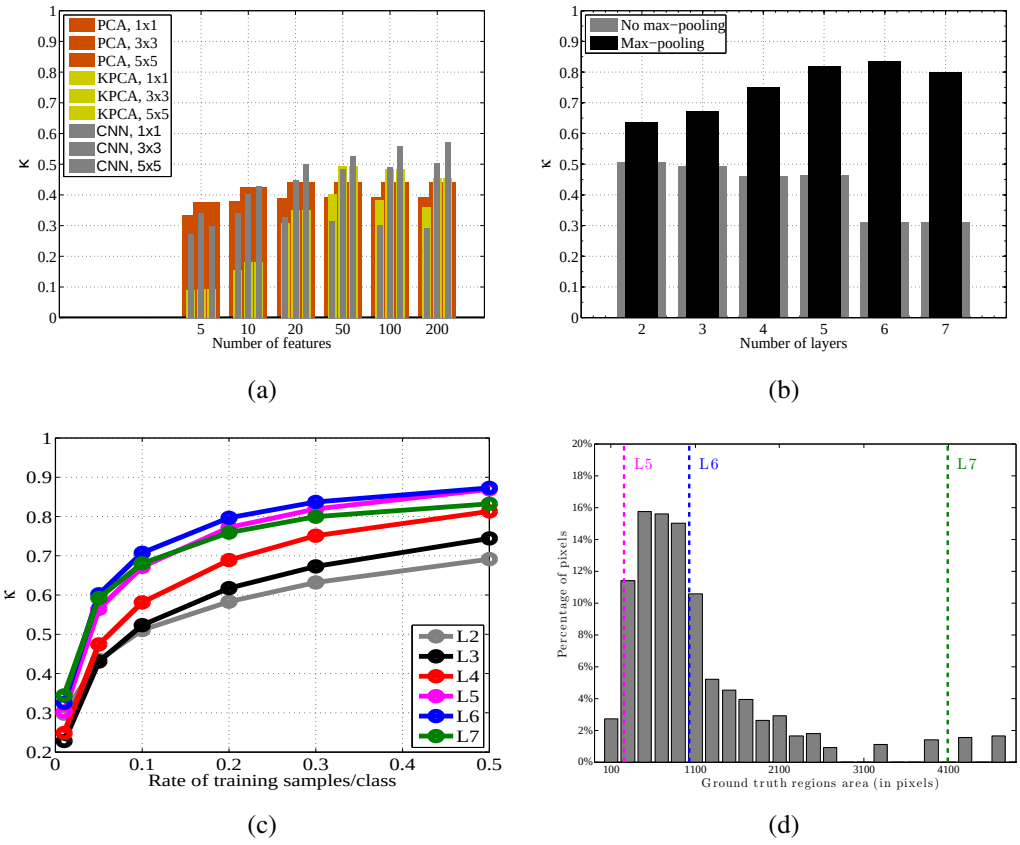


Figure 4.5: AVIRIS pixel classification performance estimated by means of Cohen's kappa statistic, κ, on the validation set for (a) increasing numbers of features, different size of the receptive fields (for single layer networks) or the included Gaussian filtered features (for PCA and kPCA) using 30% of the training data; (b) deep architectures with different number of convolutional layers, either followed or not by pooling layers; (c) for architectures (containing pooling layers) trained on different rates of training samples per class, $\{1\%, 5\%, 10\%, 20\%, 30\%, 50\%\}$. The last figure (d) shows the percentage of ground truth pixels as a function of labeled region areas (see text for details).

explanation, we created the histogram in Figure 4.5(d), which shows the percentage of ground truth pixels as a function of labeled region areas. As it can also be seen in Figure 4.4(right), the labeled regions are mainly rectangular with an average area around 500 pixels. Vertical lines in Figure 4.5(d) show the theoretical spatial resolution in the case the last representation layer is resized using a nearest neighbor interpolation. As it can be observed, when using 7 layers (L7, green), the resolution is too low to capture regions smaller than 4096 pixels (64×64). It has to be noted that we perform the upscaling of the last representation layer by means of a bilinear

interpolation; this explains why, despite the lower spatial resolution, the result using 6 layers is still superior to the one with 5 layers.

An important aspect of the proposed deep architectures lies in the fact that they typically give rise to compact hierarchical representations. The best three features extracted by the networks according to the mutual information with labels are depicted in Figure 4.6 for a subset of the whole image. It is worth stressing that deeper layers retrieve more complicated and abstract features, except for the seventh layer that provides spatially over-regularized features due to the downscaling impact of the max-pooling layers. Interestingly, it is also observed that, features in the bottom layers present high levels of redundancy (they look more alike) and, the deeper we go, the higher spatial decorrelation of the best features we obtain.

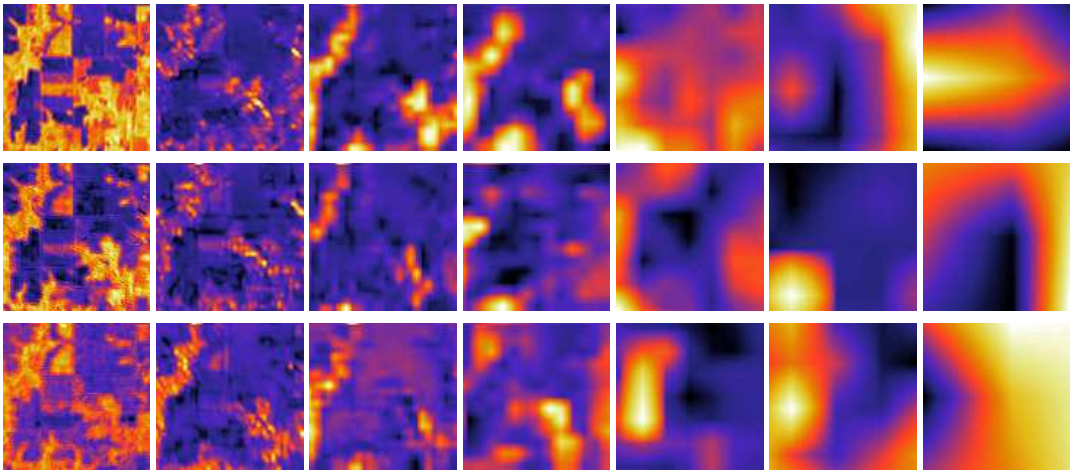


Figure 4.6: Best three AVIRIS features (in rows) according to the mutual information computed between the classification labels and the features extracted by the different layers (1st to 7th in columns) for a subregion of the whole image.

4.1.3.3 Very High Resolution Image Pixel Classification

This section studies the application of the proposed method to segment very high resolution images (VHR). Very high spatial resolution has been one of the major achievements of satellite imagery of the last decades. Sensors providing sub-metric resolution have been developed and satellites such as QuickBird, GeoEye-1 or WorldView-3 have been or are about to be launched. These sensors provide images that are unique in terms of spatial detail and open a wide range of challenges for geospatial information processing.

We validate our framework on two VHR images gathered by Quickbird II, an imaging satellite that employs a four-band sensor with 2.4-m spatial resolution for blue, green, red and near-infrared spectral wavelengths in addition to a 0.6-m resolution panchromatic (black and white) band (©2008 Digitalglobe, all rights reserved). Images were taken in late summer or early fall periods, coinciding with seasonal base flow conditions and field surveys. Each image was geo-referenced using field-collected ground control points that yielded an average root mean square geo-location error of less than 1.5 m. The scenes were acquired over Nayak-Middle Fork ($1659 \times 1331 \times 4$) of the Flathead River in the Nyack flood plain bordering Glacier National Park, Montana, and the Kol flood plain ($1617 \times 1660 \times 4$), during 2008[6]. Both images have been widely used to study and characterize the physical complexity of North Pacific Rim rivers to assist wild salmon conservation. Therefore, the labeled land use classes of these scenes correspond to shallow shore, parafluvial and orthofluvial salmon habitat types [132, 133]. Figure 4.7 shows the RGB composite of the two VHR images used for pixel classification.

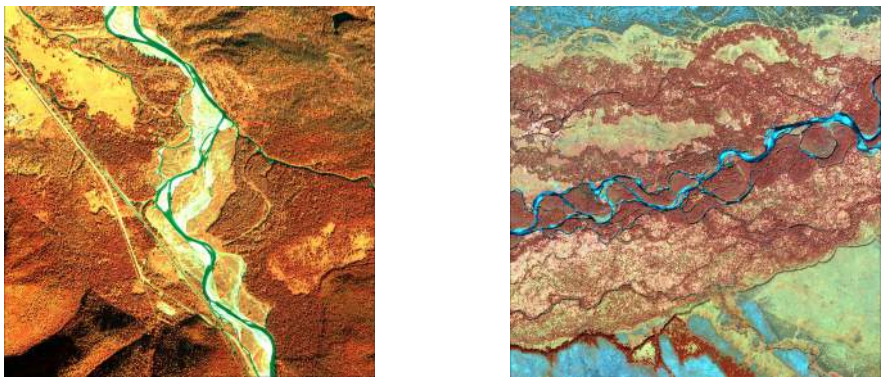


Figure 4.7: RGB composition of the two VHR images considered for pixel classification: 'Nayak - Middle Fork' (left) and 'Kol' (right).

The experimental setting involves an independent feature extraction and pixel classification per image, which is a standard scenario in remote sensing pixel classification. The aim of the experiment is to assess the capabilities of sparse hierarchical representations to capture spectral-spatial structure for habitat classification. To do so, we design models of varying depth and fix the number of features per layer to $\mathrm{N}_{\mathrm{h}}^{l} = 200 \; \forall l$. For single layer architectures, we use a receptive field of 5×5 pixels, while for deep architectures we use a receptive field of

[6]We would like to thank Prof. Diane Whited at the University of Montana for providing the VHR imagery.

3×3 pixels. We apply a max-pooling layer on non-overlapping regions of size 2×2 pixels after each convolutional layer, except for the last convolutional layer.

We train the deep architectures following the framework depicted in Figure 4.1. We employ the logistic non-linearity during training and, after that, we apply a natural encoding without polarity split to extract the hierarchical features. Finally, we feed the features of the last representation to a 1-NN classifier with Euclidean distance. As in experiment 4.1.3.2, validation results are compared to PCA and kPCA followed by 1-NN classifier in terms of performance and robustness to the number of training samples.

Figure 4.8 shows the pixel classification results in terms of overall accuracy and κ statistic on the validation sets of the two VHR images, as a function of the number of training samples per class ($\{0.5\%, 1\%, 2.5\%, 5\%, 10\%\}$) and layers (1–6). Three main observations are made from the experiments: 1) results are improved when increasing the number of training samples; 2) non-linear feature extractors (both kPCA and the deep representations) outperform the linear PCA; and 3) deeper networks tend to achieve better performance. We observe an average gain with a deep architecture of about +10% for the Nayak image and of +20% for the Kol image in terms of κ statistic. Again, results saturate for 6 layers and for more than 5% of training samples per class (this issue has been also observed in the hyper-spectral image classification problem in section 4.1.3.2).

4.1.3.4 Multispectral Image Classification

This section tackles the challenging problem of cloud screening using multispectral images. We validate our framework on seven images taken by the Medium Resolution Imaging Spectrometer (MERIS) instrument on-board the Environmental Satellite (ENVISAT). In particular, we use images acquired over Abracos (2004), Ascension Island (2005), Azores (2004), Barcelona (2006), Capo Verde (2005), Longyearbyen (2006) and Mongu (2003). These images not only cover different geographic locations but were taken in different seasons and, therefore, present different types of clouds and surfaces. Moreover, the scenes include different landscapes; soils covered by vegetation or bare; or even ice and snow. All images consist of 321×490 pixels and have 16 channels. In this case, we consider all 16 channels for the feature extraction. The leftmost column of Figure 4.10 shows a RGB composite of the seven cloud screening images used in this experiment.

The proposed experimental setup aims to assess the expressive power and robustness of sparse hierarchical features. To do so, we design a deep architecture, where each layer has

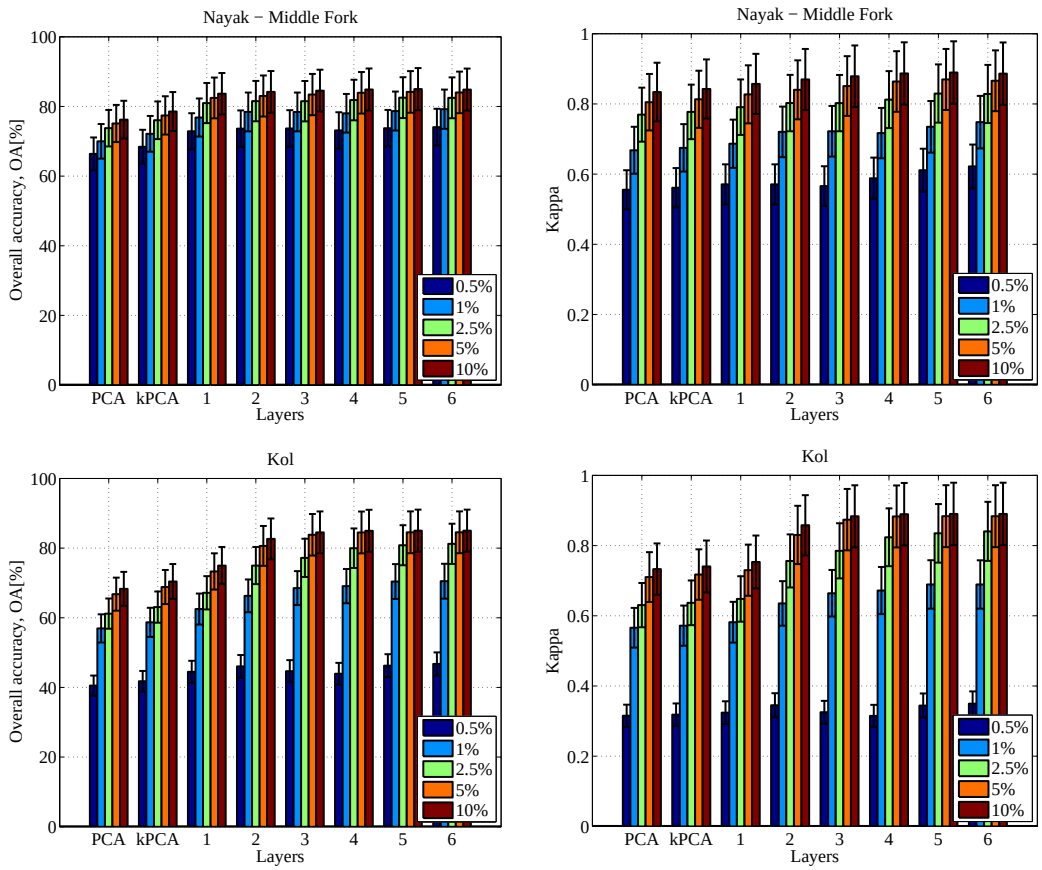


Figure 4.8: VHR classification results (overall accuracy and κ statistic on the validation set) in the form of the average $\pm$ standard deviation bars over 10 realizations of the pixel classification experiment in two VHR images, as a function of the number of training samples and the architecture depth.

the same number of features. We use the same receptive field size (3×3 pixels) for each layer and apply a max-pooling layer on non-overlapping regions of size 2×2 pixels after each convolutional layer, except for the last convolutional layer. We train the deep architecture as shown in Figure 4.1 by means of EPLS with logistic non-linearity[7]. Then, we apply a natural encoding without polarity split to extract the hierarchical features. Finally, the features of the last representation layer are fed to a 1-NN classifier with Euclidean distance. In this case, we follow an image-fold cross-validation scheme, i.e. we train on six out of seven images and validate on the remaining one.

[7]We also tried with a linear activation function but the extracted features were less performing.

As in experiments 4.1.3.2 and 4.1.3.3, validation results are compared to PCA and kPCA followed by 1-NN classifier in terms of performance and robustness to the number of training samples. For the sake of a fair comparison, we extract a fixed number of features in all cases. We set a number of features $\mathrm{N_f} = 120$ for both PCA and kPCA and use $\mathrm{N_h^l} = 120\ \forall l$ for the deep architecture (in this experiment, $L = 2$ exhibited the best performance). Note that the cloud screening images shown in the leftmost column of Figure 4.10 contain more texture than structure, and the details are usually very smooth. We validate our method both quantitatively and qualitatively.

Figure 4.9 shows the pixel classification results in terms of overall accuracy and κ statistic over the seven multispectral images, as a function of the number of training samples. Results show that both measures are consistent and follow the same trends. As it can be observed, the proposed deep architecture generally outperforms the other feature extractors, especially when few training samples are used for pixel classification. This suggests that the extracted features are potentially richer and more discriminative, which can be compensated with the information conveyed by using more labeled samples. Note that tuning the kernel parameter for kPCA did not yield better results.

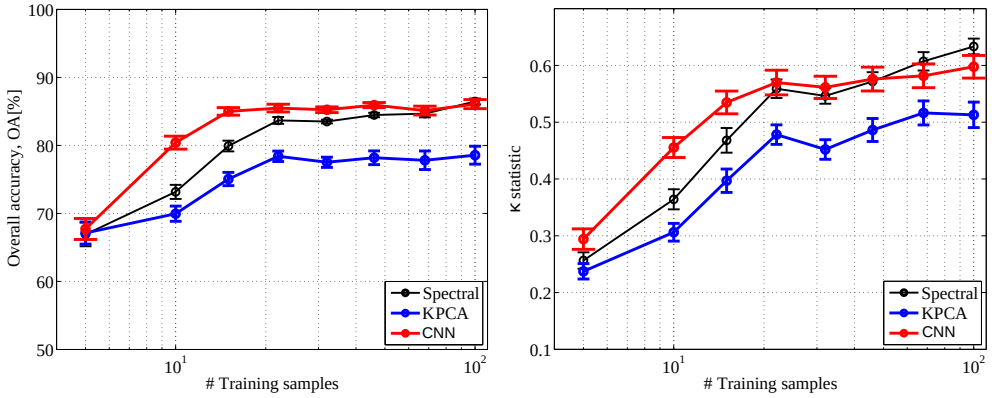


Figure 4.9: Classification results for the multispectral MERIS images (mean and standard deviation) over the seven multispectral images, as a function of the number of training samples.

Figure 4.10 shows the different MERIS images (1st column) along with the classification maps obtained with a 1-NN classifier on top of the raw spectral information (2nd leftmost column), the features extracted by kPCA (2nd rightmost column) and the ones extracted by the deep architecture (rightmost column). In terms of κ statistic, the deep architecture (NNET on the figure) demonstrates a significant gain over the other approaches in most of the cases.

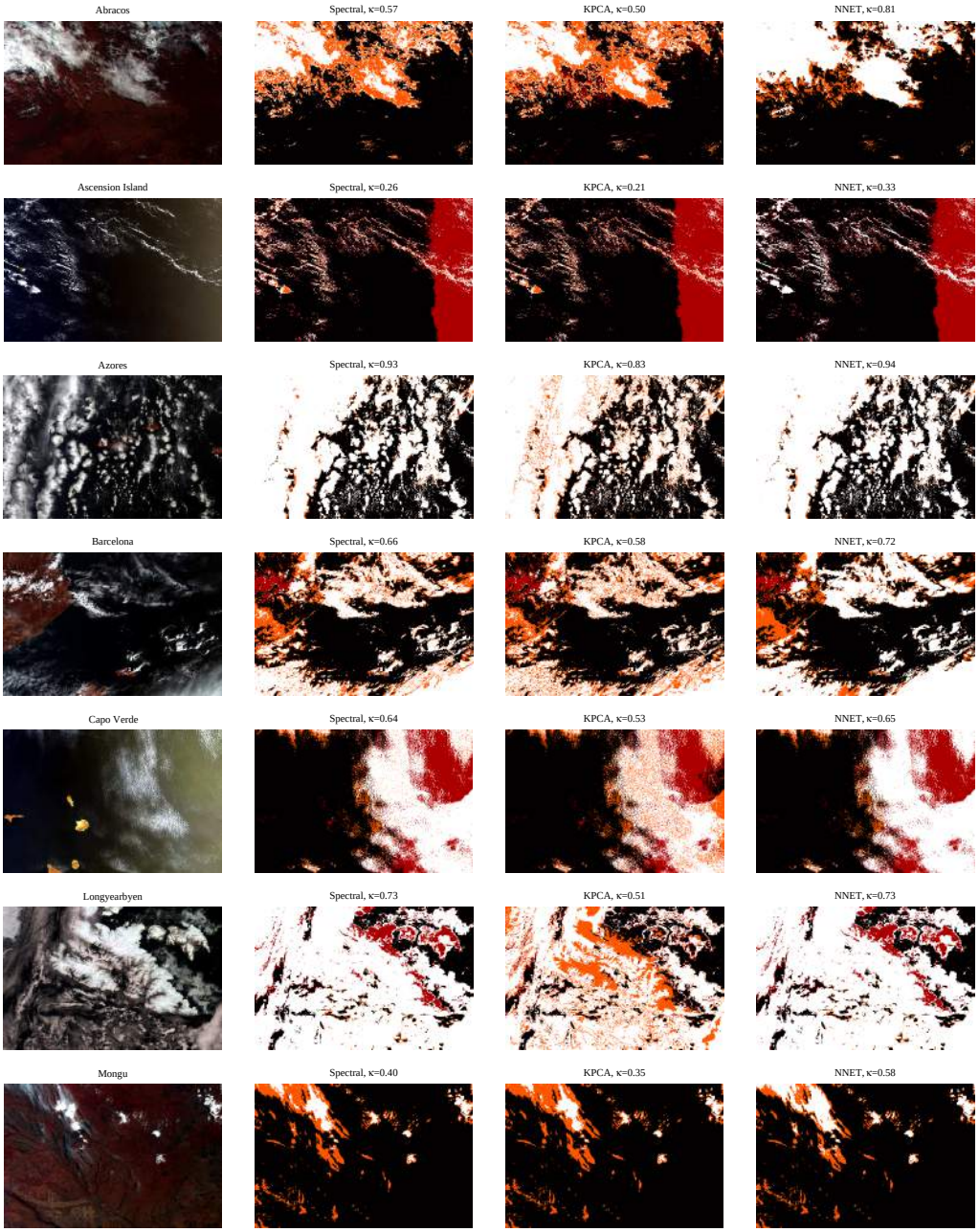


Figure 4.10: Classification maps for the different multispectral MERIS images (1st column) obtained by a 1-NN classifier on top of raw spectral information (2nd column), the features extracted by kPCA (3rd column) and our trained deep architecture (4th column). The obtained κ statistic is shown on top of the maps.

Nevertheless, in some particular cases, such as Azores, Capo Verde and Longyearbyen, the deep architecture's improvement is only apparent when compared to kPCA. This might be explained by a potentially low efficiency in extracting spatially relevant features from areas highly affected by clouds over snowy mountains (as in Longyearbyen) or by sunglint (as in the east part of Capo Verde). Additionally, the hierarchical features extracted by the deep architecture may not have an added value when dealing with very easy scenes, such as compact clouds over sea (as in Azores).

It is worth noting that in other cases with sunglint conditions (as Ascension island), the gain obtained by the proposed deep architecture is noticeable (+7% w.r.t raw spectral features and +12% w.r.t. kPCA), especially because of the high rate of positive detections in the part of the scene not affected by the sunglint. Another interesting case of study is the image from Barcelona. Although all methods obtain visually similar maps, the deep architecture yields a lower false alarm rate in compact structures (southern and northern big clouds), demonstrating that the spatial-spectral information has been well captured. Note that this property of the method is highlighted in the Abracos and Mongu scenes. These scenes strongly benefit from a reduced false alarm rate in clouds over flat landscapes, attaining a remarkable performance improvement of +31% and +23% w.r.t. kPCA features and an improvement of 24% and 8% w.r.t. raw spectral information.

4.1.4 Summary

In this section, we exploited the properties of EPLS to train deep architectures in a greedy layer-wise fashion [15, 16]. We applied the proposed framework to learn unsupervised features from a wide variety of remote sensing images of different spatial and spectral resolutions, from multi- and hyper-spectral images, to very high resolution problems. We provided and in-depth analysis on the impact of depth, sparsity, number of features, pooling, receptive field size and number of labeled training samples required on the performance of the system.

Results revealed that the trained deep architectures are very effective at encoding spatio-spectral information of the images. Experiments showed that (1) lifetime sparsity is indeed important to achieve good hierarchical representations; (2) depth is crucial to improve the performance of a system, as long as the spatial resolution of the features is not excessively reduced; (3) including spatial information is essential in order to avoid poor performance in single layer networks; and (4) max-pooling layers are crucial to achieve good performance.

Furthermore, hierarchical features extracted by deep architectures proved to be more robust to low numbers of labeled training samples.

4.2 Application to Image Parsing

After analyzing the potential of EPLS to learn hierarchical features in Section 4.1, we aim to apply the algorithm to the *image parsing* problem. *Image parsing* (also known as scene parsing, scene labeling, semantic segmentation or pixel classification) decomposes an image into its visual components, i.e. it seeks a semantically meaningful label for each pixel. The main challenge of image parsing is to capture basic underlying structure of the image that not only models the *global* and *local* appearance and geometry of the image but also exploits the spatial interaction between labels. *Global features* provide information on the whole image; *local features* describe a given receptive field (even a large one); whereas *semantic features* encode the spatial relations between classes.

As in many other computer vision problems, convolutional deep architectures have shown to extract effective hierarchical features for image parsing, reaching state-of-the-art performance on several datasets and being faster than previous methods at inference [134, 135, 136]. Deep architectures allow to capture local features with larger spatial dependencies by successively applying convolutional layers. However, they do not extract any global nor semantic feature from the images, which has shown to improve image parsing performance [137]. In order to include semantic information, approaches such as [135] add a post-processing step based on conditional random fields or optimal purity cover criteria. This post-processing turns out to be crucial to achieve state-of-the-art performance. Other approaches attempt to endow the system with the necessary structure to learn semantic features. This is the case of [136], which adds a top-down semantic feedback by means of a recurrent version of convolutional networks. The training of all these (modified) architectures relies on back-propagation (or back-propagation through time), which is a slow learning technique and, in the case of [136] is the main reason to limit the number of recursions.

In this section, we propose to train convolutional deep architectures in a greedy layer-wise fashion (see Section 2.2.2.1) by means of EPLS unsupervised learning algorithm [21] introduced in Chapter 3. We incorporate top-down semantic feedback from the output of the softmax layer of the network, which computes the class posterior probabilities of the input samples. Moreover, we include global and semantic features in a simple and intuitive way, with-

out substantially increasing the training and inference computational burden. The approach is characterized by an efficient training and a sufficiently fast inference. Experimental results outline the advantages of he proposed architecture. Table 4.1 summarizes the main properties of state-of-the-art convolutional deep networks compared to our proposal. Note that the approach presented in this section incorporates global/local appearance/semantic features as well as top-down feedback and has significantly faster training than the others.

Table 4.1: Characteristics of our image parsing method compared to convolutional-based state-of-the-art image parsing approaches. Training and testing times are excerpted from respective papers for the configurations that gave the best results as reported in Table 4.2. Training time for the method in [136] has been provided by the author in a personal communication. We present the testing speed in kilo pixels per second, to make it independent on the image size.

Method	Local features		Global features		Top-down	Training	Testing
	Appearance	Semantic	Appearance	Semantic	feedback	Time	Speed
[135]	Learned	No	No	No	No	2∼5d	1.3 kpx/s
[136]	Learned	Learned	No	No	Yes	$>$ 1w	8.9 kpx/s
Ours	Learned	Learned	Learned	Learned	Yes	3h50'	3.7 kpx/s

4.2.1 Image Parsing Architecture

The system we design to learn local and global appearance and semantic features for image parsing is based on a convolutional deep network. The network is trained in a greedy layer-wise fashion [15, 16] by means of EPLS [21]. Features extracted from all layers are combined, to account for different levels of abstraction, and fed to a softmax classifier as in [138].

Figure 4.11(a) shows how the proposed image parsing architecture is built. An input image $\mathbf{H}^0$ is fed to the convolutional deep architecture and propagated from the first to the last representation layer. Feature maps $\mathbf{H}^l$ extracted from each convolutional layer take two different paths: (1) they are propagated through a max-pooling layer to the next convolutional layer; and (2) they are upscaled, if necessary, to match the size of the input image $\mathbf{H}^0$ by means of a bicubic interpolation. After that, the upscaled feature maps are concatenated into one vector. Optionally, a feature vector providing information on prior spatial classes distribution ($\mathbf{O}$) is added. The complete feature vector is then fed to a softmax classifier, which provides a vector P of class posterior probabilities for all image pixels. During the training phase, the softmax classifier receives the ground truth (GT) information as input as well.

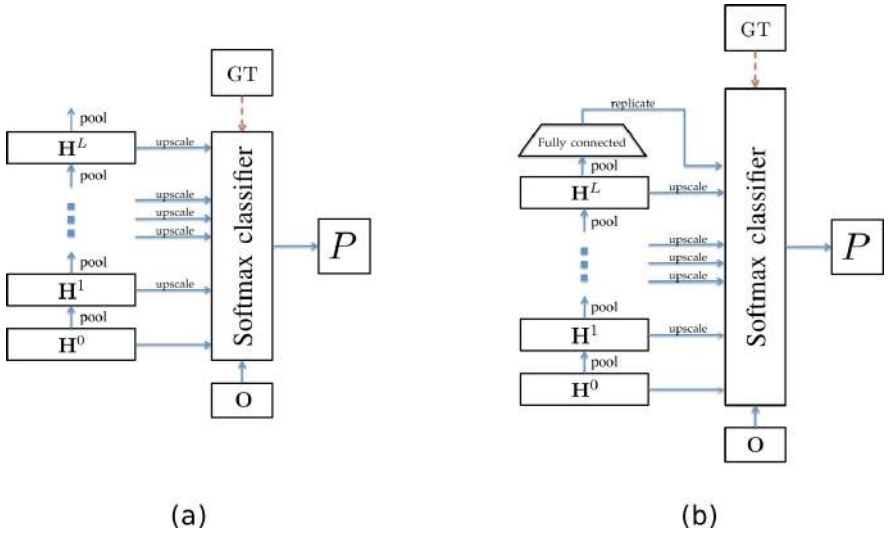


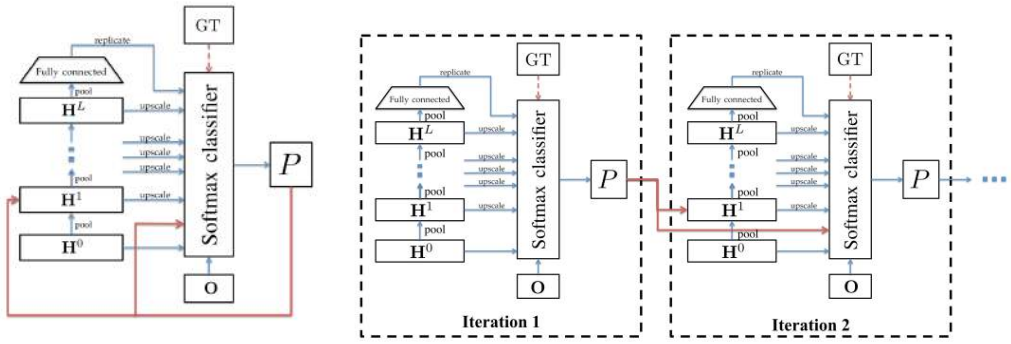
Figure 4.11: Image parsing: (a) The basic deep architecture coupled with a softmax classifier. (b) The addition of a top fully-connected layer, which provides a compact global appearance descriptor.

Note that the input data corresponding to the photometric RGB image is normalized for contrast and brightness. Moreover, all layers' input data is standardized to accelerate the gradient descent convergence during training.

4.2.2 Top Fully-connected Layer: a Global Appearance Descriptor

The architecture presented in Section 4.2.1 applies successive convolutional layers, followed by pooling layers and leads to an output representation of spatial size $\mathrm{R}^{out} \times \mathrm{C}^{out}$. Hence, the top-most representation is a 3-D tensor of size $\mathrm{R}^{out} \times \mathrm{C}^{out} \times \mathrm{N}_{\mathrm{h}}^{out}$, summarizing the content of the whole input image. This representation might still be too high-dimensional, especially if $\mathrm{N}_{\mathrm{h}}^{out}$ is large, and possible correlations between the representation spatial support $\mathrm{R}^{out} \times \mathrm{C}^{out}$ are not explicitly encoded.

Therefore, we propose to add a top fully-connected unsupervised layer that maps the $\mathrm{R}^{out} \times \mathrm{C}^{out} \times \mathrm{N}_{\mathrm{h}}^{out}$ elements into a smaller feature representation (see Figure 4.11(b)) that captures correlations between all the features $\mathrm{N}_{\mathrm{h}}^{out}$ and their spatial support $\mathrm{R}^{out} \times \mathrm{C}^{out}$. The resulting descriptor encodes global information of the whole image. To feed this information to the softmax classifier, the feature vector must be replicated for all image pixels. The behavior and the contribution of this additional layer to the deep architecture's performance is quantitatively and qualitatively evaluated in Section 4.2.4.1.



(a) (b)

Figure 4.12: Image parsing semantic information: (a) Schema introducing the loopy top-down semantic feedback. (b) The unrolled architecture for a two iterations case.

4.2.3 Unrolling Loopy Top-down Semantic Feedback

Up to now, we have built a deep architecture, which is able to capture global information of the input image. Yet the semantic information is still missing. Figure 4.12(a) shows a possible way to introduce top-down semantic feedback in the proposed architecture. Since we use the output of all layers as features, the unfolding approach proposed in [136] cannot be employed in a straightforward way. Instead, we can replicate the architecture as many times as we want and feed all the deep networks (except the first one) with the posterior probability generated by the previous softmax classifier. Figure 4.12(b) shows our approach for the two iterations case. The parameters of different deep architectures and classifiers cannot be shared since the deep architectures are trained in an unsupervised way and their input data depends on the output of previous classifier. While this seems a disadvantage with respect to [136], it in fact allows to train the whole system without the need of an expensive training algorithm as the back-propagation through time (BPTT) used in [136]. The BPTT algorithm is the main reason to limit the number of recurrencies to 3 in [136]. Another advantage of our method is that subsequent deep architectures can learn different features depending on the input data, thus being able to blend information from RGB data and posterior probability in an implicit way.

4.2.4 Experimental Results

We test our method on the SIFTflow dataset [139], which is composed of 2688 images and presents 33 different categories. We use the standard train/test split of [139]. As in [136], we

re-scale the input image by 1/2 to speed-up both training and testing phases. Nonetheless, for a fair comparison to other methods, the evaluation is performed by upscaling by a factor 2 the posterior probability P before comparing the maximum a posteriori labeling with the ground truth.

The basic architecture is composed of 6 convolutional layers with receptive field of 3×3 pixels, followed by spatial max-pooling operations of non-overlapping 2×2 pixel regions. The size of the pooling region is set to its minimum possible value, to allow for deeper architectures. Similarly, the receptive field is set to the minimal symmetric size, so to minimize the computational cost of convolutions and to delegate, whenever possible, the learning of complex spatial configuration to higher layers. Each convolutional layer has $\mathrm{N}_{\mathrm{h}}^{l} = 100$ features. The spatial prior $\mathbf{O}$ is computed by accumulating the occurrences of each class in 33 separate maps (at full resolution) for all the training images; then the resulting maps are normalized and blurred with a Gaussian filter with $\sigma = 32$ pixels. When employed, the top fully connected layer has 33 units such that (1) we match the spatial prior size, for a fair comparison in Section 4.2.4.1; and (2) when using the unrolled architecture the total number of features is 3 (RGB input) + 6×100 (6 representation layers) + 33 (spatial prior, $\mathbf{O}$) + 33 (fully connected top layer) + 33 (posterior of previous iteration, P) = 702. This is specifically designed to have less features than [135] (768 features).

The training of the above-described architecture has both an unsupervised and a supervised learning stages. The unsupervised learning stage uses 50K random samples to train the convolutional filters of each layer and the top fully-connected layer. The layers are trained greedily one after another by means of EPLS. The supervised learning stage uses 1% of the labeled data per iteration. The regularization term in the softmax classifier is set to $\lambda = 10^{-3}$ in all experiments. Softmax parameters are learned using the LM-BFGS optimizer for a maximum of 500 iterations.

Following the above-described pipeline, we define two experiments to assess the benefits of the proposed method. First, in Section 4.2.4.1, we will highlight the advantage of using a fully-connected unsupervised layer as global descriptor. Second, in Section 4.2.4.2, we will discuss the advantage of adding the unrolled top-down semantic feedback by means of a quantitative and qualitative analysis of the results.

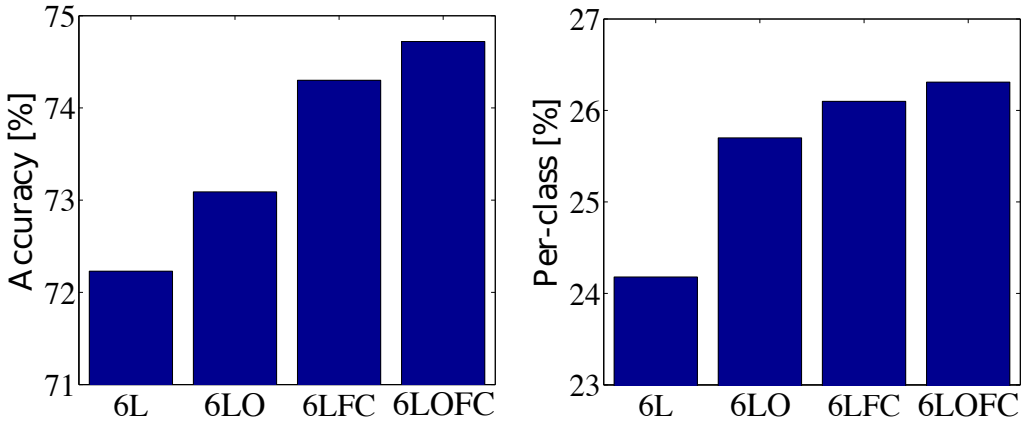


Figure 4.13: Quantitative comparison of 4 configurations with and without the top fully-connected layer (global descriptor). Global accuracy (left) and per class average accuracy (right) on the SIFTflow dataset. 6L refers to the basic 6-layer convolutional architecture; 6LO adds the spatial prior to the basic architecture; 6LFC adds the fully-connected layer instead; 6LOFC adds both the spatial prior and the fully-connected layer to the basic architecture. See text for details.

4.2.4.1 Unsupervised Global Image Descriptor

In this first set experiment, we analyze the effect of the top fully-connected layer on the system's performance. Since the softmax classifier is fed with the features extracted by all convolutional layers, the spatial prior and the output of the fully-connected layer, we separate these three components to evaluate their contribution. Figure 4.13 shows the results in terms of global accuracy (left) and average per class accuracy (right) for four configurations. The first configuration is the basic 6-layer convolutional architecture (6L) using a total of 603 (3 + 100×6) features. The second one adds the spatial prior as an additional feature (6LO, 603 + 33 features). The third one adds the top fully-connected layer to the basic configuration (6LFC, 603 + 33 features). The last one incorporates both the top fully-connected layer and the spatial prior (6LOFC, 603 + 33 + 33 features). Results show that both the spatial prior and the top fully-connected layer increase the performance of the system significantly. However, the impact of the top fully-connected layer is superior to the spatial prior one. Therefore, adding a top fully-connected layer to learn a global image descriptor is a clear advantage of our system.

In order to show the representative power of the top fully-connected layer, we use its output as a global image descriptor and, given an input image, we perform a ranking based on the angle between the global descriptor of the input image and all the global descriptors of the training

set. The angle is a proper measure of (dis-)similarity since the softmax classifier is based on linear hyperplanes before the exponentiation. Figure 4.14 shows some of the obtained results. The image on the left is an input image from the test set, and the four images in the same row are the retrieved and ranked images from the training set. These results show that with our global descriptor, we achieve a similar effect as [137]. Since the global descriptor is replicated for all the image pixels, its information acts as a global contextual priming for the classification. When used in conjunction with the top-down semantic feedback, this descriptor is able to blend appearance and semantic global features in a very compact way.

4.2.4.2 Effect of Top-down Semantic Feedback

In this second experiment, we analyze the effect of incorporating top-down semantic feedback to our architecture. This feedback allows to generate successive parsing hypothesis and refine them progressively. Since the deep architecture learns features from the input image and the previous posterior probability, the system is able to learn appearance-semantic configurations from the second iteration. Figure 4.15 shows this effect in terms of pixel accuracy (left) and per class average accuracy (right) as a function of iterations. The red dashed line represents the performance of a single iteration when using 10% of the training data (twice the quantity used in the 5 iterations). Results suggest that the top-down feedback is a fundamental component to obtain state-of-the-art-performance.

The per class accuracy is also shown in Figure 4.16 separately for each class, ordered by decreasing performance. From the results, we observe that the top-down semantic feedback improves the per class accuracy for almost all classes. The improvement is particularly relevant when considering rare classes, showing that semantic feedback contributes to learning a meaningful context. However, it is also present in the most common classes, showing that the algorithm is refining the parsing by better delineating boundaries and/or removing noisy classifications. Figure 4.17 shows some visual results, where the above-mentioned effects can be observed. The result in the first row is especially interesting: the first iteration presents a lot of heterogeneous classes, as road, mountain, car, sea, grass and field; this can be explained by the poor global appearance image prior information (see last row of Figure 4.14). However, in subsequent iterations, both local and global information allow to reject inconsistent classes rapidly (first 3 iterations), while refining the boundaries of classification.

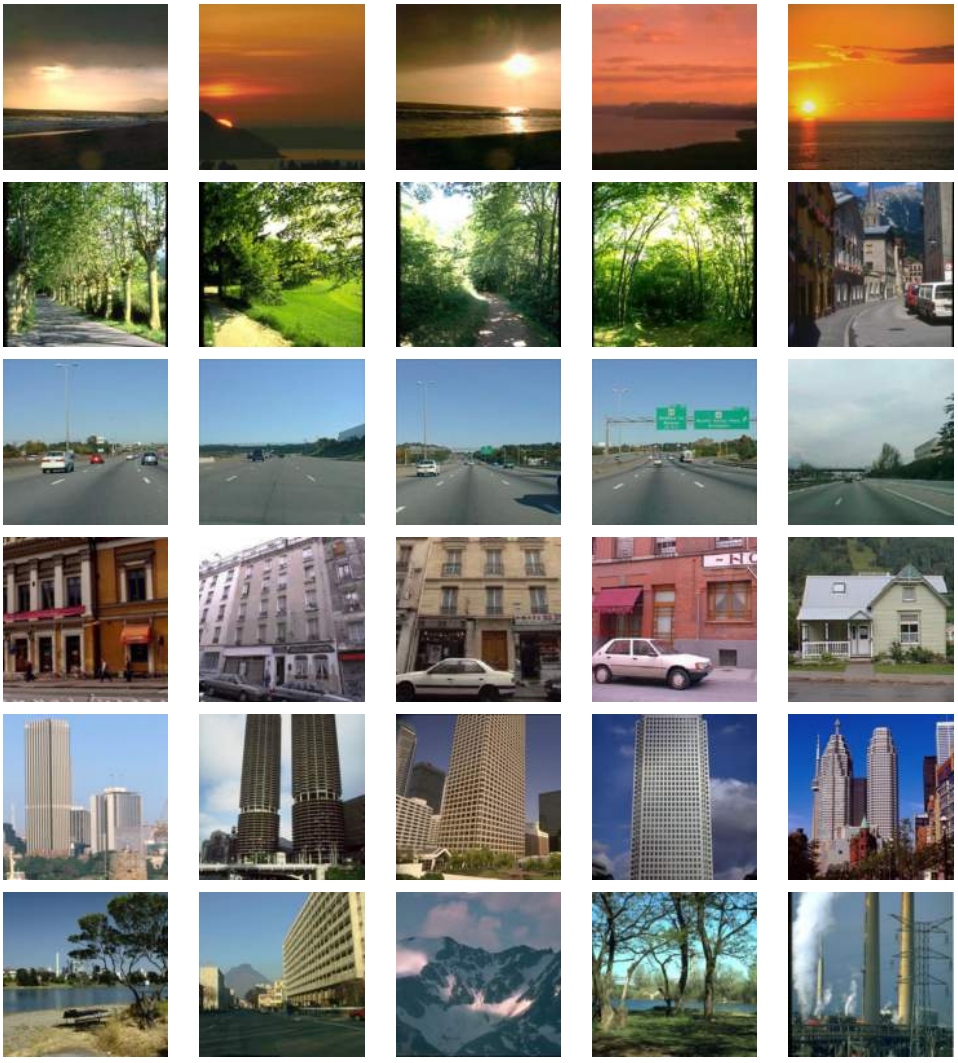


Figure 4.14: Six examples of the SIFTflow ranking obtained by using the top fully-connected layer of our system. See text for a detailed explanation.

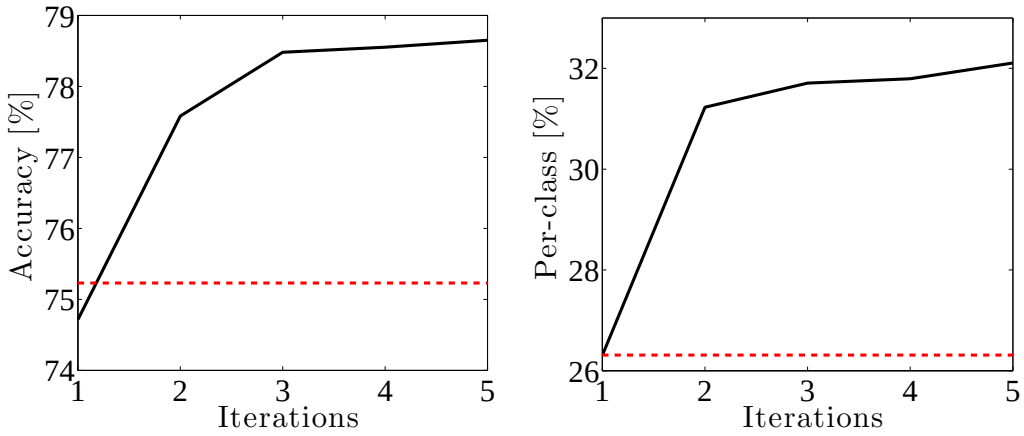


Figure 4.15: SIFTflow global accuracy (left) and per class average accuracy (right) as a function of the iterations when using the unrolled top-down semantic feedback architecture.

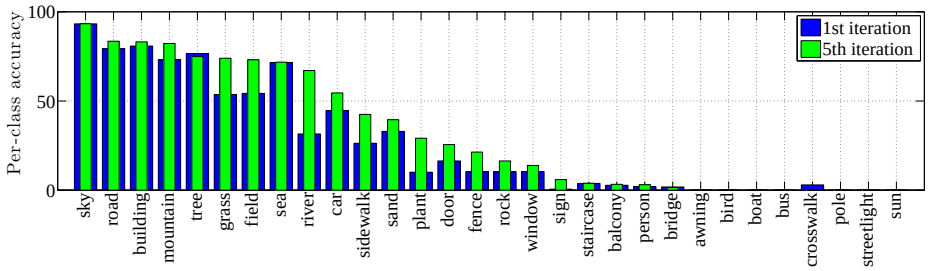


Figure 4.16: SIFTflow per class accuracy, in descending order for the 1st and 5th iteration.

4.2.5 Comparison to State-of-the-art

In this last experimental section, we aim to highlight the potential of our approach. Table 4.2 summarizes the results obtained on SIFTflow test set compared to the best performing convolutional deep methods. Our method outperforms previous convolutional deep approaches and gets really close to the result reported by the non-parametric method in [137], which obtains 79.2% and 33.8% for global and average per class accuracy, respectively. The contribution of the top-down semantic feedback becomes evident, providing an improvement of 4% in accuracy and 5.8% in per class average accuracy over 5 iterations.

An important characteristic of our method is that the training procedure is one order of magnitude faster than previous methods (see Table 4.1). We trained the 5-iteration system in less than 4 hours on a quad-core i7@2.3Ghz, using mildly-optimised Matlab code. Testing speed is comparable to previous convolutional deep methods and allows to process a 128×128

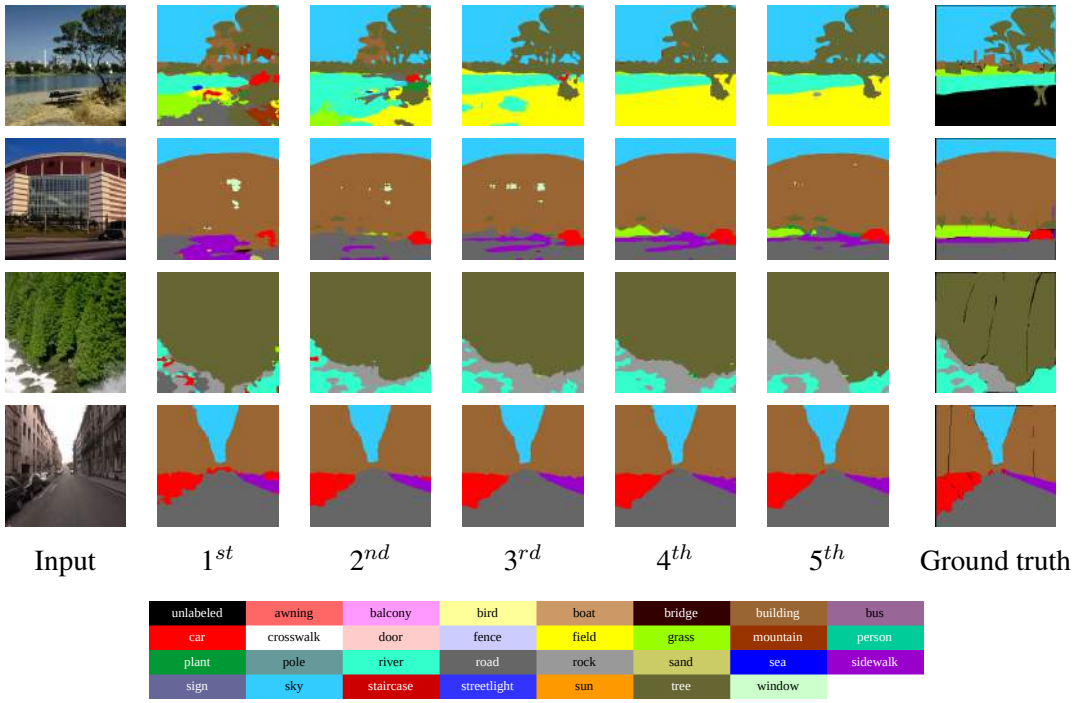


Figure 4.17: SIFTflow test results of image parsing for four test images (top). Color coded legend (bottom).

pixels image in 4.4 seconds; which compares very well with the 20 seconds per image required by [137].

4.2.6 Summary

In this section, we proposed a strategy to incorporate top-down semantic feedback to an image parsing system based on convolutional deep networks. The strategy shows that global appearance/semantic features can be easily incorporated to standard convolutional architectures and exploits the properties of the EPLS algorithm to achieve an efficient training.

All experiments were designed to study the impact of the method's novelties and provide a fair comparison with similar state-of-the-art methods. Results showed that combining both global appearance descriptor and top-down semantic feedback, we can improve the state-of-the-art of convolutional deep networks in the field of image parsing. Nevertheless, the performance of our method can still be improved in at least two ways: (1) the multi-scale approach

Table 4.2: Comparison with convolutional deep state-of-the-art methods in terms of global and average per class accuracy. The improvement from 1st to 5th iteration is provided in the last row.

	Global	Per class
Farabet et al. [135]	78.5%	29.6%
Pinheiro et al. [136]	77.7%	29.8 %
1st iteration	74.7%	26.3%
5th iteration	**78.7%**	**32.1%**
Δ (5th - 1st)	+4.0%	+5.8%

proposed in [135] could be used in our approach by learning multiple deep architectures and feeding the result from different scales into the softmax classifier; and (2) adding transformations to the training set, such as horizontal flipping, rotations, etc, could also help improving the system performance, as shown in [135]. Moreover, the scalability of the proposed method with respect to the number of classes is potentially problematic, since the posterior probability map is used as input to the next iteration, increasing its dimensionality. A classical dimensionality reduction technique, such as an auto-encoder, could be employed to alleviate this problem.

4.3 Summary of Deep EPLS Applications

In this chapter, we exploited and extended the EPLS algorithm to train deep architectures. Deep architectures trained by means of EPLS were successfully evaluated in the context of remote sensing to solve image and pixel classification tasks in a wide variety of scenarios. Moreover, EPLS was used as a basis to train modified deep architectures that include global appearance and global semantic features for image parsing. With all these experimental support, we demonstrated the EPLS potential and ability to find good parameter configurations for deep architectures, especially when the amount of labeled samples is limited.

5

FitNets: Hints for Thin Deep Nets

Although unsupervised pre-training has proven to be effective at learning deep architectures and is a potentially interesting research field, with the appearance of very large labeled dataset [140], most of the recent industrial success of deep learning has revolved around supervised learning [12]. Supervised deep networks have recently exhibited state-of-the-art performance in computer vision tasks such as image classification and object detection [18, 19]. These top-performing systems usually involve very *wide* and *deep* networks, with numerous parameters. Once learned, a major drawback of such wide and deep models is that they result in very time consuming systems at inference time, since they need to perform a huge number of multiplications. Moreover, having large amounts of parameters makes the models high memory demanding. For these reasons, wide and deep top-performing networks are not well suited for applications with memory or time limitations.

There have been several attempts in the literature to tackle the problem of model compression to reduce the computational burden at inference time. In [141], authors propose to train a neural network to mimic the output of a complex and large ensemble. The method uses the ensemble to label unlabeled data and trains the neural network with the data labeled by the ensemble, thus mimicking the function learned by the ensemble and achieving similar accuracy. The idea has been recently adopted in [142] to compress deep and wide networks into shallower but even wider ones, where the compressed model mimics the function learned by the complex model, in this case, by using data labeled by a deep (or an ensemble of deep) networks. More recently, Knowledge Distillation (KD) [143] was introduced as a model compression framework, which eases the training of deep networks by following a student-teacher paradigm, in which the student is penalized according to a softened version of the teacher's output. The framework compresses an ensemble of deep networks (*teacher*) into a *student* network of *similar depth*. To do so, the student is trained to predict the output of the teacher, as well as the true classification labels. All previous works related to Convolutional Neural Networks focus on compressing a teacher network or an ensemble of networks into either networks of similar width and depth or into shallower and wider ones; not taking advantage of depth.

As discussed in Chapter 2, depth is a fundamental aspect of representation learning, since it encourages the re-use of features, and leads to more abstract and invariant representations at higher layers [8]. The importance of depth has been verified (1) theoretically: deep representations are exponentially more expressive than shallow ones for some families of functions [144]; and (2) empirically: the two top-performers of ImageNet use convolutional deep networks with

19 and 22 layers, respectively [18] and [19]. Although, as stated in [145], regularization techniques such as dropout [42] along with new activation functions and proper initialization have been key factors to train deep networks in a supervised fashion [17], the difficulties encountered while training the intermediate and lower layers in deep architectures still appear to be problematic. In the past few years, methods to assist the supervised training of deep architectures have been introduced in the literature (see Chapter 2 for details). These methods often provide some kind of guidance to intermediate layers to help learning very deep networks [16, 19, 77, 80, 81, 82].

In this chapter, we will present another important contribution of this thesis [25], which addresses the network compression problem by trading width for depth. In Section 5.1 we will propose a novel approach to train *thin* and *deep* networks, called *FitNets*, to compress *wide* and shallower (but still *deep*) networks. The method is rooted in the recently proposed Knowledge Distillation (KD) [143] and extends the idea to allow for thinner and deeper student models, by introducing *intermediate-level hints* from the teacher hidden layers to guide the training process of the student, i.e. we want the student network (FitNet) to learn an intermediate representation that is predictive of the intermediate representations of the teacher network. Hints allow the training of thinner and deeper networks. After that, in Section 5.2 we will validate the proposed method on benchmark datasets and provide evidence that our method matches or outperforms the teacher's performance, while requiring notably fewer parameters and multiplications. Section 5.3 will be devoted to experimentally confirm that having deeper models allow us to generalize better, whereas making these models thin help us reduce the computational burden significantly. Finally, we will summarize the contribution in Section 5.4

5.1 Method

In this section, we will detail the proposed student-teacher framework to train FitNets from shallower and wider nets. First, in Section 5.1.1 we will review the recently proposed KD. Second, in Section 5.1.2, we will highlight the proposed hints algorithm to guide the FitNet throughout the training process. Finally, in Section 5.1.3, we will describe how the FitNet is trained in a stage-wise fashion.

5.1.1 Review of Knowledge Distillation

In order to obtain a faster inference, we explore the recently proposed compression framework [143], which trains a *student network*, from the softened output of an ensemble of wider networks, *teacher network*. The idea is to allow the student network to capture not only the information provided by the true labels, but also the finer structure learned by the teacher network. The framework can be summarized as follows.

Let T be a teacher network with a softmax activation $\mathrm{P_T} = \mathrm{softmax}(\mathbf{a_T})$ where $\mathbf{a_T}$ is the vector of teacher pre-softmax activations, for some example. In the case where the teacher model is a single network, $\mathbf{a_T}$ represents the weighted sums of the units of the last representation layer, whereas if the teacher model is the result of an ensemble either $\mathrm{P_T}$ or $\mathbf{a_T}$ are obtained by averaging outputs from different networks (respectively for arithmetic or geometric averaging). Let S be a student network with parameters $\mathbf{W_S}$ and output probability $\mathrm{P_S} = \mathrm{softmax}(\mathbf{a_S})$, where $\mathbf{a_S}$ is the student's softmax pre-activation. The student network will be trained such that its output $\mathrm{P_S}$ is similar to the teacher's output $\mathrm{P_T}$, as well as to the true labels $\mathbf{y}$. Since $\mathrm{P_T}$ might be very close to the one hot code representation of the sample's true label, a relaxation $\tau > 1$ is introduced to soften the signal arising from the output of the teacher network, and thus, provide more information during training[1]. The same relaxation is applied to the output of the student network ($\mathrm{P_S^\tau}$), when it is compared to the teacher's softened output ($\mathrm{P_T^\tau}$):

$$\mathrm{P_T^\tau} = \mathrm{softmax}\left(\frac{\mathbf{a_T}}{\tau}\right), \quad \mathrm{P_S^\tau} = \mathrm{softmax}\left(\frac{\mathbf{a_S}}{\tau}\right). \tag{5.1}$$

The student network is then trained to optimize the following loss function:

$$\mathcal{L}_{KD}(\mathbf{W_S}) = \mathcal{H}(\mathbf{y}, \mathrm{P_S}) + \lambda \mathcal{H}(\mathrm{P_T^\tau}, \mathrm{P_S^\tau}), \tag{5.2}$$

where $\mathcal{H}$ refers to the cross-entropy and λ is a tunable parameter to balance both cross-entropies. Note that the first term in Eq. (5.2) corresponds to the traditional cross-entropy between the output of a (student) network and the labels, whereas the second term enforces the student network to learn from the softened output of the teacher network.

To the best of our knowledge, KD is designed such that student networks mimic teacher architectures of similar depth. Although we found the KD framework to achieve encouraging

[1]For example, as argued in [143], with softened outputs, more information is provided about the relative similarity of the input to classes other than the one with the highest probability.

results even when student networks have slightly deeper architectures, as we increase the depth of the student network, KD training still suffers from the difficulty of optimizing deep nets (see Section 5.3.1).

5.1.2 Hint-based Training

In order to help the training of deep FitNets (deeper than their teacher), we introduce *hints* from the teacher network. A *hint* is defined as the output of a teacher's hidden layer responsible for guiding the student's learning process. Analogously, we choose a hidden layer of the FitNet, the *guided* layer, to learn from the teacher's hint layer. We want the guided layer to be able to predict the output of the hint layer. Note that having hints is a form of regularization and thus, the pair hint/guided layer has to be chosen such that the student network is not over-regularized. The deeper we set the guided layer, the less flexibility we give to the network and, therefore, FitNets are more likely to suffer from over-regularization. In our case, we choose the hint to be the middle layer of the teacher network. Similarly, we choose the guided layer to be the middle layer of the student network.

Given that the teacher network will usually be wider than the FitNet, the selected hint layer may have more outputs than the guided layer. For that reason, we add a regressor to the guided layer, whose output matches the size of the hint layer. Then, we train the FitNet parameters from the first layer up to the guided layer as well as the regressor parameters by minimizing the following loss function:

$$\mathcal{L}_{HT}(\mathbf{W_{Guided}}, \mathbf{W_r}) = \frac{1}{2}||u_h(\mathbf{x}; \mathbf{W_{Hint}}) - r(v_g(\mathbf{x}; \mathbf{W_{Guided}}); \mathbf{W_r})||^2, \tag{5.3}$$

where u_h and v_g are the teacher/student deep nested functions up to their respective hint/guided layers with parameters $\mathbf{W_{Hint}}$ and $\mathbf{W_{Guided}}$, r is the regressor function on top of the guided layer with parameters $\mathbf{W_r}$. Note that the outputs of u_h and r have to be comparable, i.e. u_h and r must be the same non-linearity.

Nevertheless, using a fully-connected regressor increases the number of parameters and the memory consumption dramatically in the case where the guided and hint layers are convolutional. Let $\mathrm{R}^h_\mathrm{T} \times \mathrm{C}^h_\mathrm{T}$ and F^h_T be the teacher hint's spatial size and number of features, respectively. Similarity, let $\mathrm{R}^g_\mathrm{S} \times \mathrm{C}^g_\mathrm{S}$ and F^g_S be the FitNet guided layer's spatial size and number of features. The number of parameters in the weight matrix of a fully connected regressor is $\mathrm{R}^h_\mathrm{T} \times \mathrm{C}^h_\mathrm{T} \times \mathrm{F}^h_\mathrm{T} \times \mathrm{R}^g_\mathrm{S} \times \mathrm{C}^g_\mathrm{S} \times \mathrm{F}^g_\mathrm{S}$. To mitigate this limitation, we use a convolutional regressor instead. The convolutional regressor is designed such that it considers approximately the same

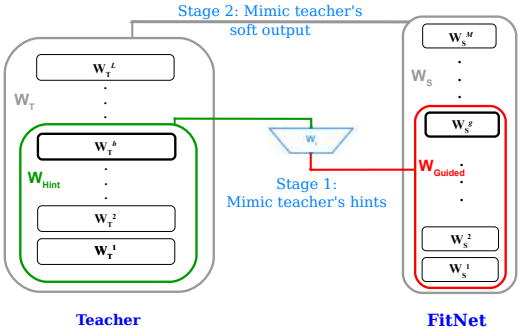
spatial region of the input image as the teacher hint. Therefore, the output of the regressor has the same spatial size as the teacher hint. Given a teacher hint of spatial size $\mathrm{R}_{\mathrm{T}}^{h} \times \mathrm{C}_{\mathrm{T}}^{h}$, the regressor takes the output of the Fitnet's guided layer of size $\mathrm{R}_{\mathrm{S}}^{g} \times \mathrm{C}_{\mathrm{S}}^{g}$ and adapts its kernel shape $\mathrm{w}_{\mathrm{r}}^{r} \times \mathrm{w}_{\mathrm{c}}^{r}$ such that $\mathrm{R}_{\mathrm{S}}^{g} - \mathrm{w}_{\mathrm{r}}^{r} + 1 = \mathrm{R}_{\mathrm{T}}^{h}$ and $\mathrm{C}_{\mathrm{S}}^{g} - \mathrm{w}_{\mathrm{c}}^{r} + 1 = \mathrm{C}_{\mathrm{T}}^{h}$. The number of parameters in the weight matrix of a the convolutional regressor is $\mathrm{w}_{\mathrm{r}}^{r} \times \mathrm{w}_{\mathrm{c}}^{r} \times \mathrm{F}_{\mathrm{T}}^{h} \times \mathrm{F}_{\mathrm{S}}^{g}$, where $\mathrm{w}_{\mathrm{r}}^{r} \times \mathrm{w}_{\mathrm{c}}^{r}$ is significantly lower than $\mathrm{R}_{\mathrm{T}}^{h} \times \mathrm{C}_{\mathrm{T}}^{h} \times \mathrm{R}_{\mathrm{S}}^{g} \times \mathrm{C}_{\mathrm{S}}^{g}$.

5.1.3 FitNet Stage-wise Training

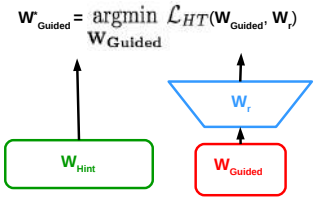
We train the FitNet in a stage-wise fashion following the student/teacher paradigm. Figure 5.1 summarizes the training pipeline. Starting from a trained teacher network and a randomly initialized FitNet (Fig. 5.1 (a)), we add a regressor parameterized by $\mathbf{W_r}$ on top of the FitNet guided layer and train the FitNet parameters $\mathbf{W_{Guided}}$ up to the guided layer to minimize Eq. (5.3) (see Fig. 5.1 (b)). Finally, from the pre-trained parameters, we train the parameters of whole FitNet $\mathbf{W_S}$ to minimize Eq. (5.2) (see Fig. 5.1 (c)). Algorithm 3 details the FitNet training process.

Algorithm 3 FitNet Stage-Wise Training.
The algorithm receives as input the trained parameters $\mathbf{W_T}$ of a teacher, the randomly initialized parameters $\mathbf{W_S}$ of a FitNet, and two indices h and g corresponding to hint/guided layers, respectively. Let $\mathbf{W_{Hint}}$ be the teacher's parameters up to the hint layer h. Let $\mathbf{W_{Guided}}$ be the FitNet's parameters up to the guided layer g. Let $\mathbf{W_r}$ be the regressor's parameters. The first stage consists in pre-training the student network up to the guided layer, based on the prediction error of the teacher's hint layer (line 4). The second stage is a KD training of the whole network (line 6).

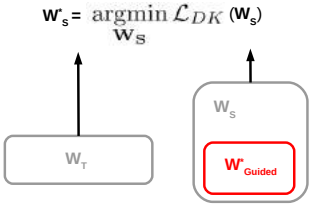
Input: $\mathbf{W_S}, \mathbf{W_T}, g, h$
Output: $\mathbf{W_S^*}$
$\mathbf{W_{Hint}} \leftarrow \{\mathbf{W_T}^1, \ldots, \mathbf{W_T}^h\}$
$\mathbf{W_{Guided}} \leftarrow \{\mathbf{W_S}^1, \ldots, \mathbf{W_S}^g\}$
Intialize $\mathbf{W_r}$ to small random values
$\mathbf{W^*_{Guided}} \leftarrow \arg\min_{\mathbf{W_{Guided}}} \mathcal{L}_{HT}(\mathbf{W_{Guided}}, \mathbf{W_r})$
$\{\mathbf{W_S}^1, \ldots, \mathbf{W_S}^g\} \leftarrow \{\mathbf{W_{Guided}}^{*1}, \ldots, \mathbf{W_{Guided}}^{*g}\}$
$\mathbf{W_S^*} \leftarrow \arg\min_{\mathbf{W_S}} \mathcal{L}_{KD}(\mathbf{W_S})$



(a) Teacher and Student Networks



(b) Hints Training



(c) Knowledge Distillation

Figure 5.1: Training a student network using hints.

5.1.4 Relation to Curriculum Learning

In this section, we argue that our hint-based training with KD can be seen as a particular form of Curriculum Learning [82]. Curriculum learning has proven to accelerate the training convergence as well as potentially improve the model generalization by properly choosing a sequence of training distributions seen by the learner: from simple examples to more complex ones. A curriculum learning extension [80] has also shown that by using guidance hints on an intermediate layer during the training, one could considerably ease training. However, [82] uses hand-defined heuristics to measure the "simplicity" of an example in a sequence and [80]'s guidance hints require some prior knowledge of the end-task. Both of these curriculum learning strategies tend to be problem-specific.

Our approach alleviates this issue by using a teacher model. Indeed, intermediate representations learned by the teacher are used as hints to guide the FitNet optimization procedure. In addition, the teacher confidence provides a measure of example "simplicity" by means of

teacher cross-entropy term in Eq. (5.2). This term ensures that examples with a high teacher confidence have a stronger impact than examples with low teacher confidence: the latter correspond to probabilities closer to the uniform distribution, which exert less of a push on the student parameters. In other words, the teacher penalizes the training examples according to its confidence. Note that parameter λ in Eq. (5.2) controls the weight given to the teacher cross-entropy, and thus, the importance given to each example. In order to promote the learning of more complex examples (examples with lower teacher confidence), we gradually anneal λ during the training with a linear decay. The curriculum can be seen as composed of two stages: first learn intermediate concepts via the hint/guided layer transfer, then train the whole student network jointly, annealing λ, which allows easier examples (on which the teacher is very confident) to initially have a stronger effect, but progressively decreasing their importance as λ decays. Therefore, the hint-based training introduced in this chapter is a generic curriculum learning approach, where prior information about the task-at-hand is deduced purely from the teacher model.

5.2 Results on Benchmark Datasets

In this section, we show the results on several benchmark datasets[2].

5.2.1 CIFAR-10 and CIFAR-100

The CIFAR-10 and CIFAR-100 datasets [146] are composed of 32x32 pixel RGB images belonging to 10 and 100 different classes, respectively. They both contain 50K training images and 10K test images. CIFAR-10 has 1000 samples per class, whereas CIFAR-100 has 100 samples per class. Like in [35], we normalized the datasets for contrast normalization and applied ZCA whitening.

CIFAR-10: To validate our approach, we train a teacher network of maxout convolutional layers as reported in [35] and design a FitNet with 17 maxout convolutional layers, followed by a maxout fully-connected layer and a top softmax layer, with roughly $1/3$ of the parameters. The 11th layer of the student network is trained to mimic the 2nd layer of the teacher network. Like in [35, 81], we augment the data with random flipping during training. Table 5.1 summarizes the obtained results. Our student model outperforms the teacher model, while

[2]Code to reproduce the experiments publicly available: `https://github.com/adri-romsor/FitNets`

requiring notably less parameters, suggesting that depth is crucial to achieve better representations. When compared to network compression methods, our algorithm achieves outstanding results; i.e. the student network achieves an accuracy of 91.61%, which is significantly higher than the top-performer 85.8% of [142], while requiring roughly 28 times fewer parameters. When compared to state-of-the-art methods, our algorithm matches the best performers.

One could argue the choice of hinting the inner layers with the hidden state of a wide teacher network. A straightforward alternative would be to hint them with the desired output. This could be addressed in a few different ways: (1) Stage-wise training, where stage 1 optimizes the 1st half of the network w.r.t. classification targets and stage 2 optimizes the whole network w.r.t. classification targets. In this case, stage 1 set the network parameters in a good local minima but such initialization did not seem to help stage 2 sufficiently, which failed to learn. To further assist the training of the thin and deep student network, we could add extra hints with the classification targets at different hidden layers. Nevertheless, as observed in [16], with supervised pre-training the guided layer may discard some factors from the input, which require more layers and non-linearity before they can be exploited to predict the classes. (2) Stage-wise training with KD, where stage 1 optimizes the 1st half of the net w.r.t. classification targets and stage 2 optimizes the whole network w.r.t. Eq. (5.2). As in the previous case, stage 1 set the network parameters in a good local minima but such initialization did not seem to help stage 2 sufficiently, which failed to learn. (3) Jointly optimizing both stages w.r.t. the sum of the supervised hint for the guided layer and classification target for the output layer. We performed this experiment, tried different initializations and learning rates with RMSprop [43] but we could not find any combination to make the network learn. Note that we could ease the training by adding hints to each layer and optimizing jointly as in Deeply Supervised Networks (DSN) [81]. Therefore, we built the above-mentioned 19-layer architecture and trained it by means of DSN, achieving a test performance of 88.2%, which is significantly lower than the performance obtained by the FitNets hint-based training (91.61%). Such result suggests that using a very discriminative hint w.r.t. classification at intermediate layers might be too aggressive; using a smoother hint (such as the guidance from a teacher network) offers better generalization. (4) Jointly optimizing both stages w.r.t. the sum of supervised hint for the guided layer and Eq. (5.2) for the output layer. Adding supervised hints to the middle layer of the network did not ease the training of such a thin and deep network, which failed to learn.

Moreover, when hinting the inner layers with the hidden state of a wide teacher network, one could choose to use more than one hint. Adding extra hint layers does not seem to have

Algorithm	# params	Accuracy
Compression		
FitNet	∼2.5M	**91.61%**
Teacher	∼9M	90.18%
Mimic single	∼54M	84.6%
Mimic single	∼70M	84.9%
Mimic ensemble	∼70M	85.8%
State-of-the-art methods		
Maxout		90.65%
Network in Network		91.2%
Deeply-Supervised Networks		**91.78%**
Deeply-Supervised Networks (19)		88.2%

Table 5.1: FitNets accuracy on CIFAR-10.

Algorithm	# params	Accuracy
Compression		
FitNet	∼2.5M	**64.96%**
Teacher	∼9M	63.54%
State-of-the-art methods		
Maxout		61.43%
Network in Network		64.32%
Deeply-Supervised Networks		**65.43%**

Table 5.2: FitNets accuracy on CIFAR-100.

any significant impact on the FitNet's performance as long as the last training stage has enough flexibility to adapt the network's parameters. However, choosing to hint one of the top FitNet's layers might result in performance drops due to an over-regularization of the network.

CIFAR-100: To validate our approach, we train a teacher network of maxout convolutional layers as reported in [35] and use the same FitNet architecture as in CIFAR-10. As in [81], we augment the data with random flipping during training. Table 5.2 summarizes the obtained results. As in the previous case, our FitNet outperforms the teacher model, reducing the number of parameters by a factor of 3 and, when compared to state-of-the-art methods, the FitNet provides near state-of-the-art performance.

5.2.2 SVHN

The SVHN dataset [147] is composed by 32×32 color images of house numbers collected by GoogleStreet View. There are 73,257 images in the training set, 26,032 images in the test set and 531,131 less difficult examples. We follow the evaluation procedure of [35] and use their maxout network as teacher. We train a 13-layer FitNet composed of 11 maxout convolutional layers, a fully-connected layer and a softmax layer.

Table 5.3 shows that our FitNet achieves comparable accuracy than the teacher despite using only 32% of teacher capacity. Our FitNet is comparable in terms of performance to other state-of-art methods, such as Maxout and Network in Network.

5.2.3 MNIST

As a sanity check for the training procedure, we evaluate the proposed method on the MNIST dataset [148]. MNIST is a dataset of handwritten digits (from 0 to 9) composed of 28x28 pixel greyscale images, with 60K training images and 10K test images. We train a teacher network of maxout convolutional layers as reported in [35] and design a FitNet twice as deep as the teacher network and with roughly 8% of the parameters. The 4th layer of the student network is trained to mimic the 2nd layer of the teacher network.

Table 5.4 reports the obtained results. To verify the influence of using hints, we train the FitNet architecture using either (1) standard backprop (w.r.t. classification labels), (2) KD or (3) Hint-based Training (HT). When training the FitNet with standard backprop from the softmax layer, the deep and thin architecture achieves 1.9% misclassification error. Using KD, the very same network achieves 0.65%, which confirms the potential of the teacher network; and when adding hints, the error still decreases to 0.51%. Furthermore, the student network achieves slightly better results than the teacher network, while requiring 12 times fewer parameters.

5.2.4 AFLW

AFLW [149] is a real-world face database, containing 25K annotated images. In order to evaluate the proposed framework in a face recognition setting, we extract positive samples by re-sizing the annotated regions of the images to fit 16x16 pixels patches. Similarly, we extract 25K 16x16 pixels patches not containing faces from ImageNet [140] dataset, as negative samples. We use 90% of the extracted patches to train the network.

Algorithm	**# params**	**Misclass**
Compression		
FitNet	∼1.5M	2.42%
Teacher	∼4.9M	$\mathbf{2.38}\%$
State-of-the-art methods		
Maxout		2.47%
Network in Network		2.35%
Deeply-Supervised Networks		$\mathbf{1.92}\%$

Table 5.3: FitNets SVHN error.

Algorithm	**# params**	**Misclass**
Compression		
Teacher	∼361K	0.55%
Standard backprop	∼30K	1.9%
KD	∼30K	0.65%
FitNet	∼30K	$\mathbf{0.51}\%$
State-of-the-art methods		
Maxout		0.45%
Network in Network		0.47%
Deeply-Supervised Networks		$\mathbf{0.39}\%$

Table 5.4: FitNets MNIST error.

In this experiment, we aim to evaluate the method on a different kind of architecture. Therefore, we train a teacher network of 3 ReLU convolutional layers and a sigmoid output layer. We design a first FitNet (FitNet 1) with 15 times fewer multiplications than the teacher network, and a second FitNet (FitNet 2) with 2.5 times fewer multiplications than the teacher network. Both FitNets have 7 ReLU convolutional layers and a sigmoid output layer.

The teacher network achieves 4.21% misclassification error on the validation set. We train both FitNets by means of KD and HT. On the one hand, we report a misclassification error of 4.58% when training FitNet 1 with KD and a misclassification error of 2.55% when training it with HT. On the other hand, we report a misclassification error of 1.95% when training FitNet 2 with KD and a misclassification error of 1.85% when training it with HT. These results show how the method is extensible to different kind of architectures and highlight the benefits of using hints, especially when dealing with thinner architectures.

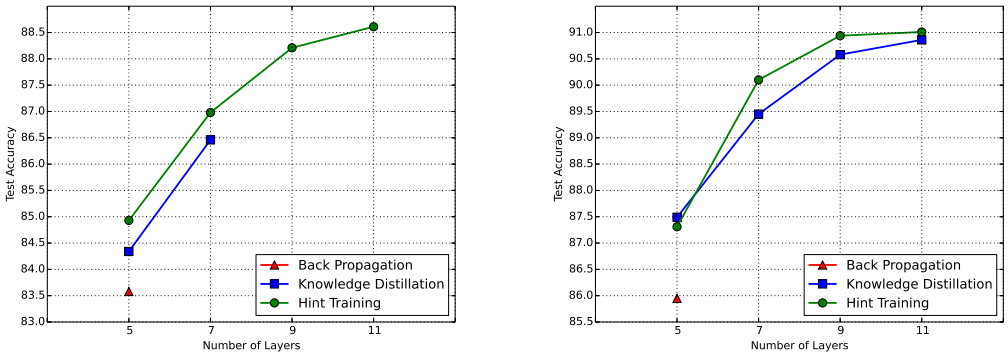
5.3 Analysis of Empirical Results

We empirically investigate the benefits of our approach by comparing various networks trained using standard backpropagation (cross-entropy w.r.t. labels), KD or Hint-based Training (HT). Experiments are performed on CIFAR-10 dataset [146].

We compare networks of increasing depth given a fixed computational budget. Each network is composed of successive convolutional layers of kernel size 3×3, followed by a maxout non-linearity and a non-overlapping 2×2 max-pooling. The last max-pooling takes the maximum over all remaining spatial dimensions leading to a 1×1 spatial support for each. We only change the depth and the number of features per convolution between different networks, i.e. the number of features per convolutional layer decreases as a network depth increases to respect a given computational budget.

5.3.1 Assisting the Training of Deep Networks

In this section, we investigate the impact of HT. We consider two computational budgets of approximately 30M and 107M operations, corresponding to the multiplications needed in an image forward propagation. For each computational budget, we train networks composed of 3, 5, 7 and 9 convolutional layers, followed by a fully-connected layer and a softmax layer. We compare their performances when they are trained with standard backpropagation, KD and HT.



(a) 30M Multiplications

(b) 107M Multiplications

Figure 5.2: Comparison of Standard Back-Propagation, Knowledge Distillation and Hint-based Training on CIFAR-10.

Figure 5.2 reports test on CIFAR-10 using early stopping on the validation set, i.e. we do not retrain our models on the training plus validation sets.

Due to their depth and small capacity, FitNets are hard to train. As shown in Figure 5.2(a), we could not train 30M multiplications networks with more than 5 layers with standard back-prop. When using KD, we succesfully trained networks up to 7 layers. Adding KD's teacher cross-entropy to the training objective (Eq. (5.2)) gives more importance to easier examples, i.e. samples for which the teacher network is confident and, can lead to a smoother version of the training cost [82]. Despite some optimization benefits, it is worth noticing that KD training still suffers from the increasing depth and reaches its limits for 7-layer networks. HT tends to ease these optimization issues and is able to train 13-layer networks of 30M multiplications. The only difference between HT and KD is the starting point in the parameter space: either random or obtained by means of the teacher's hint. On the one hand, the proliferation of local minima and especially saddle points in highly non-linear functions such as very deep networks highlights the difficulty of finding a good starting point in the parameter space at random [150]. On the other hand, results in Figure 5.2(a) indicate that HT can guide the student to a better initial position in the parameter space, from which we can minimize the cost through stochastic gradient descent. Therefore, HT provides benefits from an optimization point of view. Networks trained with HT also tend to yield *better test performances* than the other training methods when we fix the capacity and number of layers. For instance, in Figure 5.2(b), the 7-layers network, trained with hints, obtains a $+0.7\%$ performance gain on

Network	# layers	# params	# mult	Acc	Speed-up	Compression rate
Teacher	5	∼9M	∼725M	90.18%	1	1
FitNet 1	11	∼250K	∼30M	89.01%	**13.36**	**36**
FitNet 2	11	∼862K	∼108M	91.06%	4.64	10.44
FitNet 3	13	∼1.6M	∼392M	91.10%	1.37	5.62
FitNet 4	19	∼2.5M	∼382M	**91.61%**	1.52	3.60

Table 5.5: FitNets accuracy/speed trade-off on CIFAR-10.

the test set compared to the model that does not use any hints (the accuracy increases from 89.45% to 90.1%). As pointed by [14], pre-training strategies can act as regularizers. These results suggest that HT is a stronger regularizer than KD, since it leads to better generalization performance on the test set. Finally, Figure 5.2 highlights that deep models have better performances than shallower ones given a fixed computational budget. Indeed, considering networks that are trained with hints, an 11-layer network outperforms a 5-layer network by an absolute improvement of 4.11% for 107M multiplications and of 3.4% for 30M multiplications. Therefore, the experiments validate our hypothesis that given a fixed number of computations, we leverage depth in a model to achieve faster computation and better generalization.

In summary, this experiment shows that (1) using HT, we are able to train deeper models than with standard back-propagation and KD; and (2) given a fixed capacity, deeper models performed better than shallower ones.

5.3.2 Trade-off Between Model Performance and Efficiency

To evaluate FitNets efficiency, we measure their total inference times required for processing CIFAR-10 test examples on a GPU as well as their parameter compression. Table 5.5 reports both the speed-up and compression rate obtained by various FitNets w.r.t. the teacher model along with their number of layers, capacity and accuracies. In this experiment, we retrain our FitNets on training plus validation sets as in [35], for fair comparison with the teacher.

FitNet 1, our smallest network, with $36\times$ less capacity than the teacher, is one order of magnitude faster than the teacher and only witnesses a minor performance decrease of 1.3%. FitNet 2, slightly increasing the capacity, outperforms the teacher by 0.9%, while still being faster by a strong 4.64 factor. By further increasing network capacity and depth in FitNets 3 and 4, we improve the performance gain, up to 1.6%, and still remain faster than the teacher.

Although a trade-off between speed and accuracy is introduced by the compression rate, FitNets tend to be significantly faster, matching or outperforming their teacher, even when having low capacity.

A few works such as matrix factorization [151, 152] focus on speeding-up deep networks' convolutional layers at the expense of slightly deteriorating their performance. Such approaches are complementary to FitNets and could be used to further speed-up the FitNet's convolutional layers.

Other works related to quantization schemes [153, 154, 155] aim at reducing storage requirements. Unlike FitNets, such approaches witness a little decrease in performance when compressing the network parameters. Exploiting depth allows FitNets to obtain performance improvements w.r.t. their teachers, even when reducing the number of parameters $10\times$. However, we believe that quantization approaches are also complementary to FitNets and could be used to further reduce the storage requirements. It would be interesting to compare how much redundancy is present in the filters of the teacher networks w.r.t. the filters of the FitNet and, therefore, how much FitNets filters could be compressed without witnessing significant performance drop. This analysis is outside of the scope of this thesis and is left as future work.

5.4 Summary

In this chapter, we proposed a novel framework to compress *wide* and *deep* networks into *thin* and *deeper* ones, by introducing *intermediate-level hints* from the teacher hidden layers to guide the training process of the student. We were able to use these hints to train very deep student models with less parameters, which can generalize better and/or run faster than their teachers. We provided empirical evidence that hinting the inner layers of a thin and deep network with the hidden state of a teacher network generalizes better than hinting them with the classification targets. Our experiments on benchmark datasets emphasized that deep networks with low capacity are able to extract feature representations that are comparable or even better than networks with as much as 10 times more parameters. The success of hint-based training suggests that more efforts should be devoted to exploring new training strategies to leverage the power of deep networks.

6

Conclusion

In the past few years, there have been plenty of theoretical and empirical evidence on the impact of depth in neural network's performance [17, 18, 144, 156]. However, increasing depth makes the training of a network more difficult. Despite the numerous efforts and advances proposed in the literature, the training of very deep networks is still an open problem. In this thesis, we aimed to present different alternatives to ease the training of deep architectures. We began the work based on the assumption that learning to represent the world in a general way is likely to be helpful to perform subsequent specific tasks, such as image classification or image parsing. To that end, we introduced an unsupervised learning algorithm to extract representations that allowed to efficiently learn deep architectures, providing discriminative features of increasing abstraction. Sparsity, more accurately lifetime sparsity, revealed to be a key component of the method, improving the discriminability of the extracted features and achieving compelling results in both shallow and deep settings. Moreover, we provided a detailed analysis on the influence of sparsity, which outlined the benefits of having sparse representations. Then, we went on applying the proposed algorithm to a wide variety of image classification and pixel classification (i.e. image parsing) problems, where the method confirmed its potential.

Unsupervised learning algorithms have proven to be effective at training deep architectures. However, with the introduction of very large labeled datasets, they seem to have gone into the background, enjoying only moderate attention but still being of great value when labeled data is scarce, and being outshone by supervised learning methods.

Supervised learning is currently the focus of many recent research advances, which have shown to excel at many computer vision tasks. Given their unquestionable success, they have rapidly found their way in the industry. This industry-wide phenomenon has highlighted new needs to be addressed. While large deep learning models exhibit state-of-the-art results at many computer vision tasks, they are not well suited for applications with time or memory limitations. Moreover, supervised methods still suffer from the difficulties of training the intermediate and lower layers of a deep model. Therefore, we proceeded with the proposal of a novel algorithm to compress state-of-the-art large wide and deep models into thin and deeper ones. In order to mitigate the optimization difficulties induced by depth and extending the idea in [143], we trained the compressed model to imitate not only the output but also the intermediate representation of the large state-of-the-art one, achieving faster-to-run or better-performing compressed models.

Some of the contributions of this thesis have already influenced the research of other groups; more specifically, a significant amount of research has followed on the idea of guiding

the learning process of models that are hard to train. In [157], authors propose to use a simple convolutional network to assist the training of a complex recurrent neural network. In [158], authors train a student network from a Bayesian predictive distribution of the teacher, which is a Monte Carlo ensemble of teacher neural networks (TNN). Most of these methods follow the student-teacher paradigm introduced in [143] and somehow transfer the knowledge from the teacher network to the student network.

Moreover, the task of speeding up state-of-the-art top performing models has become an active topic of research. Following the FitNets hint-based training, [159] incrementally train a network by iteratively introducing new subsets of the input data and using the network trained in the previous iteration as a guide to the current one. In [160], authors propose to sparsify the convolution operations to attain acceleration. In [161], authors speed up very deep networks by assuming low rank assumptions and decompositions.

Further investigation should be devoted to the topics discussed in this thesis. On one hand, one lesson we can glean is that, despite the advances in unsupervised learning, it still remains a holy grail for machine learning and future research should attempt to better understand the kind of general features that would be useful to succeed at subsequent tasks. Although sparsity has shown to be an important property of the learned features, it is not clear whether it is equally influential for all representation layers or whether different layers could benefit from different levels of sparsity.

On the other hand, supervised learning with very large datasets has become a new academic and industry-wide trend that is paving the way for many attractive problems that deserve future attention. First, the encouraging results presented in this thesis highlight the potential of transferring knowledge from a teacher network to a student network. Hence, the student-teacher paradigm could be extended in order to give the student network the opportunity to learn and generalize from different teachers, which would be proficient at different tasks. Second, the promising results achieved by thin and deep networks highlight the potential of such architectures and suggest that more efforts should be devoted to explore new training strategies that would be proficient, possibly even without the need for a pre-trained network. Work in this direction already includes [162], which follows the same spirit of FitNets, aiming to exploit the advantages of depth, but delves into an alternative that does not require any pre-trained model. Third, computational complexity remains a challenge, not only from the inference but also from the training perspective; it is not realistic to keep increasing the computational power in order to successfully address more ambitious tasks. Last, from a computer vision perspective,

it would be interesting to further investigate the quality of the features in FitNets compared to their teacher or even to similar architectures trained without requiring a pre-trained model. One way of doing this is by visualizing the learned representations as in [40, 41, 163, 164].

7

Publications

7. PUBLICATIONS

The work presented in this thesis has appeared in other journal and/or conference publications. The publications derived from this thesis include:

Meta-parameter free Unsupervised Sparse Feature Learning; Adriana Romero, Petia Radeva, and Carlo Gatta; IEEE Transactions on Pattern Analysis and Machine Intelligence, 37(8):17161722, 2015.

Unsupervised Deep Feature Extraction Of Hyperspectral Images; Adriana Romero, Carlo Gatta, and Gustau Camps-Valls; IEEE Transactions on Geoscience and Remote Sensing, Accepted.

FitNets: Hints for Thin Deep Nets; Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio; In Proc. of the International Conference on Learning Representations, 2015.

Unsupervised Deep Feature Extraction Of Hyperspectral Images; Adriana Romero, Carlo Gatta, and Gustau Camps-Valls; In IEEE GRSS Workshop on Hyperspectral Image and Signal Processing (WHISPERS), 2014.

Unrolling Loopy Top-Down Semantic Feedback in Convolutional Deep Networks; Carlo Gatta, Adriana Romero, and Joost Van De Weijer; In IEEE Conference on Computer Vision & Pattern Recognition Workshops (CVPRW), Deep Vision, pages 504511, 2014.

Bibliography

[1] DAVID MARR. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. Henry Holt and Co., Inc., New York, NY, USA, 1982. 2

[2] PEDRO DOMINGOS. **A Few Useful Things to Know About Machine Learning**. *Communications of the ACM*, **55**(10):78–87, October 2012. 2

[3] DAVID G. LOWE. **Distinctive Image Features from Scale-Invariant Keypoints**. *International Journal of Computer Vision*, **60**:91–110, 2004. 2

[4] NAVNEET DALAL AND BILL TRIGGS. **Histograms of Oriented Gradients for Human Detection**. In CORDELIA SCHMID, STEFANO SOATTO, AND CARLO TOMASI, editors, *Proc. of the Conference on Computer Vision & Pattern Recognition*, **2**, pages 886–893. IEEE, June 2005. 2

[5] M. A. RANZATO, C. POULTNEY, S. CHOPRA, AND Y. LECUN. **Efficient learning of sparse representations with an energy-based model**. In *Advances in Neural Information Processing Systems*, pages 1137–1144, 2006. 3, 27, 41, 54

[6] J. YANG, K. YU, Y. GONG, AND T. HUANG. **Linear spatial pyramid matching using sparse coding for image classification**. In *Proc. of the Conference on Computer Vision & Pattern Recognition*, pages 1794–1801. IEEE, 2009. 3

[7] A. COATES, H. LEE, AND A. Y. NG. **An Analysis of Single-Layer Networks in Unsupervised Feature Learning**. In *Proc. of the International Conference on Artificial Intelligence and Statistics*, pages 214–223, 2011. 3, 48, 49, 51, 52

[8] YOSHUA BENGIO, AARON C. COURVILLE, AND PASCAL VINCENT. **Representation Learning: A Review and New Perspectives**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **35**(8):1798–1828, 2013. 3, 31, 41, 92

[9] JERRY A. FODOR AND ZENON W. PYLYSHYN. **Connectionism and cognitive architecture: A critical analysis**. *Cognition*, **28**:3–71, 1988. 3

[10] E. BIENENSTOCK AND S. GEMAN. **Compositionality in neural systems**. *The Handbook of Brain Theory and Neural Networks*, pages 223–226, 1995. 3

[11] ELIE BIENENSTOCK, STUART GEMAN, AND DANIEL POTTER. **Compositionality, MDL Priors, and Object Recognition**. In *Advances in Neural Information Processing Systems*, pages 838–844. MIT Press, 1997. 3

[12] YOSHUA BENGIO, IAN J. GOODFELLOW, AND AARON COURVILLE. **Deep Learning**. Book in preparation for MIT Press, 2015. 3, 13, 15, 32, 92

[13] YOSHUA BENGIO. **Learning deep architectures for AI**. *Foundations and Trends in Machine Learning*, **2**(1):1–127, 2009. 3, 26, 28

[14] D. ERHAN, P.A. MANZAGOL, Y. BENGIO, S. BENGIO, AND P. VINCENT. **The Difficulty of Training Deep Architectures and the effect of Unsupervised Pre-Training**. In *Proc. of the International Conference on Artificial Intelligence and Statistics*, pages 153–160, 2009. 3, 23, 104

[15] GEOFFREY E. HINTON, SIMON OSINDERO, AND YEE-WHYE TEH. **A fast learning algorithm for deep belief nets**. *Neural Computation*, **18**(7):1527–1554, 2006. 3, 4, 25, 27, 32, 40, 60, 61, 78, 80

[16] Y. BENGIO, P. LAMBLIN, D. POPOVICI, AND H. LAROCHELLE. **Greedy layer-wise training of deep networks**. In *Advances in Neural Information Processing Systems*, pages 153–160, 2007. 3, 4, 25, 32, 33, 40, 60, 78, 80, 93, 99

[17] A. KRIZHEVSKY, I. SUTSKEVER, AND G. HINTON. **Imagenet classification with deep convolutional neural networks**. In *Advances in Neural Information Processing Systems*, 2012. 3, 13, 16, 93, 108

[18] K. SIMONYAN AND A. ZISSERMAN. **Very deep convolutional networks for large-scale image recognition**. In *Proc. of the International Conference on Learning Representations*, 2015. 4, 92, 93, 108

[19] C. SZEGEDY, W. LIU, Y. JIA, P. SERMANET, S. REED, D.A., D. ERHAN, V. VANHOUCKE, AND A. RABINOVICH. **Going Deeper with Convolutions**. *CoRR*, **abs/1409.4842**, 2014. 4, 34, 92, 93

[20] MATTHEW D. ZEILER AND ROB FERGUS. **Visualizing and Understanding Convolutional Networks**. *CoRR*, **abs/1311.2901**, 2013. 4

[21] ADRIANA ROMERO, PETIA RADEVA, AND CARLO GATTA. **Meta-Parameter Free Unsupervised Sparse Feature Learning**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **37**(8):1716–1722, 2015. 6, 40, 61, 79, 80

[22] ADRIANA ROMERO, CARLO GATTA, AND GUSTAU CAMPS-VALLS. **Unsupervised Deep Feature Extraction Of Hyperspectral Images**. In *IEEE GRSS Workshop on Hyperspectral Image and Signal Processing (WHISPERS)*. IEEE, 2014. 6

[23] ADRIANA ROMERO, CARLO GATTA, AND GUSTAU CAMPS-VALLS. **Unsupervised Deep Feature Extraction for Remote Sensing Image Classification**. *Accepted to IEEE Transactions on Geoscience and Remote Sensing*, 2015. 6

[24] CARLO GATTA, ADRIANA ROMERO, AND JOOST VAN DE WEIJER. **Unrolling Loopy Top-Down Semantic Feedback in Convolutional Deep Networks**. In *IEEE Conference on Computer Vision & Pattern Recognition Workshops (CVPRW), Deep Vision*, pages 504–511. IEEE, 2014. 6

[25] ADRIANA ROMERO, NICOLAS BALLAS, SAMIRA EBRAHIMI KAHOU, ANTOINE CHASSANG, CARLO GATTA, AND YOSHUA BENGIO. **FitNets: Hints for Thin Deep Nets**. In *Proc. of the International Conference on Learning Representations*, 2015. 6, 93

[26] F. ROSENBLATT. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington, 1962. 8

[27] YANN LECUN, LEON BOTTOU, GENEVIEVE ORR, AND KLAUS MÜLLER. **Efficient BackProp**. In *Neural Networks: Tricks of the Trade*, pages 9–50. Springer Berlin, 1998. 13, 45, 46, 47

[28] KEVIN JARRETT, KORAY KAVUKCUOGLU, MARC'AURELIO RANZATO, AND LECUN. **What is the Best Multi-Stage Architecture for Object Recognition?** In *Proc. of the International Conference on Computer Vision*. IEEE, 2009. 13

[29] VINOD NAIR AND GEOFFREY E. HINTON. **Rectified Linear Units Improve Restricted Boltzmann Machines.** In JOHANNES FÜRNKRANZ AND THORSTEN JOACHIMS, editors, *Proc. of the International Conference on Machine Learning*, pages 807–814, 2010. 13

[30] XAVIER GLOROT, ANTOINE BORDES, AND YOSHUA BENGIO. **Deep Sparse Rectifier Neural Networks**. In GEOFFREY J. GORDON AND DAVID B. DUNSON, editors, *Proc. of the International Conference on Artificial Intelligence and Statistics*, **15**, pages 315–323. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011. 13

[31] PIERRE SERMANET, DAVID EIGEN, XIANG ZHANG, MICHAEL MATHIEU, ROB FERGUS, AND YANN LECUN. **OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks**. In *Proc. of the International Conference on Learning Representations*. CBLS, April 2014. 13

[32] CHARLES DUGAS, YOSHUA BENGIO, FRANÇOIS BÉLISLE, CLAUDE NADEAU, AND RENÉ GARCIA. **Incorporating Second-Order Functional Knowledge for Better Option Pricing**. In *Advances in Neural Information Processing Systems*, pages 472–478, 2001. 13

[33] ANDREW L. MAAS, AWNI Y. HANNUN, AND ANDREW Y. NG. **Rectifier Nonlinearities Improve Neural Network Acoustic Models**. In *Proc. of the International Conference on Machine Learning*, 2013. 13

[34] KAIMING HE, XIANGYU ZHANG, SHAOQING REN, AND JIAN SUN. **Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification**. *CoRR*, **abs/1502.01852**, 2015. 13, 17

[35] I.J. GOODFELLOW, D. WARDE-FARLEY, M. MIRZA, A. COURVILLE, AND Y. BENGIO. **Maxout Networks**. In *Proc. of the International Conference on Machine Learning*, 2013. 14, 17, 18, 98, 100, 101, 104

[36] LE CUN, B. BOSER, J. S. DENKER, D. HENDERSON, R. E. HOWARD, W. HUBBARD, AND L. D. JACKEL. **Handwritten Digit Recognition with a Back-Propagation Network**. In *Advances in Neural Information Processing Systems*, pages 396–404. Morgan Kaufmann, 1990. 14

[37] YANN LECUN AND YOSHUA BENGIO. *Convolutional Networks for Images, Speech and Time Series*, pages 255–258. The MIT Press, 1995. 14

[38] MIN LIN, QIANG CHEN, AND SHUICHENG YAN. **Network In Network**. *CoRR*, **abs/1312.4400**, 2013. 17, 18

[39] K. HORNIK, M. STINCHCOMBE, AND H. WHITE. **Multilayer Feedforward Networks Are Universal Approximators**. *Neural Networks*, **2**(5):359–366, July 1989. 17

[40] MATTHEW D. ZEILER AND ROB FERGUS. **Stochastic Pooling for Regularization of Deep Convolutional Neural Networks**. *CoRR*, **abs/1301.3557**, 2013. 18, 110

[41] JOST TOBIAS SPRINGENBERG, ALEXEY DOSOVITSKIY, THOMAS BROX, AND MARTIN RIEDMILLER. **Striving for Simplicity: The All Convolutional Net**. *CoRR*, **abs/1412.6806**, 2014. 18, 110

[42] G.E. HINTON, N. SRIVASTAVA, A. KRIZHEVSKY, I. SUTSKEVER, AND R.R. SALAKHUTDINOV. **Improving neural networks by preventing co-adaptation of feature detectors**. *CoRR*, **abs/1207.0580**, 2012. 20, 93

[43] T. TIELEMAN AND G. HINTON. **Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude**. COURSERA: Neural Networks for Machine Learning, 2012. 22, 99

[44] TOM SCHAUL, SIXIN ZHANG, AND YANN LECUN. **No More Pesky Learning Rates**. In *Proc. of the International Conference on Machine Learning*, 2013. 22, 45, 46, 47

[45] Y. LECUN, B. BOSER, J. DENKER, D. HENDERSON, R. HOWARD, W. HUBBARD, AND L. JACKEL. **Backpropagation applied to handwritten zip code recognition**. *Neural Computation*, 1989. 22

[46] DAVID E. RUMELHART, GEOFFREY E. HINTON, AND RONALD J. WILLIAMS. **Learning Representations by Back-propagating Errors**. In JAMES A. ANDERSON AND EDWARD ROSENFELD, editors, *Neurocomputing: Foundations of Research*, pages 696–699. MIT Press, Cambridge, MA, USA, 1988. 22, 26

[47] H. LAROCHELLE, D. ERHAN, A. COURVILLE, J. BERGSTRA, AND Y. BENGIO. **An Empirical Evaluation of Deep Architectures on Problems with Many Factors of Variation**. In *Proc. of the International Conference on Machine Learning*, pages 473–480, 2007. 23

[48] HUGO LAROCHELLE, YOSHUA BENGIO, JÉRÔME LOURADOUR, AND PASCAL LAMBLIN. **Exploring Strategies for Training Deep Neural Networks**. *Journal of Machine Learning Research*, **10**:1–40, 2009. 23, 25

[49] PASCAL VINCENT, HUGO LAROCHELLE, YOSHUA BENGIO, AND PIERRE-ANTOINE MANZAGOL. **Extracting and Composing Robust Features with Denoising Autoencoders**. In *Proc. of the International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM. 25, 27

[50] PASCAL VINCENT, HUGO LAROCHELLE, ISABELLE LAJOIE, YOSHUA BENGIO, AND PIERRE-ANTOINE MANZAGOL. **Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion**. *Journal of Machine Learning Research*, **11**:3371–3408, December 2010. 25, 27

[51] DUMITRU ERHAN, AARON COURVILLE, YOSHUA BENGIO, AND PASCAL VINCENT. **Why Does Unsupervised Pre-training Help Deep Learning?** In *Proc. of the International Conference on Artificial Intelligence and Statistics*, **9**, pages 201–208, 2010. 25, 32

[52] H. BOURLARD AND Y. KAMP. **Auto-association by multilayer perceptrons and singular value decomposition**. *Biological Cybernetics*, **59**(4):291–294, September 1988. 26

[53] P. DAYAN, G. E. HINTON, R. N. NEAL, AND R. S. ZEMEL. **The Helmholtz Machine**. *Neural Computation*, **7**:889–904, 1995. 26

[54] SALAH RIFAI, PASCAL VINCENT, XAVIER MULLER, XAVIER GLOROT, AND YOSHUA BENGIO. **Contracting auto-encoders: Explicit invariance during feature extraction**. In *Proc. of the International Conference on Machine Learning*, 2011. 27

[55] G. E. HINTON. **A Practical Guide to Training Restricted Boltzmann Machines**. Technical report, University of Toronto, 2010. 27

[56] HANLIN GOH, NICOLAS THOME, MATTHIEU CORD, AND JOO-HWEE LIM. **Unsupervised and Supervised Visual Codes with Restricted Boltzmann Machines**. In *Proc. of the European Conference on Computer Vision*, pages 298–311, 2012. 27

[57] MIGUEL A. CARREIRA-PERPINAN AND GEOFFREY E. HINTON. **On Contrastive Divergence Learning**. In *Proc. of the International Conference on Artificial Intelligence and Statistics*, 2005. 28

[58] T. TIELEMAN. **Training Restricted Boltzmann Machines using Approximations to the Likelihood Gradient**. In *Proc. of the International Conference on Machine Learning*, pages 1064–1071. ACM New York, NY, USA, 2008. 28

[59] H. LEE, C. EKANADHAM, AND A. Y. NG. **Sparse deep belief net model for visual area V2**. In *Advances in Neural Information Processing Systems*, pages 873–880, 2008. 28, 41

[60] B. OLSHAUSEN AND D. J. FIELD. **Sparse coding with an overcomplete basis set: a strategy employed by V1?** *Vision Research*, **37**(23):3311–3325, 1997. 28, 40, 41

[61] RAJAT RAINA, ALEXIS BATTLE, HONGLAK LEE, BENJAMIN PACKER, AND ANDREW Y. NG. **Self-taught Learning: Transfer Learning from Unlabeled Data**. In *Proc. of the International Conference on Machine Learning*, pages 759–766, 2007. 28, 40

[62] KORAY KAVUKCUOGLU, MARC'AURELIO RANZATO, ROB FERGUS, AND YANN LECUN. **Learning Invariant Features through Topographic Filter Maps**. In *Proc. of the Conference on Computer Vision & Pattern Recognition*. IEEE, 2009. 28, 40

[63] KORAY KAVUKCUOGLU, PIERRE SERMANET, Y-LAN BOUREAU, KAROL GREGOR, MICHAËL MATHIEU, AND YANN LECUN. **Learning Convolutional Feature Hierachies for Visual Recognition**. In *Advances in Neural Information Processing Systems*, 2010. 29, 31, 40, 58

[64] Y. C. PATI, R. REZAIFAR, AND P. S. KRISHNAPRASAD. **Orthogonal Matching Pursuit: recursive function approximation with applications to wavelet decomposition.** In *Asilomar Conference on Signals, Systems and Computers*, pages 40–44, 1993. 29

[65] T. BLUMENSATH AND M. E. DAVIES. **On the difference between orthogonal matching pursuit and orthogonal least squares**. *Unpublished manuscript*, 2007. 29

[66] ADAM COATES AND ANDREW NG. **The importance of encoding versus training with Sparse Coding and Vector Quantization**. In *Proc. of the International Conference on Machine Learning*, pages 921–928, 2011. 29, 35, 36, 48, 49, 51, 52

[67] J. NGIAM, P. W. KOH, Z. CHEN, S. BHASKAR, AND A. Y. NG. **Sparse Filtering**. In *Advances in Neural Information Processing Systems*, pages 1125–1133, 2011. 30, 40, 41, 42, 49, 51, 54, 56, 57

[68] A. HYVÄRINEN AND E. OJA. **Independent component analysis: algorithms and applications**. *Neural Networks*, **13**(4–5):411–430, 2000. 30, 40, 57

[69] A. HYVÄRINEN, J. KARHUNEN, AND E. OJA. *Independent component analysis*. Wiley Interscience, 2000. 30, 31, 40

[70] Q. V. LE, A. KARPENKO, J. NGIAM, AND A. Y. NG. **ICA with Reconstruction Cost for Efficient Overcomplete Feature Learning**. In *Advances in Neural Information Processing Systems*, pages 1017–1025, 2011. 30, 31, 41, 50, 51, 56, 57

[71] QUOC V. LE, JIQUAN NGIAM, ZHENGHAO CHEN, DANIEL CHIA, PANG WEI KOH, AND ANDREW Y. NG. **Tiled convolutional neural networks**. In *Advances in Neural Information Processing Systems*, 2010. 31

[72] HONGLAK LEE, ROGER GROSSE, RAJESH RANGANATH, AND ANDREW Y. NG. **Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations**. In *Proc. of the International Conference on Machine Learning*, pages 609–616, New York, NY, USA, 2009. ACM. 31

[73] MOHAMMAD NOROUZI, MANI RANJBAR, AND GREG MORI. **Stacks of convolutional Restricted Boltzmann Machines for shift-invariant feature learning**. In *Proc. of the Conference on Computer Vision & Pattern Recognition*, pages 2735–2742. IEEE, 2009. 31

[74] KORAY KAVUKCUOGLU, PIERRE SERMANET, Y LAN BOUREAU, KAROL GREGOR, MICHAEL MATHIEU, AND YANN L. CUN. **Learning Convolutional Feature Hierarchies for Visual Recognition**. In J.D. LAFFERTY, C.K.I. WILLIAMS, J. SHAWE-TAYLOR, R.S. ZEMEL, AND A. CULOTTA, editors, *Advances in Neural Information Processing Systems*, pages 1090–1098. Curran Associates, Inc., 2010. 31

[75] HILTON BRISTOW, ANDERS ERIKSSON, AND SIMON LUCEY. **Fast Convolutional Sparse Coding**. In *Proc. of the Conference on Computer Vision & Pattern Recognition*, pages 391–398. IEEE, 2013. 31

[76] JONATHAN MASCI, UELI MEIER, DAN CIREŞAN, AND JÜRGEN SCHMIDHUBER. **Stacked Convolutional Auto-encoders for Hierarchical Feature Extraction**. In *Proc. of the International Conference on Artificial Neural Networks*, pages 52–59, Berlin, Heidelberg, 2011. Springer-Verlag. 31

[77] J. WESTON, F. RATLE, AND R. COLLOBERT. **Deep Learning via Semi-Supervised Embedding**. In *Proc. of the International Conference on Machine Learning*, 2008. 32, 33, 93

[78] GEOFFREY E. HINTON AND RUSLAN R SALAKHUTDINOV. **Using Deep Belief Nets to Learn Covariance Kernels for Gaussian Processes**. In J.C. PLATT, D. KOLLER, Y. SINGER, AND S.T. ROWEIS, editors, *Advances in Neural Information Processing Systems*, pages 1249–1256. Curran Associates, Inc., 2008. 32

[79] HUGO LAROCHELLE AND YOSHUA BENGIO. **Classification Using Discriminative Restricted Boltzmann Machines**. In *Proc. of the International Conference on Machine Learning*, pages 536–543, New York, NY, USA, 2008. ACM. 32, 33

[80] C. GULCEHRE AND Y. BENGIO. **Knowledge Matters: Importance of Prior Information for Optimization**. In *Proc. of the International Conference on Learning Representations*, 2013. 33, 34, 93, 97

[81] L. CHEN-YU, X. SAINING, G. PATRICK, Z. ZHENGYOU, AND T. ZHUOWEN. **Deeply-Supervised Nets**. *CoRR*, **abs/1409.5185**, 2014. 34, 93, 98, 99, 100

[82] Y. BENGIO, J. LOURADOUR, R. COLLOBERT, AND J. WESTON. **Curriculum learning**. In *Proc. of the International Conference on Machine Learning*, pages 41–48. ACM, 2009. 34, 93, 97, 103

[83] M. P. KUMAR, B. PACKER, AND D. KOLLER. **Self-Paced Learning for Latent Variable Models**. In J.D. LAFFERTY, C.K.I. WILLIAMS, J. SHAWE-TAYLOR, R.S. ZEMEL, AND A. CULOTTA, editors, *Advances in Neural Information Processing Systems*, pages 1189–1197. Curran Associates, Inc., 2010. 35

[84] L. JIANG, D. MENG, S. YU, Z. LAN, S. SHAN, AND A. HAUPTMANN. **Self-Paced Learning with Diversity**. In *Advances in Neural Information Processing Systems*, pages 2078–2086. Curran Associates, Inc., 2014. 35

[85] J. SNOEK, H. LAROCHELLE, AND R. P. ADAMS. **Practical Bayesian Optimization of Machine Learning Algorithms**. In *Advances in Neural Information Processing Systems*, pages 2960–2968, 2012. 40

[86] H. LEE, A. BATTLE, R. RAINA, AND A. Y. NG. **Efficient sparse coding algorithms**. In *Advances in Neural Information Processing Systems*, pages 801–808, 2006. 40

[87] D. J. FIELD. **What is the goal of sensory coding?** *Neural Computation*, **6**(4):559–601, 1994. 41

[88] B. WILLMORE AND D. J. TOLHURST. **Characterizing the sparseness of neural codes.** *Network*, **12**(12):255–270, 2001. 41, 55

[89] HAROLD W. KUHN. **The Hungarian Method for the assignment problem**. *Naval Research Logistics Quarterly*, **2**:83–97, 1955. 43

[90] ANIL CHERIYADAT. **Unsupervised Feature Learning for Aerial Scene Classification**. *IEEE Transactions on Geoscience and Remote Sensing*, **52**(1):439–451, 2014. 48, 52, 54, 63, 65, 66

[91] A. KRIZHEVSKY. **Learning Multiple Layers of Features from Tiny Images**. Technical report, University of Toronto, 2009. 48

[92] YI YANG AND SHAWN NEWSAM. **Bag-of-visual-words and Spatial Extensions for Land-use Classification**. In *Proc. of the SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 270–279. ACM, 2010. 48, 65, 66

[93] KIHYUK SOHN AND HONGLAK LEE. **Learning Invariant Representations with Local Transformations**. In *Proc. of the International Conference on Machine Learning*, 2012. 58

[94] S. LIANG. *Quantitative Remote Sensing of Land Surfaces*. John Wiley & Sons, New York, 2004. 60

[95] T. M. LILLESAND, R. W. KIEFER, AND J. CHIPMAN. *Remote Sensing and Image Interpretation*. John Wiley & Sons, New York, 2008. 60

[96] G. SHAW AND D. MANOLAKIS. **Signal Processing for Hyperspectral Image Exploitation**. *Signal Processing Magazine*, **50**:12–16, Jan 2002. 60

[97] G. CAMPS-VALLS, D. TUIA, L. BRUZZONE, AND J. ATLI BENEDIKTSSON. **Advances in Hyperspectral Image Classification: Earth Monitoring with Statistical Learning Methods**. *Signal Processing Magazine, IEEE*, **31**(1):45–54, Jan 2014. 60, 61, 62

[98] G. CAMPS-VALLS, D. TUIA, L. GÓMEZ-CHOVA, S. JIMÉNEZ, AND J. MALO, editors. *Remote Sensing Image Processing*. Morgan & Claypool Publishers, LaPorte, CO, USA, Sept 2011. 61

[99] R.M. WILLETT, M.F. DUARTE, M.A. DAVENPORT, AND R.G. BARANIUK. **Sparsity and Structure in Hyperspectral Imaging : Sensing, Reconstruction, and Target Detection**. *Signal Processing Magazine*, **31**(1):116–126, Jan 2014. 61, 62

[100] W.-K. MA, J.M. BIOUCAS-DIAS, TSUNG-HAN CHAN, N. GILLIS, P. GADER, A.J. PLAZA, A. AMBIKAPATHI, AND CHONG-YUNG CHI. **A Signal Processing Perspective on Hyperspectral Unmixing: Insights from Remote Sensing**. *Signal Processing Magazine*, **31**(1):67–81, Jan 2014. 61, 62

[101] G.R. ARCE, D.J. BRADY, L. CARIN, H. ARGUELLO, AND D.S. KITTLE. **Compressive Coded Aperture Spectral Imaging: An Introduction**. *Signal Processing Magazine*, **31**(1):105–115, Jan 2014. 61, 62

[102] D.P. ROY, M.A. WULDER, T.R. LOVELAND, WOODCOCK C.E., R.G. ALLEN, M.C. ANDERSON, D. HELDER, J.R. IRONS, D.M. JOHNSON, R. KENNEDY, T.A. SCAMBOS, C.B. SCHAAF, J.R. SCHOTT, Y. SHENG, E.F. VERMOTE, A.S. BELWARD, R. BINDSCHADLER, W.B. COHEN, F. GAO, J.D. HIPPLE, P. HOSTERT, J. HUNTINGTON, C.O. JUSTICE, A. KILIC, V. KOVALSKYY, Z.P. LEE, L. LYMBURNER, J.G. MASEK, J. MCCORKEL, Y. SHUAI, R. TREZZA, J. VOGELMANN, R.H. WYNNE, AND Z. ZHU. **Landsat-8: Science and product vision for terrestrial global change research**. *Remote Sensing of Environment*, **145**(0):154 – 172, 2014. 61

[103] N. LONGBOTHAM, F. PACIFICI, B. BAUGH, AND G. CAMPS-VALLS. **Prelaunch assessment of WorldView-3 information content**. In *IEEE GRSS Workshop on Hyperspectral Image and Signal Processing (WHISPERS)*, pages 479–486, Lausanne, Switzerland, 2014. 61

[104] M. DRUSCH, U. DEL BELLO, S. CARLIER, O. COLIN, V. FERNANDEZ, F. GASCON, B. HOERSCH, C. ISOLA, P. LABERINTI, P. MARTIMORT, A. MEYGRET, F. SPOTO, O. SY, F. MARCHESE, AND P. BARGELLINI. **Sentinel-2: ESA's Optical High-Resolution Mission for GMES Operational Services**. *Remote Sensing of Environment*, **120**:25–36, 2012. 61

[105] C. DONLON, B. BERRUTI, A. BUONGIORNO, M.-H. FERREIRA, P. FÉMÉNIAS, J. FRERICK, P. GORYL, U. KLEIN, H. LAUR, C. MAVROCORDATOS, J. NIEKE, H. REBHAN, B. SEITZ, J. STROEDE, AND R. SCIARRA. **The Global Monitoring for Environment and Security (GMES) Sentinel-3 mission**. *Remote Sensing of Environment*, **120**:37–57, 2012. 61

[106] T. STUFFLER, C. KAUFMANN, S. HOFER, K.P. FARSTER, G. SCHREIER, A. MUELLER, A. ECKARDT, H. BACH, B. PENNÉ, U. BENZ, AND R. HAYDN. **The EnMAP hyperspectral imager-An advanced optical payload for future applications in Earth observation programmes**. *Acta Astronautica*, **61**(1–6):115–120, 2007. 61

[107] D.A. ROBERTS, D.A. QUATTROCHI, G.C. HULLEY, S.J. HOOK, AND R.O. GREEN. **Synergies between VSWIR and TIR data for the urban environment: An evaluation of the potential for the Hyperspectral Infrared Imager (HyspIRI) Decadal Survey mission**. *Remote Sensing of Environment*, **117**:83–101, 2012. 61

[108] S. KRAFT, U. DEL BELLO, M. BOUVET, M. DRUSCH, AND J. MORENO. **FLEX: ESA's Earth Explorer 8 candidate mission**. In *2012 IEEE International Geoscience and Remote Sensing Symposium*, pages 7125–7128. IEEE, 2012. 61

[109] G. CAMPS-VALLS AND L. BRUZZONE. **Kernel-based methods for hyperspectral image classification**. *IEEE Transactions on Geoscience and Remote Sensing*, **43**(6):1351–1362, June 2005. 61

[110] G. CAMPS-VALLS AND L. BRUZZONE, editors. *Kernel methods for Remote Sensing Data Analysis*. Wiley & Sons, UK, Dec 2009. 61

[111] YOSHUA BENGIO, PASCAL LAMBLIN, DAN POPOVICI, AND HUGO LAROCHELLE. **Greedy layer-wise training of deep networks**. In *Advances in Neural Information Processing Systems*, pages 153–160, 2006. 61

[112] I.T. JOLLIFFE. *Principal component analysis*. Springer, 2002. 62

[113] J. A. LEE AND M. VERLEYSEN. *Nonlinear dimensionality reduction*. Springer, 2007. 62

[114] C.M. BACHMANN, T.L. AINSWORTH, AND R.A. FUSINA. **Improved Manifold Coordinate Representations of Large-Scale Hyperspectral Scenes**. *IEEE Trans. Geosci. Remote Sens.*, **44**(10):2786–2803, 2006. 62

[115] J. ARENAS-GARCÍA, K. B. PETERSEN, G. CAMPS-VALLS, AND L. K. HANSEN. **Kernel Multivariate Analysis Framework for Supervised Subspace Learning: A Tutorial on Linear and Kernel Multivariate Methods**. *IEEE Sig. Proc. Mag.*, **30**(4):16–29, 2013. 62

[116] G. E. HINTON AND R. R. SALAKHUTDINOV. **Reducing the Dimensionality of Data with Neural Networks**. *Science*, **313**(5786):504–507, July 2006. 62

[117] V. LAPARRA, G. CAMPS, AND J. MALO. **Iterative Gaussianization: from ICA to Random Rotations**. *IEEE Transactions on Neural Networks*, **22**(4):537–549, 2011. 62

[118] A. BARALDI AND F. PARMIGGIANI. **A neural network for unsupervised categorization of multivalued input patterns: an application to satellite image clustering**. *IEEE Transactions on Geoscience and Remote Sensing*, **33**(2):305–316, Mar 1995. 62

[119] S. GHOSH, L. BRUZZONE, S. PATRA, F. BOVOLO, AND A. GHOSH. **A Context-Sensitive Technique for Unsupervised Change Detection Based on Hopfield-Type Neural Networks**. *IEEE Transactions on Geoscience and Remote Sensing*, **45**(3):778–789, March 2007. 62

[120] F. DEL FRATE, G. LICCIARDI, AND R. DUCA. **Autoassociative neural networks for features reduction of hyperspectral data**. In *Hyperspectral Image and Signal Processing: Evolution in Remote Sensing, 2009. WHISPERS '09. First Workshop on*, pages 1–4, Aug 2009. 62

[121] G. LICCIARDI, F. DEL FRATE, AND R. DUCA. **Feature reduction of hyperspectral data using Autoassociative neural networks algorithms**. In *Geoscience and Remote Sensing Symposium,2009 IEEE International,IGARSS 2009*, **1**, pages I–176–I–179, July 2009. 62

[122] Z. WANG, N.M. NASRABADI, AND T.S. HUANG. **Spatial-Spectral Classification of Hyperspectral Images Using Discriminative Dictionary Designed by Learning Vector Quantization**. *IEEE Transactions on Geoscience and Remote Sensing*, **PP**(99):1–15, 2013. 63

[123] S. YANG, H. JIN, M. WANG, Y. REN, AND L. JIAO. **Data-Driven Compressive Sampling and Learning Sparse Coding for Hyperspectral Image Classification**. *IEEE Geoscience and Remote Sensing Letters*, **11**(2):479–483, Feb 2014. 63

[124] S. LI, H. YIN, AND L. FANG. **Remote Sensing Image Fusion via Sparse Representations Over Learned Dictionaries**. *IEEE Transactions on Geoscience and Remote Sensing*, **51**(9):4779–4789, Sept 2013. 63

[125] D. DAI AND W. YANG. **Satellite Image Classification via Two-Layer Sparse Coding With Biased Image Representation**. *IEEE Geoscience and Remote Sensing Letters*, **8**(1):173–176, Jan 2011. 63

[126] I. RIGAS, G. ECONOMOU, AND S. FOTOPOULOS. **Low-Level Visual Saliency With Application on Aerial Imagery**. *IEEE Geoscience and Remote Sensing Letters*, **10**(6):1389–1393, Nov 2013. 63

[127] H. SUN, X. SUN, H. WANG, Y. LI, AND X. LI. **Automatic Target Detection in High-Resolution Remote Sensing Images Using Spatial Sparse Coding Bag-of-Words Model**. *IEEE Geosc. Rem. Sens. Lett.*, **9**(1):109–113, Jan 2012. 63

[128] C. VADUVA, I. GAVAT, AND M. DATCU. **Deep learning in very high resolution remote sensing image information mining communication concept**. In *Proc. of the European Signal Processing Conference (EUSIPCO)*, pages 2506–2510, Aug 2012. 63

[129] VOLODYMYR MNIH AND GEOFFREY HINTON. **Learning to Label Aerial Images from Noisy Data**, 2012. 63

[130] X. CHEN, S: XIANG, C.-L. LIU, AND C.-H. PAN. **Vehicle Detection in Satellite Images by Hybrid Deep Convolutional Neural Networks**. *IEEE Geoscience and Remote Sensing Letters*, **11**(10):1797–1801, 2014. 63

[131] Y. CHEN, Z. LIN, X. ZHAO, G. WANG, AND Y. GU. **Deep learning-based classification of hypersepctral data**. *Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, **7**:2094–2107, 2014. 63

[132] A. C. HILL, T. S. BANSAK, B. K. ELLIS, AND J. A. STANFORD. **Merits and limits of ecosystem protection for conserving wild salmon in a northern coastal British Columbia river**. *Ecology and Society*, **15**(2):20, 2010. 73

[133] D. C. WHITED, J. S. KIMBALL, M.S. LORANG, AND J. STANFORD. **Estimation of juvenile salmon habitat in Pacific rim rivers using multiscalar remote sensing and geospatial analysis**. *River Research and Applications*, **48**(1):207–220, 2011. 73

[134] DAVID GRANGIER, LÉON BOTTOU, AND RONAN COLLOBERT. **Deep Convolutional Networks for Scene Parsing**. Workshop on Learning Feature Hierarchies, ICML, June 2009. 79

[135] CLEMENT FARABET, CAMILLE COUPRIE, LAURENT NAJMAN, AND YANN LECUN. **Learning Hierarchical Features for Scene Labeling**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **35**(8):1915–1929, 2013. 79, 80, 83, 89

[136] PEDRO PINHEIRO AND RONAN COLLOBERT. **Recurrent Convolutional Neural Networks for Scene Labeling**. *Journal of Machine Learning Research*, **1**(32):82–90, 2014. 79, 80, 82, 89

[137] GAUTAM SINGH AND JANA KOSECKA. **Nonparametric Scene Parsing with Adaptive Feature Relevance and Semantic Context.** In *Proc. of the Conference on Computer Vision & Pattern Recognition*, pages 3151–3157. IEEE, 2013. 79, 85, 87, 88

[138] P. SERMANET, K. KAVUKCUOGLU, S. CHINTALA, AND Y. LECUN. **Pedestrian Detection with Unsupervised Multi-Stage Feature Learning**. In *Proc. of the Conference on Computer Vision & Pattern Recognition*. IEEE, 2013. 80

[139] CE LIU, JENNY YUEN, AND ANTONIO TORRALBA. **Nonparametric Scene Parsing via Label Transfer**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **33**(12):2368–2382, 2011. 82

[140] O. RUSSAKOVSKY, J. DENG, H. SU, J. KRAUSE, S. SATHEESH, S. MA, Z. HUANG, A. KARPATHY, A. KHOSLA, M. BERNSTEIN, A. C. BERG, AND L. FEI-FEI. **ImageNet Large Scale Visual Recognition Challenge**. *CoRR*, **abs/1409.0575**, 2014. 92, 101

[141] C. BUCILA, R. CARUANA, AND A. NICULESCU-MIZIL. **Model compression**. In *Proc. of the Conference on Knowledge Discovery and Data Mining*, pages 535–541, 2006. 92

[142] J. BA AND R. CARUANA. **Do Deep Nets Really Need to be Deep?** In *Advances in Neural Information Processing Systems*, pages 2654–2662. Curran Associates, Inc., 2014. 92, 99

[143] O. HINTON, G. VINYALS AND J. DEAN. **Distilling Knowledge in a Neural Network**. In *Deep Learning & Representation Learning Workshop, NIPS*, 2014. 92, 93, 94, 108, 109

[144] G.F. MONTUFAR, R. PASCANU, K. CHO, AND Y. BENGIO. **On the Number of Linear Regions of Deep Neural Networks**. In *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2014. 92, 108

[145] YOSHUA BENGIO. **Deep Learning of Representations: Looking Forward**. *CoRR*, **abs/1305.0445**, 2013. 93

[146] A. KRIZHEVSKY AND G. HINTON. **Learning multiple layers of features from tiny images**. *Master's thesis, Department of Computer Science, University of Toronto*, 2009. 98, 102

[147] Y. NETZER, T. WANG, A. COATES, A. BISSACCO, B. WU, AND A. NG. **Reading digits in natural images with unsupervised feature learning**. In *Deep Learning & Unsupervised Feature Learning Workshop, NIPS*, 2011. 100

[148] Y. LECUN, L. BOTTOU, Y. BENGIO, AND P. HAFFNER. **Gradient-Based Learning Applied to Document Recognition**. *Proceedings of the IEEE*, **86**(11):2278–2324, November 1998. 101

[149] M. KOESTINGER, P. WOHLHART, P.M. ROTH, AND H. BISCHOF. **Annotated Facial Landmarks in the Wild: A Large-scale, Real-world Database for Facial Landmark Localization**. In *IEEE International Workshop on Benchmarking Facial Image Analysis Technologies*, 2011. 101

[150] Y. DAUPHIN, R. PASCANU, C. GULCEHRE, K. CHO, S. GANGULI, AND Y. BENGIO. **Identifying and attacking the saddle point problem in high-dimensional non-convex optimization**. In *Advances in Neural Information Processing Systems*, 2014. 103

[151] M. JADERBERG, A. VEDALDI, AND A. ZISSERMAN. **Speeding up Convolutional Neural Networks with Low Rank Expansions**. In *Proc. of the British Machine Vision Conference*, 2014. 105

[152] EMILY L DENTON, WOJCIECH ZAREMBA, JOAN BRUNA, YANN LECUN, AND ROB FERGUS. **Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation**. In *Advances in Neural Information Processing Systems*, pages 1269–1277. Curran Associates, Inc., 2014. 105

[153] YONGJIAN CHEN, TAO GUAN, AND CHENG WANG. **Approximate Nearest Neighbor Search by Residual Vector Quantization**. *Sensors*, **10**(12):11259–11273, 2010. 105

[154] HERVÉ JÉGOU, MATTHIJS DOUZE, AND CORDELIA SCHMID. **Product quantization for nearest neighbor search**. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **33**(1):117–128, 2011. 105

[155] YUNCHAO GONG, LIU LIU, MIN YANG, AND LUBOMIR BOURDEV. **Compressing Deep Convolutional Networks using Vector Quantization**. *CoRR*, **abs/1412.6115**, 2014. 105

[156] J. HÅSTAD AND M. GOLDMANN. **On the power of small-depth threshold circuits**. *Computational Complexity*, 1991. 108

[157] DONG WANG, CHAO LIU, ZHIYUAN TANG, ZHIYONG ZHANG, AND MENGYUAN ZHAO. **Recurrent Neural Network Training with Dark Knowledge Transfer**. *CoRR*, **abs/1505.04630**, 2015. 109

[158] ANOOP KORATTIKARA, VIVEK RATHOD, KEVIN MURPHY, AND MAX WELLING. **Bayesian Dark Knowledge**. *CoRR*, **abs/1506.04416**, 2015. 109

[159] DAVID HERYANTO AND TAT-SENG CHUA. **Incremental Training of Neural Network with Knowledge Distillation**. *B.Comp Dissertation, Department of Computer Science, National University of Singapore*, 2015. 109

[160] VADIM LEBEDEV AND VICTOR LEMPITSKY. **Fast ConvNets Using Group-wise Brain Damage**. *CoRR*, **abs/1506.02515**, 2015. 109

[161] XIANGYU ZHANG, JIANHUA ZOU, KAIMING HE, AND JIAN SUN. **Accelerating Very Deep Convolutional Networks for Classification and Detection**. *CoRR*, **abs/1505.06798**, 2015. 109

[162] RUPESH KUMAR SRIVASTAVA, KLAUS GREFF, AND JURGEN SCHMIDHUBER. **Highway Networks**. *CoRR*, **abs/1505.00387**, 2015. 109

[163] K. CHATFIELD, K. SIMONYAN, A. VEDALDI, AND A. ZISSERMAN. **Return of the Devil in the Details: Delving Deep into Convolutional Nets**. In *Proc. of the British Machine Vision Conference*, 2014. 110

[164] ARAVINDH MAHENDRAN AND ANDREA VEDALDI. **Understanding Deep Image Representations by Inverting Them**. In *Proc. of the Conference on Computer Vision & Pattern Recognition*, 2015. 110