**University of Alberta**

**Library Release Form**

**Name of Author**: Ilya Levner

**Title of Thesis**: Data Driven Object Segmentation

**Degree**: Doctor of Philosophy

**Year this Degree Granted**: 2009



_____

Ilya Levner

**Date**: _____

*If you torture the data long enough,*
*eventually it will confess to anything*

–Anonymous

**University of Alberta**

# DATA DRIVEN OBJECT SEGMENTATION

by

**Ilya Levner**

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**

Department of Computer Science

Edmonton, Alberta
Spring 2009

**University of Alberta**

**Faculty of Graduate Studies and Research**

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Data Driven Object Segmentation** submitted by Ilya Levner in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

_____

Dr. Russell Greiner
Co-Supervisor

_____

Dr. Hong Zhang
Co-Supervisor

_____

Dr. Walter Bischof

_____

Dr. Nilanjan Ray

_____

Dr. Mrinal Mandal

_____

Dr. Yoshua Benjio
External Examiner

**Date**: _____

# Abstract

In recent decades, learning has emerged as the quintessential component of both biological and artificial vision systems. This thesis develops machine learning methods to address the problem of object segmentation, whereby the goal is to separate objects of a specific class from the background and each other. To solve this task we propose the Data Driven Region Growing framework, which utilizes three machine learned mappings to identify: (a) foreground regions, (b) object boundaries, and (c) object markers/seeds. The three pixel maps are subsequently used within the region growing framework to delineate individual target objects. The second part of the thesis focuses on machine learning approaches for automated feature extraction and regression methods used to learn the three aforementioned mappings. In contrast to classical algorithms, which attempt to label individual pixels, we argue that algorithms should utilize higher order reasoning and label whole image neighborhoods. To demonstrate the concept, we present several algorithms based on decomposing the ground truth in a manner analogous to feature extraction. Using greedy layer-wise learning approach, a deep convolutional neural network can be trained to automatically extract relevant input image features and utilize them to produce output patches in a fully automated manner. By using deep neural networks utilizing the concept of output decomposition and in conjunction with region growing methods, a state-of-the-art object segmentation system can be constructed with minimal input from software engineers and domain experts.

# Acknowledgements

Thesis writing is inherently a solitary endeavor. Yet I cannot help remembering the numerous people who have helped, guided and supported me throughout these years. First and foremost, this thesis would not have been possible without the guidance and support of my supervisors, Russell Greiner and Hong Zhang, who have patiently helped me shape my critical thinking and research skills. Walter Bischof introduced me to the fascinating area of computational neuroscience, which has never stopped being a source of inspiration for my work. I owe a loving debt of gratitude to my wife Ekaterina Levner. Her patience and support has made graduate studies an incredible journey. Watching my children, Andrew and Victoria, grow has been the joy of my life and a great inspiration for my work. I would also like to thank my parents for endowing me with a solid foundation as well as their unwavering belief in my abilities. Deepest thanks go out to Mark Pollack and David Laing for helping us run OSA. Likewise, we are grateful to Yury Potapovich for helping us run the WipFrag system. We are grateful to Li Cheng and Terry Caelli for providing the Forestry Images and the corresponding ground truth. This research is supported in part by National Science Research Council, Alberta Ingenuity Fund, iCORE, Killam Fund, and the University of Alberta.

# Contents


**1 Introduction** **1**
- 1.1 Problem Formulation . . . 2
- 1.2 Manually Coded Systems . . . 2
- 1.3 Machine Learned Systems . . . 4
  - 1.3.1 Feature Extraction . . . 6
- 1.4 Data Driven Region Growing . . . 7
- 1.5 Thesis Statement . . . 7
- 1.6 Thesis Outline . . . 7

**2 Data Driven Region Growing** **9**
- 2.1 Digital Image Representation . . . 10
  - 2.1.1 Graph Theory . . . 10
  - 2.1.2 Regular/Digital Lattices . . . 10
- 2.2 Image Segmentation . . . 12
  - 2.2.1 Edge Based Methods . . . 12
  - 2.2.2 Thresholding and Histogram Based Methods . . . 13
  - 2.2.3 Region Based Methods . . . 14
  - 2.2.4 General Region Growing . . . 16
- 2.3 Object Segmentation . . . 19
  - 2.3.1 Data Driven Region Growing . . . 21

**3 Learning to Label** **27**
- 3.1 Pixel Classification . . . 28
- 3.2 Learning Classifiers . . . 30
- 3.3 Neural Networks . . . 31
  - 3.3.1 Regularization Techniques . . . 33
  - 3.3.2 Cascade Correlation . . . 34
- 3.4 Model Fusion . . . 35
  - 3.4.1 Generalized Pixel Labeling . . . 35
  - 3.4.2 Stacked Generalization . . . 36
  - 3.4.3 Heterogeneous Stacking . . . 36

3.5 Random Field Methods . . . . . 39

**4 Automated Feature Extraction and Relevant Applications 41**
4.1 Neighborhood Analysis . . . . . 42
4.2 Linear Algebra, SVD and Eigenvalue Decomposition . . . . . 42
4.3 Principal Component Analysis (PCA) . . . . . 43
4.3.1 Linear Autoencoders . . . . . 44
4.4 Independent Component Analysis (ICA) . . . . . 45
4.4.1 Convolutional Encoding/Decoding of Images . . . . . 48
4.4.2 ICA based Neural Networks . . . . . 50
4.4.3 Sparse Code Shrinkage for (Image) Denoising . . . . . 51
4.4.4 ICA for Feature Extraction from Color and Stereo Images . 53
4.4.5 ICA for Regression . . . . . 53
4.5 Autoencoders and Non-Linear PCA . . . . . 55
4.5.1 Diabolo Networks - Discriminative Autoencoders . . . . . 57
4.5.2 Deep Greedy Layer-wise Learning . . . . . 58
4.6 Convolutional Networks . . . . . 61
4.6.1 Neocognitron . . . . . 62
4.6.2 LeNet . . . . . 64
4.7 High Order Random Fields for image analysis and denoising . . . . . 66

**5 From Pixels To Patches To Structures 69**
5.1 Output Decomposition . . . . . 70
5.1.1 From pixel labeling to structure labeling . . . . . 70
5.1.2 Output Decomposition based Mixture-of-Experts . . . . . 71
5.2 A Unifying Perspective for Generative Learning . . . . . 75
5.3 A Unified View for Discriminative Learning . . . . . 77
5.4 Stacked Convolutional Regression Networks . . . . . 80
5.4.1 Network Growth and Training . . . . . 80
5.4.2 Online Inference . . . . . 82

**6 Experimental Results 84**
6.1 Classification Driven Watershed Segmentation . . . . . 85
6.1.1 Feature Extraction . . . . . 85
6.1.2 Experimental Procedure . . . . . 88
6.1.3 Experiment 1 . . . . . 88
6.1.4 Experiment 2 . . . . . 89
6.1.5 Experiment 3 - Color Image Segmentation . . . . . 92
6.2 Heterogeneous Stacking and ICA for Classification Driven Watershed Segmentation, [76] . . . . . 94

6.2.1 Experimental Procedure . . . . . 94
6.2.2 Experiment 1 . . . . . 95
6.2.3 Experiment 2 . . . . . 97
6.3 Output Decomposition Mixture of Experts . . . . . 100
6.3.1 Results . . . . . 100
6.4 Stacked Convolutional Regression (SCR) . . . . . 103
6.4.1 Experiment 1 . . . . . 103
6.4.2 Experiment 2 . . . . . 108
6.4.3 Experiment 3 . . . . . 111
6.4.4 Large Scale Experiments . . . . . 113
6.4.5 Aerial Forest Images Revisited . . . . . 115
6.4.6 Post Processing Revisited . . . . . 117
6.5 Discussion . . . . . 118

**7 Conclusion** **120**
7.1 Summary . . . . . 121
7.2 Contributions . . . . . 122
7.3 Related and Future Research Directions . . . . . 122
7.3.1 Convolutional Networks . . . . . 123
7.3.2 Adaptive Processing . . . . . 124
7.4 Final Thoughts . . . . . 124

**A Oil Sand Ore Granulometry** **133**

**B Forest Inventory Building (from [73])** **135**
B.1 Special Purpose Forestry Systems . . . . . 135
B.2 Machine Learning Approaches . . . . . 136

**C Evaluation Criteria** **138**

# Chapter 1

# Introduction

## 1.1 Problem Formulation

Computer based **image understanding** (IU) is commonly defined as the automation of a visual task typically done by a human expert. Given an input image, the goal is to produce an image interpretation with respect to a specific task. In turn, each specific task forms what is called a domain. This research explores a particular subtask within IU, namely object segmentation. Given an input image $\boldsymbol{I}$, we aim to label pixels corresponding to objects of interest and additionally, we aim to coalesce individual pixels into connected components corresponding to individual objects within a given image. Figure 1.1 on p. 3 presents input, $\boldsymbol{I}$, and desired output, $\boldsymbol{L}$ images for two different domains. To simplify matters we will concern ourselves with binary interpretations where a given pixel $\boldsymbol{I}(i, j)$ either belongs to a target object, $\boldsymbol{L}(i, j) = 1$, or is part of the background $\boldsymbol{L}(i, j) = 0$. Once pixels have been labeled they can be grouped (or clustered) into individual objects. From there on, parameters of interest can be extracted from individual objects. For example, within the forestry domain, groups of individual pixels represent tree canopies. Once tree canopies have been identified, we can use the canopy shape and area to automatically infer the width, height, age, and wood volume of the specific tree. Clearly, without a good initial object segmentation, tree parameters extracted from erroneous canopy groupings will not approximate actual (physically measured) tree parameters.

The primary focus of this thesis is therefore, the development of a fully automated **learning** framework capable of producing object segmentation systems that accurately separate objects of interest from the background and each other, in a wide range of domains.

## 1.2 Manually Coded Systems

In general, image understanding can be approached in several ways. The traditional way of building an image interpretation system is through manual design and development, whereby for each domain of interest a team of domain and computer vision experts hand-code a system to produce the necessary image labeling, pixel grouping and parameter extraction algorithms. Unfortunately, the last three decades of research produced few successful hand-engineered systems capable of working in non-trivial domains. The problem is due to the vast range of conditions present within input images. For example, in the domain of forestry, image variations result from variable sun and camera angles, seasonal changes, weather conditions, and many other uncontrollable factors. These image variations require that the image interpretation algorithms be extremely flexible and robust. To accomplish this feat, the manual engineering process must explicitly take into account all possi-

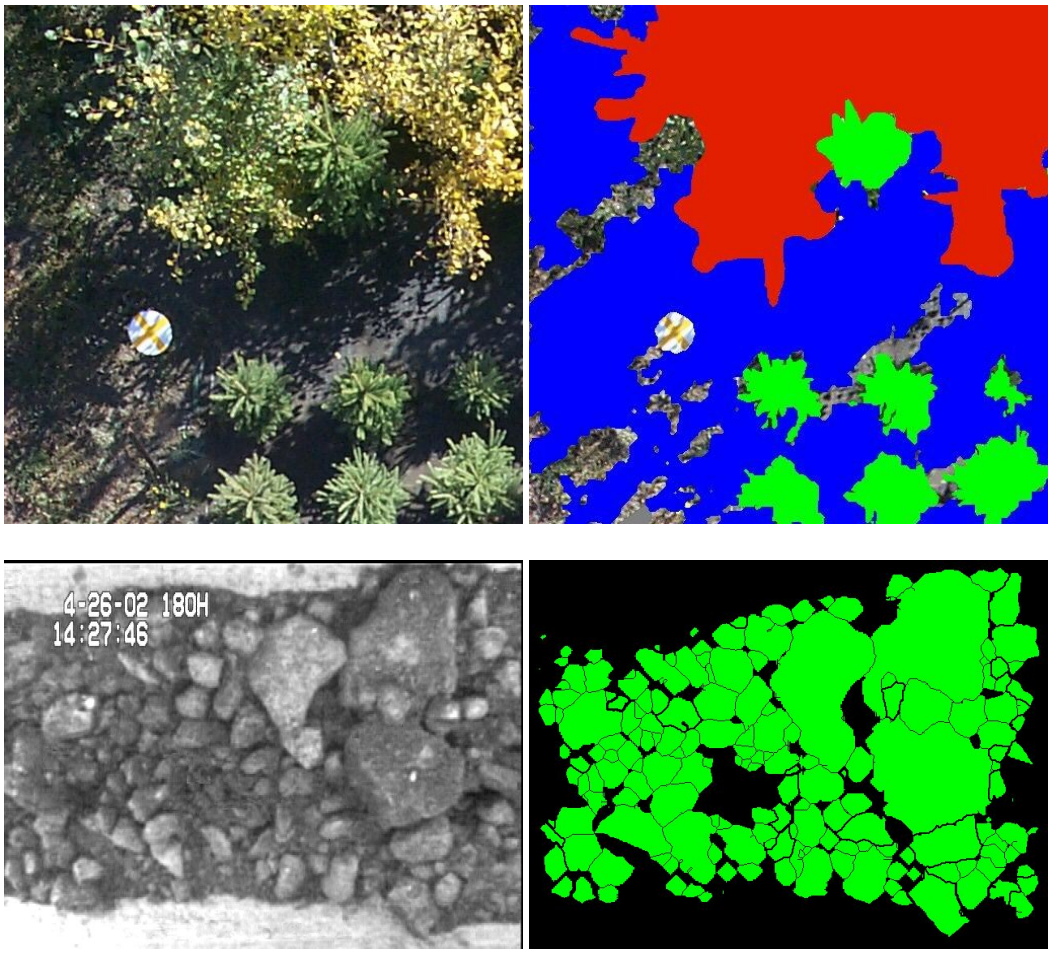



Figure 1.1: Examples of basic image interpretation. Each row corresponds to a different domain, with their own objects of interest. **Left:** Input Images **Right:** Desired pixel labeling, where green (see the pdf color version of this document) labels denote pixels belonging to objects of interest. I.e., $\boldsymbol{L}(i,j) = (0, 255, 0)$ instead of 1. **Top:** Color image of a forest plantation. The desired objects of interest are the spruce trees (green), aspen(red), and shadow (blue). Appendix B provides an overview of this domain. **Bottom:** Ore conveyor belt. The desired objects of interest are lumps of ore bigger that a specific size. Appendix A provides an overview of this domain.

ble variations that may be encountered during field operation [42, 85, 28]. This is a painstaking, tedious, and almost always heuristically guided process, which can take months or even years (see [27] for an in-depth discussion). To circumvent this problem, research has turned towards machine learning techniques (e.g., [80, 25]) in order to automate the system development process and (ideally) to improve online performance.

## 1.3 Machine Learned Systems

This thesis primarily focuses on *supervised* machine learning approaches for pixel labeling (and the subsequent grouping of pixels into objects). Such approaches require a training set composed of images and corresponding pixel labels. We denote this training test as $D = \{(\boldsymbol{I}_k, \boldsymbol{L}_k)\}_{k=1}^{n}$, where $\boldsymbol{I}$ is the input image, $\boldsymbol{L}$ is the corresponding image labeling with $\boldsymbol{L}(i,j) \in \{0,1\}$ being the label for a pixel $\boldsymbol{I}(i,j) \in [0,1]$, and $k$ is the number of training pairs $(\boldsymbol{I}, \boldsymbol{L})$.

The simplest approach is to use the training data $D$ in an attempt to induce (i.e., learn) a pixel labeling function[1] $h : \boldsymbol{I}(i,j) \mapsto \boldsymbol{L}(i,j)$, which takes as input an image pixel (a scalar for grayscale and binary images or a vector in the case of multi-spectral images). Unfortunately, this approach implicitly assumes that pixels are independent of one another. To the contrary, in most domains of interest, including those depicted in Figure 1.1, pixels exhibit local interactions with neighboring pixels. To generalize the pixel level machine learning approach to this, more realistic setting more than single pixel attributes need to be examined.

To improve the situation, a feature extraction function, $\boldsymbol{f}^{\boldsymbol{I}}(i,j)$, is typically utilized, that provides relevant information about pixel $\boldsymbol{I}(i,j)$. This function can, for example extract: intensity value information from neighboring pixels, multi-spectral and/or multi-resolution information, and/or other information relevant to classifying a pixel $\boldsymbol{I}(i,j)$ as $\boldsymbol{L}(i,j)$. Once the feature extraction function $\boldsymbol{f}(\cdot)$ has been specified, the goal is, therefore, as follows. Given a training set $D$, a machine learning algorithm $\mathfrak{A}$, and a feature extraction function $\boldsymbol{f}$, learn a mapping $h$:

$$h : \boldsymbol{f}^{\boldsymbol{I}}(i,j) \overset{\mathfrak{A}\{D\}}{\mapsto} \boldsymbol{L}(i,j) \tag{1.1}$$

induced by the learning algorithm $\mathfrak{A}$ over the training set $D$.

[1] In this dissertation we modify the typical notation for function definition

$$\langle function\ Name \rangle : \langle Domain \rangle \mapsto \langle Range \rangle$$

to denote the name of the model, the input type and producing the output type:

$$\langle model\ Name \rangle : \langle input\ Type \rangle \mapsto \langle output\ Type \rangle$$

.

Another way to produce $\boldsymbol{L}$, is to attempt learning a direct mapping $h : \boldsymbol{I} \mapsto \boldsymbol{L}$. That is, the input to the machine learning algorithm would be the entire image $\boldsymbol{I}$ and the desired output would be the entire image labeling $\boldsymbol{L}$. Unfortunately, images are composed of thousands or even millions of pixels. The large dimensionality of the input (and the output) poses a problem to most modern day machine learning algorithms since a very large number of internal parameters must be estimated from the data. In addition, a binary label image, $\boldsymbol{L}$, can take on $2^{|N|*|M|}$ unique configurations (i.e., the number of states $\boldsymbol{L}$ can take, grows exponentially in the size of the image), where $|N|, |M|$ are the dimensions of $\boldsymbol{I}$ and $\boldsymbol{L}$. Despite this challenge, research into Markov/Conditional/Disciminative Random Field methods [18, 65, 64] has produced a number of heuristic approaches for maximizing $p[\boldsymbol{L}|\boldsymbol{I}]$. These are discussed in Chapters 3 and 4.

To briefly compare the different approaches, observe that classical pixel labeling attempts to find the following mapping:

$$h_{pl} : \boldsymbol{I}(i,j) \mapsto \boldsymbol{L}(i,j) \tag{1.2}$$

by computing the probability of pixel $\boldsymbol{I}(i,j)$ belonging to the target class. Equation 1.2 treats individual pixels as i.i.d., an assumption rarely satisfied in practice, since most non-trivial domains exhibit complex pixel interactions. To overcome this problem, contextual pixel labeling defines a feature extraction function, $\boldsymbol{f}^{\boldsymbol{I}}(i,j)$, for each image pixel $\boldsymbol{I}(i,j)$. Subsequently the newly formed feature vectors are used to learn the mapping:

$$h_{cpl} : \boldsymbol{f}^{\boldsymbol{I}}(i,j) \mapsto \boldsymbol{L}(i,j) \tag{1.3}$$

To further improve pixel classification accuracy, recursive contextual pixel classification [129] and, more recently, random field methods (e.g., Markov / Conditional / Discriminative Random Fields [65, 64]) have been designed to account for label interactions as well as input pixel interactions. These systems first use the regular contextual pixel labeling, as in Equation 1.3, to produce an initial labeling $\boldsymbol{L}_0$. Subsequently a recursive procedure iteratively computes $\boldsymbol{L}_d$ as follows:

$$h_{rcpl} : [\boldsymbol{f}^{\boldsymbol{I}}(i,j), \boldsymbol{f}^{\boldsymbol{L}_{d-1}}(i,j)] \mapsto \boldsymbol{L}_d(i,j) \tag{1.4}$$

where $\boldsymbol{f}^{\boldsymbol{L}_{d-1}}(i,j)$ is a function extracting features from $\boldsymbol{L}_{d-1}$ at lattice site $(i,j)$. Typically the features extracted from $\boldsymbol{L}$ are very simple, usually just a neighborhood centered about $(i,j)$ (or cliques within the MRF framework).

In contrast to the outlined approaches, we proposes a different method. The Output Decomposition Mixture of Experts (OD-MoE) and Stacked Convolutional Regression (SCR) algorithms described in Chapter 5, extract contextual features

from **both** input images and ground truth images, and subsequently learn a mapping from the former to the latter:

$$h_{\text{OD-MoE}} : \boldsymbol{f^I}(i,j) \mapsto \boldsymbol{f^L}(i,j) \tag{1.5}$$

The essence of the OD-MoE algorithm lies in extracting output features, $\boldsymbol{f^L}$, that allow the synthesis of output $\boldsymbol{L}$. Once the input/output decomposition scheme has extracted the input and output features, machine learning algorithms are utilized to train models that instantiate the mapping in Equation 1.5.

### 1.3.1 Feature Extraction

Several options exist for defining the feature extraction functions, $\boldsymbol{f^I}(i,j)$ and $\boldsymbol{f^L}(i,j)$. Once again the typical approach, in image processing and many other domains utilizing machine learning, involves a domain expert who painstakingly hand-crafts a set of feature extraction routines comprising $\boldsymbol{f^I}$. This is a tedious and time consuming process, but less so than manually building the whole system. Furthermore, there are practically an infinite number of features one can extract from an image. For instance even for a small $3 \times 3$ feature extraction filter restricted to eight grayscale values, there are $8^9 = 2^{27} = 134,217,728$ configurations. Furthermore, even if a domain expert can succeed in finding the "right" features, these hand-crafted features will most likely be highly domain dependent, and hence, for a different domain the feature extraction process will need to be repeated. One way to circumvent this problem is to code a very large number of features in the hopes that a subset of the features will be applicable to the problem at hand. One can then employ a wealth of feature selection methods, roughly divided into filter, wrapper and embedded methods [41], to select a subset of relevant features for a given domain (see [11] for a system utilizing a combination of the aforementioned algorithms).

An alternative way to bypass the need for hand-crafting features is to employ automated or data-driven feature extraction techniques that can directly extract relevant features from the raw data. The most widely used technique for automated feature extraction typically employ principal component analysis (PCA), or independent component analysis (ICA). In contrast, Chapter 5 presents a specialized Convolutional Neural Network specifically designed for image segmentation. Motivated by the Output Decomposition Mixture of Experts philosophy, the proposed Stacked Convolutional Regression method is able to simultaneously extract features from both input images and output labels and constructs a regression function capable of mapping input features to the desired output patches.

## 1.4 Data Driven Region Growing

Despite the capability of Stacked Convolutional Regression to automatically extract relevant features and be highly accurate, even a single pixel error can merge together two objects into a single connected component. To further improve the quality of object segmentation, the Data Driven Region Growing framework (DDRG), is developed in Chapter 2. The framework, based on the Classification Driven Watershed algorithm [75], consists of three machine learned mappings for identifying: (i) object markers, (ii) foreground-background regions and (iii) object boundaries. These three components are subsequently combined and utilized by a region growing algorithm (morphological watershed) to improve the separation of individual object from each other. The need to learn three mappings within the DDRG framework further necessitates the need for automated feature extraction and relies on the SCR algorithm for automatically learning the three aforementioned mappings.

## 1.5 Thesis Statement

The main thesis developed in this dissertation is that there are significant advantages to manipulating and/or decomposing the ground truth image in a manner analogous to feature extraction with respect to object segmentation. As subsequent chapters will demonstrate ground truth manipulation and decomposition notions can be utilized to produce state-of-the-art object segmentation systems.

## 1.6 Thesis Outline

Chapter 2 introduces notation used throughout this dissertation and image processing algorithms related to image and object segmentation. The chapter also presents the first major contribution of this thesis, the framework for object segmentation, called Data Driven Region Growing (DDRG). Chapter 3 presents machine learning algorithms related and used in subsequent chapters. In addition, the chapter presents Heterogeneous Stacking, that has been developed specifically for the DDRG framework. Chapter 4 presents an in-depth look at automated feature extraction methods and develops a unified view of these methods. Chapter 5 presents the other main contributions of this work, namely Output Decomposition Mixture of Experts and Stacked Convolutional Regression. Subsequently, Chapter 6 presents detailed experimental results. Three distinct systems are presented to demonstrate the efficacy of the data driven region growing framework. In addition, results using Output Decomposition and automated feature extraction using convolutional neural networks are presented. The efficacy of the aforementioned systems and algorithms is tested

on two separate domains depicted in Figure 1.1. Chapter 7 concludes the thesis with a discussion of future research directions and final thoughts.

# Chapter 2

# Data Driven Region Growing

The following chapter introduces the notation used throughout the dissertation and reviews the literature on image segmentation methods. In addition, the chapter goes on to define the concept of object segmentation and presents the Data Driven Region Growing framework (DDRG), the first major contribution of this work. When combined with a machine learning method capable of automated feature extraction, such as Stacked Convolutional Regression presented in Chapter 5, the framework can create a highly accurate object segmentation system with minimal input from domain experts.

## 2.1 Digital Image Representation

### 2.1.1 Graph Theory

A graph $G = (V, E)$ consists of vertices $v \in V$, and edges $e \in E \subseteq V \times V$. Edges connect vertices $e = (v_1, v_2)$ and therefore, two vertices are adjacent/connected/neighbors if there exists an edge between them. The neighborhood function $\partial^G(v)$ is the set of all vertices connected to vertex $v$. A path $\pi_{v_1,v_l}$, in graph $G$, is a set of vertices $\pi_{v_1,v_l} = \{v_1, v_2, ..., v_l\}$ that indirectly links vertex $v_l$ and $v_1$. A graph is connected if every vertex is reachable from every other vertex. Intuitively a connected component is a subgraph, $G_1 \subseteq G$ that forms a connected graph. The set of maximally (i.e., the largest) connected components partition a graph, $G = G_1 \cup G_2 \cup ... \cup G_n$.

### 2.1.2 Regular/Digital Lattices

A lattice is a specialized graph (usually) defined on a two dimensional grid with length $M$ and width $N$ corresponding to an $N \times M$ matrix. Formally, we let $(i, j)$ index a discrete set of sites (i.e., vertices $v_{i,j} = (i, j)$) on a spatially regular $N \times M$ lattice, $S$:

$$S = \{(i, j)|1 \leq i \leq N, 1 \leq j \leq M\} \tag{2.1}$$

The main distinguishing characteristic of a regular lattice is the type of connectivity imposed on the graph. Usually either a 4-connectivity or an 8-connectivity is imposed in the graph. Under the 4-connectivity, lattice site (i,j) is connected to sites $(i + 1, j), (i - 1, j), (i, j + 1), (i, j - 1)$, provided they exist. The 8-connectivity, includes the 4-connectivity and the four diagonal elements $(i+1, j+1), (i-1, j-1), (i-1, j+1), (i+1, j-1)$. In order to allow more general connectivity between (non-adjacent) pixels, let $\partial(i, j)$ refer to the set of neighbors of $(i, j)$. The neighborhood system defined by $\partial(i, j)$ is symmetric, $(i', j') \in \partial(i, j) \Rightarrow (i, j) \in \partial(i', j'), (i, j) \notin \partial(i, j)$. A clique $c$ is a fully connected subgraph of the original graph and is defined as a the set of points that are all neighbors of each other, $c = \{(i, j) \in S, (i', j') \in S|(i', j') \in \partial(i, j), \forall(i, j)\}$

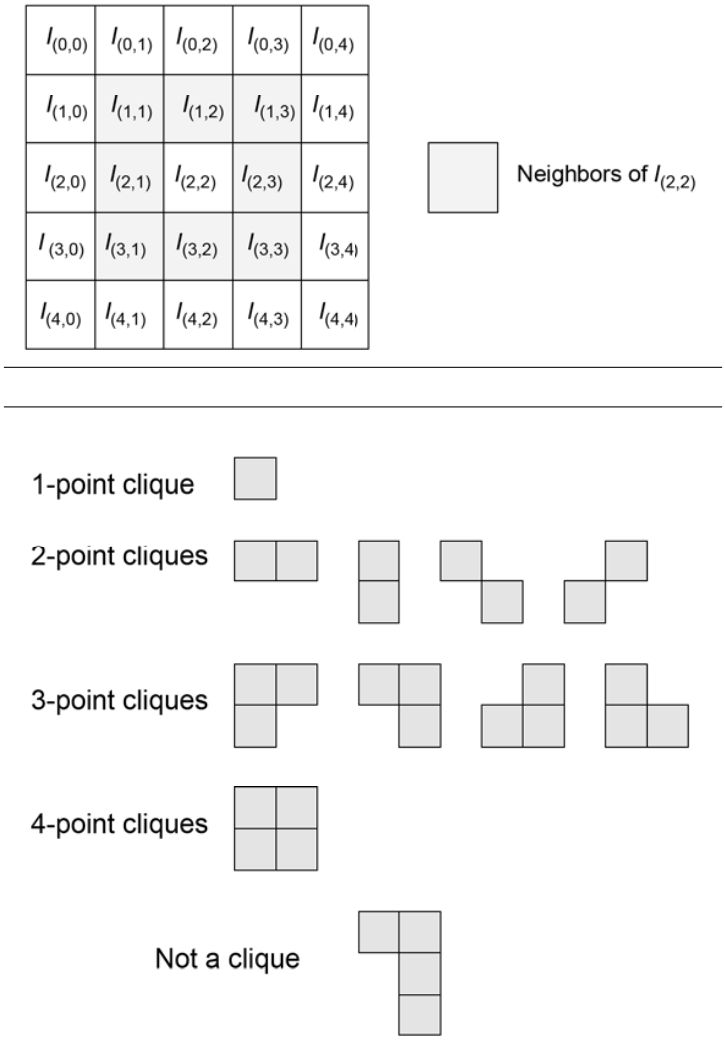



Figure 2.1: **Top:** Example of a $5 \times 5$ image patch, $N_2(2,2)$, centered at location $(2,2)$. The set of neighbors, $\partial(2,2)$ based on a smaller patch $N_1(2,2)$ is shown in gray. **Bottom:** Clique examples for a discrete lattice with an 8-connectivity structure. Figure modified from [14].

Mainly for convenience, let $\boldsymbol{N}_r(i,j)$ denote a symmetric neighborhood function, corresponding to an $(2r+1) \times (2r+1)$ sub-image centered at, and including, location $(i,j)$. The neighbors of $(i,j)$ are then given by $\partial(i,j) = \boldsymbol{N}_r(i,j) \setminus (i,j)$ (see Figure 2.1 for an example). Note that $\{(i,j)\} = \boldsymbol{N}_0(i,j) \subset \boldsymbol{N}_1(i,j) \subset ... \subset \boldsymbol{N}_r(i,j)$. To prevent border effects the images are appropriately padded based on $r$. Throughout this thesis we may refer to $\boldsymbol{N}_r(i,j)$ as an image patch. In more practical terms images and/or image patches are usually treated as either matrices, in the

case of grayscale images, or a tensors in the case of color or other multi-channel images. Formally, a 2-D image $\boldsymbol{I}$ is then defined as a function $\boldsymbol{I} : S \subset \mathbb{Z}^2 \mapsto [0, 255]^d$, where $d = 1$ for grayscale images and $d = 3$ for color images, and just to recap, $\boldsymbol{I}(i, j)$ indexes pixel $(i, j)$.

## 2.2 Image Segmentation

In computer vision, image segmentation refers to the process of partitioning a digital image into regions (sets of pixels), whereby each region satisfies some type of homogeneity criterion[39]. The result of image segmentation is a set of regions that collectively cover the entire image. Each of the pixels in a region are similar with respect to some characteristic or computed property, such as color, intensity, or texture. Adjacent regions are significantly different with respect to the same characteristic(s). Depending on the nature of the image and the type of region homogeneity criteria, each region may or may not correspond to an object. Sometimes a region can correspond to a part of a single object or may represent several similar objects grouped into a single blob. Therefore, image segmentation, in general, is an ill-defined problem since there is no **unique** ground-truth segmentation of a particular image against which the output of an algorithm may be compared.

Image segmentation can therefore be defined as partitioning the image $\boldsymbol{I}$ into a set of regions $\boldsymbol{G}_r \subset S, r = \{1, 2, ...\}$, satisfying (i) $S = \bigcup \boldsymbol{G}_r$, (ii) $\boldsymbol{G}_{r_1} \bigcap \boldsymbol{G}_{r_2} = \emptyset$ if $r_1 \neq r_2$, and (iii) each region, $\boldsymbol{G}_r$, forms a connected component.

General purpose (unsupervised) image segmentation algorithms typically work by analyzing: (a) regions using a similarity criterion or (b) interfaces - discontinuities between distinct regions. More recently, hybrid techniques utilizing both (a) and (b) have started to become popular. By unsupervised we mean that these algorithms do not utilize a training phase where the ideal output (i.e., ground truth) is available for learning purposes. The next few subsections provide a brief overview of different image segmentation methods, both supervised and unsupervised.

### 2.2.1 Edge Based Methods

Edge detection is a well-developed field on its own within image processing. Region boundaries and edges are closely related, since there is often a sharp adjustment in intensity at the region boundaries. Edge detection techniques have therefore been used as the base for interface analysis and image segmentation. The edges identified by edge detection are often disconnected. To segment an object from an image however, one needs closed region boundaries. Discontinuities are bridged if the distance between the two edges is within some predetermined threshold.

Unlike traditional edge detection methods, the authors of [82], proposed a much

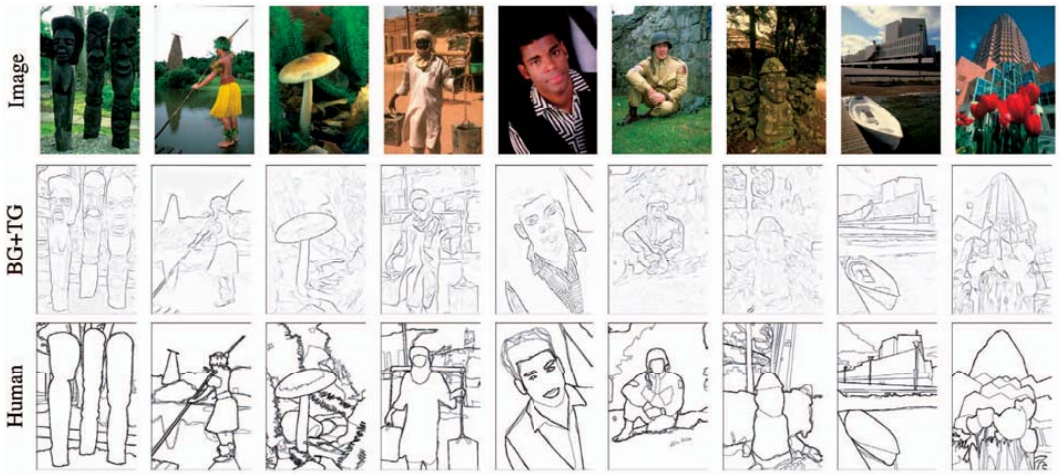



Figure 2.2: Boundary Detection using a machine learned classifier. **Top:** Input images. **Bottom:** boundaries identified by a human for training and evaluation purposes. **Middle:** Output produced by the trained classifier. From [82].

more sophisticated technique. Their goal was to accurately detect and localize boundaries in natural scenes using local pixel characteristics. The first step is to hand-design features (e.g., filters for edges, brightness, color, and texture) associated with natural boundaries, denoted $\boldsymbol{f}^{\boldsymbol{I}}(i,j)$. In order to combine the information from these features, a machine learned classifier is trained using human labeled ground truth images, $\boldsymbol{L}_{boundary}$ (see Figure 2.2). The output of this classifier provides the posterior probability of a boundary at each image lattice cite $(i,j)$:

$$\boldsymbol{P}(i,j) = p[\boldsymbol{L}(i,j) = 1 \mid \boldsymbol{f}^{\boldsymbol{I}}(i,j)]$$

with $p[\cdot]$ denoting the probability density function.. The precision-recall curves demonstrated that the resulting boundary detector significantly outperforms classic approaches. The two main results of this work are: a) cue combination can be performed adequately with a simple classifiers and b) proper, and explicit treatment of texture is required in order to detect boundaries in natural images. Traditionally, neural nets have also been shown to work well for edge detection and boundary delineation [30]. We build on this idea further in the later sections of this chapter.

### 2.2.2 Thresholding and Histogram Based Methods

The most basic thresholding technique, [39] partitions a grayscale image $\boldsymbol{I}$ into two sets of regions (foreground and background), based on a predefined threshold $\tau$. If pixel $\boldsymbol{I}(i,j) \geq \tau$, it is labeled as foreground, while pixels falling below

the threshold are considered to be background leading to a foreground-background segmentation $\boldsymbol{I}_\tau$:

$$\boldsymbol{I}_\tau(i,j) = \begin{cases} 0 & \text{if } \boldsymbol{I}(i,j) < \tau \\ 1 & \text{if } \boldsymbol{I}(i,j) \geq \tau \end{cases} \tag{2.2}$$

Thus the image is partitioned into a set of connected components representing foreground and background objects. For multichannel images, a vector, $\boldsymbol{\tau}$ can be used in lieu of a single scalar threshold.

Multiple thresholds can be used to partition the image into several regional classes. By computing the histogram from all of the pixels in the image, the peaks and valleys within the histogram can be utilized to identify the number and locations or thresholds. If histogram peaks and troughs are well defined an automated procedure can be easily constructed to automatically find the thresholds. A refinement of this technique is to recursively apply the histogram-seeking method to clusters in the image in order to divide them into smaller clusters. This is repeated with smaller and smaller clusters until no further clusters are formed.

Although thresholding is very efficient, since it typically requires only one pass through the pixels, it has significant drawbacks. One disadvantage of the histogram-seeking method is that it may be difficult to identify significant peaks and valleys in the image. As the histogram in Figure 3.1 demonstrates, for most real world domains a global threshold does not exist that will adequately separate regions of interest. To overcome this problem adaptive techniques (e.g., Multistage Adaptive Thresholding [128]) have been designed, that analyze and threshhold image patches independent of other image locations. Rather than using global image statistics to determine a set of thresholds, these algorithms identify and use a different set of threshold for each image patch, thereby adapting to local image statistics.

### 2.2.3 Region Based Methods

Unlike edge-based methods, region-based techniques are guaranteed (by definition) to produce coherent regions. Linking edges, gaps produced by missing edge pixels, and other anomalies are not an issue. Region based methods inherently work from the inside out, rather than from the outside inward. Resolving which object a pixel belongs to is immediate, and not the result of contour/curvature tests needed in edge-based methods. The methods are, however, not without drawbacks. Region membership decisions are often more difficult than applying simple edge detectors. Objects that span multiple disconnected regions (e.g., during occlusion) cannot be found by basic techniques.

### Graph Partitioning

The normalized cuts method, first proposed by Shi and Malik in [114], models the image as a weighted undirected graph, $G = (V, E_w)$. As before, each vertex $v \in V$ represents a pixel and a weighted edge, $e \in E$, is formed between every pair of connected pixels where the weight associated with an edge $w \in W$, is a measure of the similarity between two connected pixels. The image is partitioned into regions by removing the edges connecting the segments. The optimal partitioning of the graph is the one that minimizes the weights of the edges that were removed (the cut). Shi's algorithm seeks to minimize the normalized cut, which is the ratio of the number of cut edges to all of the edges in the set. The resulting connected components correspond to image regions.

### Clustering Methods

Typically considered an unsupervised machine learning technique, clustering refers to the process of grouping similar 'items' together into groups or clusters. The $K$-means algorithm is a general purpose clustering method that iteratively partitions an image into $K$ clusters. The basic algorithm is as follows:

1. Pick $K$ cluster centers, either randomly or based on some heuristic.
2. Assign each pixel in the image to the cluster that minimizes the variance between the pixel and the cluster center.
3. Re-compute the cluster centers by averaging all of the pixels in the cluster.
4. Repeat steps 2 and 3 until convergence is attained (e.g. no pixels change clusters)

In this case, variance is the squared or absolute difference between a pixel and a cluster center. The difference is typically based on pixel color, intensity, texture, and location, or a weighted combination of these factors. $K$ can be selected manually, randomly, or by a heuristic. This algorithm is guaranteed to converge, but it may not return an optimal solution (i.e., it can get stuck in a local minima). The quality of the solution depends on the initial set of clusters, the value of $K$ and as always the suitability of the similarity metric. Recently, much more specialized and sophisticated methods have been developed based on clustering, most notably the mean-shift algorithm and bilateral filtering [5]. By recursively (re)-clustering pixels based on both spacial proximity to a given region and intensity based similarity, these algorithms converge to the modes of the probability density function defined by the aforementioned joint similarity function. An interesting property of these algorithms is that they iteratively modify/reassign pixel intensities rather than their cluster memberships.

### 2.2.4 General Region Growing

One of the first region growing methods, is the seeded region growing method [2]. The algorithm takes as input a set of seeds or markers, along with a corresponding image. Each seed marks an interior subregion of an object to be segmented. The regions are iteratively grown from seeds by comparing all unallocated pixels neighboring the current regions. The algorithm uses a similarity function between a region and a neighboring pixel, in order to decide whether the pixel should be added to the region. The pixel with the highest similarity to the current region is added first. The process continues until all pixels are allocated to one of the regions. Since this type of region growing requires seeds as additional input, the final segmentation is dependent on the choice of seeds (and the similarity function used). Noise, non-uniform illumination and other image variabilities can cause the seeds to be poorly placed or erroneously identified.

Region growing without markers, is a modified algorithm that doesn't require explicit seeds. It starts off with a single (1-pixel) region $S_1$ the actual pixel chosen here does not significantly influence the final segmentation. As with seeded region growing, at each iteration the algorithm considers absorbing the neighboring pixels using a predefined similarity function. However, in contrast to seeded region growing, when the pixel-region similarity is above a predefined threshold the pixel is added to the respective region $S_k$, regardless if other neighboring pixels are a better match (i.e., have an even higher similarity score w.r.t. $S_k$). If the similarity does not fall above the threshold, the pixel is considered significantly different from the neighboring region(s) $S_1, ..., S_k$, and a new region $S_{k+1}$ is created using this pixel as the starting point.

At first glance, region growing appears to share many properties with clustering. However, pixels can move from region to region during the iterative clustering procedure. On the other hand, once a pixel is assigned to a specific region, it cannot be reassigned to a different region. The incremental assignment of pixels to regions is ideal when pixel exhibit strong spacial interdependencies. From a different perspective, the mean-shift algorithm converges to cluster modes which in turn have a loose correspondence to seeds used in region growing.

#### Split and Merge Algorithms

As with all region based methods, the basic idea of region splitting [39], is to break the image into a set of disjoint regions that are coherent within themselves. Initially the image as a whole is taken to be the area/region of interest (ROI). Given an ROI, the algorithm recursively decides if all pixels contained within the region satisfy some similarity constraint. If TRUE then the area of interest corresponds to a region within the image. If FALSE, the region is split up (usually into four equal sub-areas). Subsequently, each of the sub-areas is now considered for further

splitting. The process continues until no further splitting is possible, based on the aforementioned similarity criteria. In the worst case all the areas are just one pixel in size. Split and Merge algorithms are top down, divide-and-conquer type methods. If only a splitting schedule is used then the final segmentation most likely will contain many neighboring regions that have identical or similar properties. Therefore, a merging process is used after each split in order to compare adjacent regions and merge them if necessary, using a second similarity criterion.

### Watershed Segmentation

The watershed algorithm [10] is commonly used within the unsupervised setting of segmenting an image into a set of non-overlapping regions. The framework of mathematical morphology considers gray-scale images to be sets of points in a three-dimensional space, with the third dimension constituting gray level image intensity [39]. This topographical analogy respectively considers light and dark image areas as the hills and valleys of an image landscape. To segment a given image the "landscape" is flooded, whereby water flows from high altitude areas (areas with high gray-scale values) along lines of steepest descent until it reaches some regional minimum (low gray-scale regions). The watersheds or catchment basins of the image are the draining areas of its regional minima. These areas are separated by lines called watershed lines.

Unfortunately, the segmentation produced by a naïve application of the watershed algorithm is oftentimes inadequate: the image is usually over-segmented into a large number of minuscule regions. As a result, several extensions have been proposed in order to produce more natural image segmentation (e.g., hierarchical watersheds or region split/merge [12]). The most common remedy is to use markers for identifying relevant region minima (e.g., [2, 31]). By setting marker locations as the only local minima within the watershed image the number of regions can be automatically controlled. Unfortunately, finding markers can itself be problematic and is one of the focal points of this thesis.

To formally define the watershed segmentation algorithm several preliminary definitions are in order.

**Definition 2.2.1** *The global image minimum $G_{gmin}(\boldsymbol{I})$ of a grayscale image $\boldsymbol{I}$, is defined as:*

$$G_{gmin}(\boldsymbol{I}) = \{(i,j) \in S | \boldsymbol{I}(i,j) \leq \boldsymbol{I}(i',j') \forall (i',j') \in S\}$$

**Definition 2.2.2** *The local image minima $G_{lmin}(\boldsymbol{I})$ with respect to a neighborhood $\partial(i,j)$ function, is defined as:*

$$G_{lmin}(\boldsymbol{I}) = \{(i,j) \in S | \boldsymbol{I}(i,j) \leq \boldsymbol{I}(i',j') \forall (i',j') \in \partial(i,j)\}$$

**Meyers Watershed algorithm [86]**

The input to the algorithm is a grayscale image $\boldsymbol{I}$ and a set of markers, with each marker being a region $G_k$ of image $\boldsymbol{I}$:

$$\mathfrak{M} = \{G_k \subset S | G_{k_1} \cap G_{k_2} = \emptyset, \exists \pi_{(i,j),(i',j')} \forall (i,j),(i',j') \in G_k\}$$

where the existence of a path $\pi$ between site $(i,j)$ and $(i',j')$ implies that $G_k$ is connected. In the trivial case, the marker regions correspond to the local minima of $\boldsymbol{I}$, i.e., $\mathfrak{M} = G_{lmin}(\boldsymbol{I})$[1]. The algorithm expands as much as possible the set $\mathfrak{M}$, while preserving the number of connected components in $\mathfrak{M}$.

Let $G_0 = S \setminus \mathfrak{M}$ represent the set of as yet unassigned pixels, $G_{ws} = \emptyset$ the (initially empty) set of watershed pixels, and $\partial G_k = \{(i,j) \in G_0 | (i,j) \in \partial(i',j'),\ (i',j') \in G_k\}$ denote the unassigned neighbors of region $G_k$. The basic watershed algorithm is then given by the following steps:

1. Compute $\partial\mathfrak{M} = \bigcup \partial G_k, k \geq 1$, the set of unique pixels adjacent to, but not part of $\mathfrak{M}$.

2. If $\partial\mathfrak{M} = \emptyset$ terminate, otherwise go to Step 3.

3. Find $(i,j) \in G_{gmin}(\boldsymbol{I}(\partial\mathfrak{M}))$, the neighbor of $\mathfrak{M}$ with the lowest grey level value. If

$$\exists (i_1,j_1),(i_2,j_2) \in \partial(i,j) \mid \quad (i_1,j_1) \in G_{k_1}, (i_2,j_2) \in G_{k_2}, \quad k_1 \neq k_2, k_1 \geq 1, k_2 \geq 1$$

   then the point $(i,j)$ is adjacent to two distinct regions $G_{k_1}$, $G_{k_2}$, and is thus assigned to $G_{ws}$, the set of watershed pixels.

   Otherwise, $\exists k \geq 1 | \partial(i,j) \in G_0 \cup G_{ws} \cup G_k$, the neighborhood of $(i,j)$ is composed of: (i) unassigned locations, (ii) watershed locations, and (iii) only one region $G_k$, and therefore location $(i,j)$ is assigned to region $G_k$.

Notice that the number of regions is equivalent to the number of markers and the set of watershed points comprising the watershed lines is the complement of the set of labeled points. The outlined algorithm neither labels nor propagates these watershed pixels, which stop the growing process.

[1] Here the notation is slightly abused, since the set of markers actually corresponds to the connected components of $G_{lmin}(\boldsymbol{I})$. This ensures that individual pixels belonging to the same local minima (i.e., flat spots) do not get assigned to different marker regions.

### Level Set Methods

Curve propagation is a popular technique in image analysis for object extraction, object tracking, stereo reconstruction, etc. The central idea behind such an approach is to evolve a curve towards the lowest potential of a cost function, where its definition reflects the task to be addressed and imposes certain smoothness constraints. Lagrangian techniques are based on parameterizing the contour according to some sampling strategy and then evolve each element according to image and internal terms. While such techniques can be very efficient, they suffer from various limitations including: (i) deciding on the sampling strategy, (ii) estimating the internal geometric properties of the curve, (iii) changing its topology, (iv) addressing problems in higher dimensions.

The level set method was initially proposed to track moving interfaces by Osher and Sethian [94], and has spread across various imaging domains since then. It can be used to efficiently address the problem of curve/surface propagation in an implicit manner. The central idea is to represent the evolving contour using a signed distance function, where its zero level set corresponds to the actual contour of interest.

Then, according to the motion equation of the contour, one can easily derive a similar flow for the implicit surface that when applied to the zero-level set will reflect the propagation of the contour. The level set method encodes numerous advantages: it is implicit, parameter free, provides a direct way to estimate the geometric properties of the evolving structure, can change the topology and is intrinsic. It is, therefore a very convenient framework to address numerous applications of computer vision and medical image analysis[2]. In fact, the watershed algorithm is a special case of the more general level set algorithm family.

## 2.3 Object Segmentation

In contrast to *image segmentation*, we define *object segmentation* as the identification and delineation of target objects from the background and each other. As an example consider the images depicted in Figure 2.3. The input image (top left) contains several blobs, some of which are very close to each other. The goal is to identify each blob and distinguish it from both the background and other blobs. The bottom left sub-figure presents the result of global thresholding. While the method adequately separated foreground objects from the background, individual objects are not identified. In contrast, the result of watershed segmentation, depicted in the bottom right sub-figure, separated objects (for the most part) from both the back-

[2]Excerpt based on `http://en.wikipedia.org/wiki/Segmentation_(image_processing)`

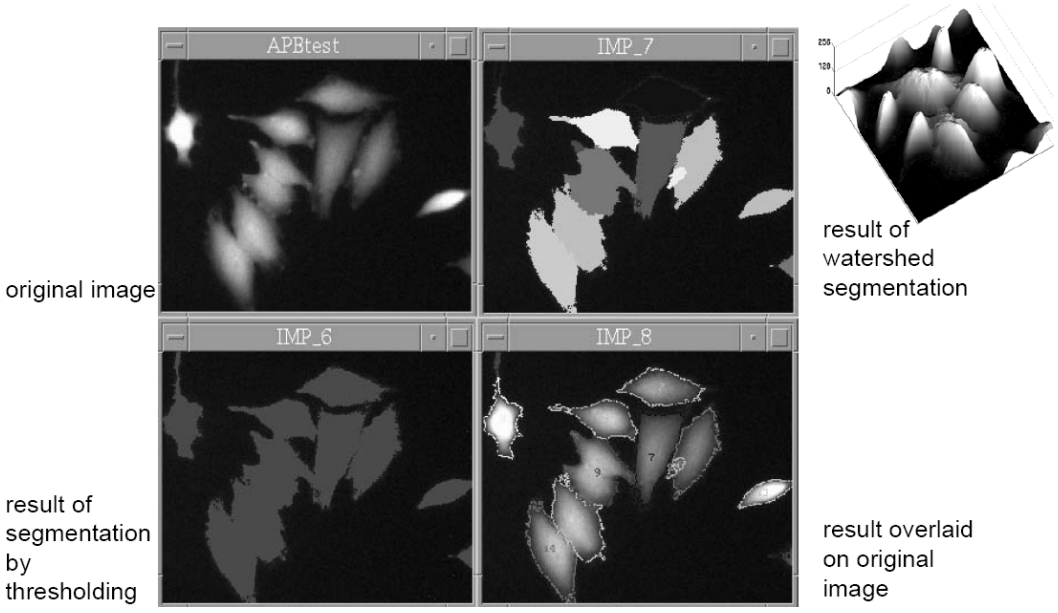



Figure 2.3: Image Segmentation versus Object Segmentation. **Top left:** The input image containing several objects, some of which are in very close proximity to each other. The goal is to identify each blob and distinguish it from both the background and other blobs. **Bottom left:** The result of global thresholding. While the method adequately separated foreground objects from the background, individual objects are not identified. **Top Middle:** The result of watershed segmentation, which separated objects from both the background and each other. **Bottom left:** Individual objects delineated by white lines. **Top right:** Visualization of the input image as a 3-D topology.

ground and each other. Unlike image segmentation, where the objective function is ill defined, it is possible (and relatively easy) to precisely define the goal of object segmentation using a ground truth image, $\boldsymbol{L}$. Figure 2.4 on p. 22 presents an example input image, the desired ground truth image, and a candidate object segmentation. Observe that while the majority of individual pixels are correctly labeled, the candidate segmentation completely obscures the characteristics (e.g., size, orientation, position) of individual objects present within the input image. Hence, the candidate object segmentation is of high quality with respect to individual pixels but is extremely poor with respect to objects. To properly evaluate the quality of the segmentation, all experimental results will use both pixel level and object level metrics presented in Appendix C on p. 138.

In general, to produce a 'good' object level segmentation, an algorithm needs to identify: (i) foreground/background pixels, and (ii) object-object boundaries. Despite the earlier criticism of pixel level analysis, pixel labeling is clearly an im-

portant intermediate step necessary for identifying foreground/background regions. With that in mind, the next chapter is entirely devoted to machine learning approaches for pixel labeling. For the moment let us define a general mapping from image $\boldsymbol{I}$ to a binary label image $\boldsymbol{L}_{region}$ as:

$$h_{region} : \boldsymbol{I} \mapsto \boldsymbol{L}_{region} \tag{2.3}$$

where $\boldsymbol{L}_{region}(i,j) \in \{0,1\}$. The $h_{region}$ classifier models the separation of pixels into foreground and background. Clearly if $\boldsymbol{L}_{region} = \boldsymbol{L}$, we have solved the problem. Unfortunately, many pixel labeling algorithms produce results akin to the candidate segmentation in Figure 2.4 and hence require subsequent object-object boundary identification step to be of practical use. Before describing the proposed Data Driven Region Growing framework, we first generalize Equation 2.3. Since machine learning algorithms will be employed as an intermediate step, the goal of this intermediate step is to produce a probability map $\boldsymbol{P}_{region}$, where $\boldsymbol{P}_{region}(i,j)$ denotes the likelihood of pixel $(i,j)$ belonging to a foreground object (or for non-probabilistic methods an un-normalized potential). To produce the label image, the probability map is thresholded via:

$$\boldsymbol{L}_{region}(i,j) = \begin{cases} 0 & \text{if } \boldsymbol{P}_{region}(i,j) < \tau \\ 1 & \text{if } \boldsymbol{P}_{region}(i,j) \geq \tau \end{cases} \tag{2.4}$$

.

### 2.3.1 Data Driven Region Growing

A popular approach to resolve object-object boundaries is to use region growing methods such as watershed. However, to be effective these methods require object markers. Using ad-hoc rules to extract markers requires a priori knowledge of either (a) the number of objects within an image as in [2], (b) specific image properties, or (c) object locations (e.g., medical images registered to an anatomical template). In each of these cases, the parameters governing marker extraction tend to vary from image to image. In contrast, our goal is develop robust and portable end-to-end object segmentation systems without the need to recode the system for every domain. This motivates the use of machine learning approaches for robust identification of object markers.

In [77], the Bayesian marker extraction algorithm utilized a naive Bayes classifier in order to generate object markers. Unfortunately, since the classifier is trained on the ground truth delineating whole objects, the approach does not provide any constraints to ensure that only one marker per target object is extracted, nor that the extracted markers even lie within the object boundary. Naturally, one could threshold the probability map, $\boldsymbol{P}_{region}$, using a higher value for threshold $\tau$. As a consequence, fewer pixels will be labeled as foreground, thus improving precision

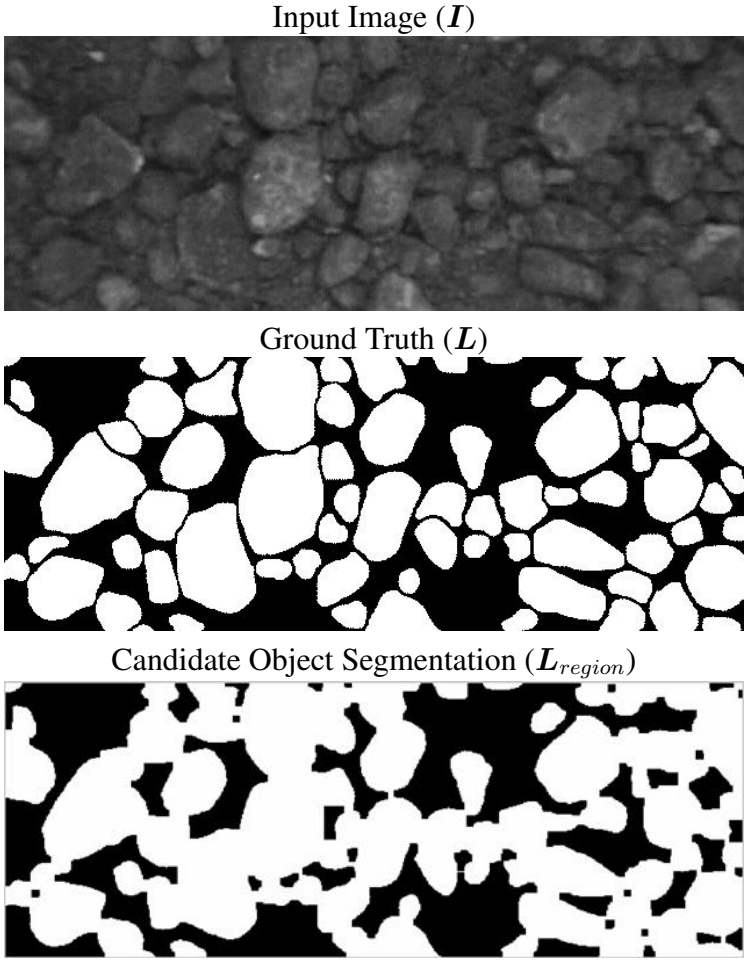



Figure 2.4: **Top:** Input image of a granulous material (in this case frozen oil sand ore) on a conveyor belt. **Middle:** Ground truth image produced by a domain expert. **Bottom:** An example of a good pixel labeling but a very bad object segmentation. See Appendix C for quantitative evaluation measures of image/object segmentation.

at the cost of recall. The potential outcome may be that pixels corresponding (with higher probability) to object markers may be extracted. However, there is still no guarantee that the markers will be within object boundaries, nor that there will be a one-to-one correspondence between objects and markers. To improve the situation, we propose training: (i) a marker identification classifier, $h_{marker}$, (ii) a boundary identification classifier, $h_{boundary}$, and (iii) a foreground-background classifier, $h_{region}$. Let:

$$\boldsymbol{L}_{eroded} = \boldsymbol{L} \ominus B \tag{2.5}$$

$$\boldsymbol{L}_{dilated} = \boldsymbol{L} \oplus B \tag{2.6}$$

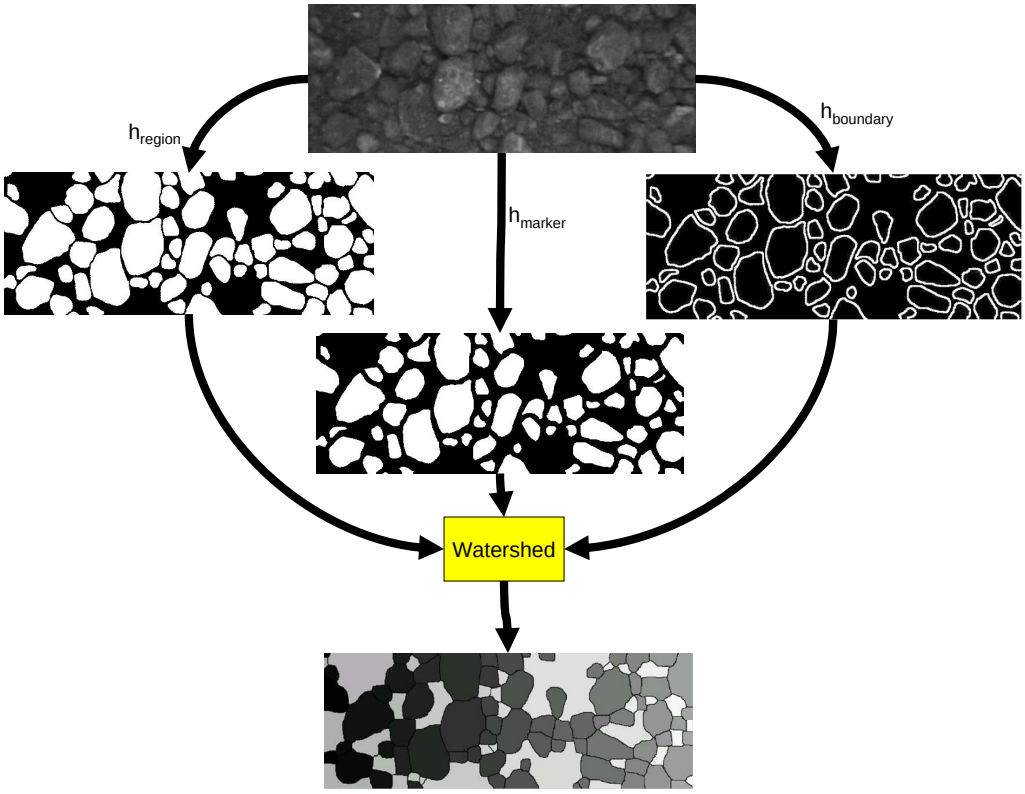



Figure 2.5: A depiction of the desired mappings used by the Data Driven Region Growing.

respectively denote the morphological erosion/dilation [39] of the true label image $L$ by a suitably chosen structural element $B$[3]. Based on these transformed ground truth images, two auxiliary mappings to produce markers and boundaries can now be constructed as follows:

$$\boldsymbol{L}_{boundary} = \boldsymbol{L}_{dilated} - \boldsymbol{L}_{eroded} \tag{2.7}$$

$$\boldsymbol{L}_{marker} = \boldsymbol{L}_{eroded} \tag{2.8}$$

To make the notational distinction more pronounced, henceforth let $h_{type}$, denote the classifier trained to produce probability map $\boldsymbol{P}_{type} = h_{type}(\boldsymbol{I})$, which is subsequently thresholded to produce the label image, $\boldsymbol{L}_{type}$:

$$\boldsymbol{L}_{type}(i,j) = \begin{cases} 0 & \text{if } \boldsymbol{P}_{type}(i,j) < \tau_{type} \\ 1 & \text{if } \boldsymbol{P}_{type}(i,j) \geq \tau_{type} \end{cases}$$

[3]We use a disk inscribed within an $N_r(i,j)$ patch. The typical size is $r = 3$. See Chapter 6 for details.

**On-line DDRG** $(I, h_{region}, h_{marker}, h_{boundary}\tau_1, \tau_2, \tau_3, ExtractFeatureMaps())$

| | | |
|---|---|---|
| INPUT: | $I$: | an input image. |
| | $h_{marker}$: | a classifier trained to produce markers. |
| | $h_{region}$: | a classifier trained to delineate object-background. |
| | $h_{boundary}$: | a classifier trained to detect object contours. |
| | $\tau_1, \tau_2, \tau_3$: | thresholds used to segment probability maps. |
| | $ExtractFeatureMaps()$: | function for feature extraction. |

OUTPUT: $L$ - a labeling for image $I$.

| | | |
|---|---|---|
| 1 | $\boldsymbol{f}$ = ExtractFeatureMaps($I$) | // Extract features for each pixel |
| 2 | $P_{marker} = h_{marker}(\boldsymbol{f})$ | // create probability map |
| 3 | $P_{region} = h_{region}(\boldsymbol{f})$ | // create probability map |
| 4 | $P_{boundary} = h_{boundary}(\boldsymbol{f})$ | // create probability map |
| 5 | $bg = invthreshold(P_{region}, \tau_1)$ | // find background regions |
| 6 | $seeds = threshold(P_{marker}, \tau_2)$ | // create object seeds |
| 7 | $contours = threshold(P_{boundary}, \tau_3)$ | // create object contours |
| 8 | $ws = setlocallmin(1 - P_{region}, seeds)$ | // set seeds ws(seeds)=$-\infty$ |
| 9 | $ws = setlocallmax(ws, contours)$ | // set contours ws(contours)=$\infty$ |
| 10 | $L = watershed(ws) - bg$ | // label image, remove background |

Figure 2.6: On-line testing/execution phase of Data Driven Region Growing.

where $type \in \{region, boundary, marker\}$ and $region$ denotes the standard ground truth output. The Data Driven Region Growing algorithm, depicted in Figure 2.5, therefore consists of three mappings:

$$h_{region} : \boldsymbol{I} \mapsto \boldsymbol{L}_{region} \tag{2.9}$$

$$h_{marker} : \boldsymbol{I} \mapsto \boldsymbol{L}_{marker} \tag{2.10}$$

$$h_{boundary} : \boldsymbol{I} \mapsto \boldsymbol{L}_{boundary} \tag{2.11}$$

As the experimental results in Section 6.1.2 will demonstrate, the $h_{marker}$ classifier is overly conservative (i.e., higher precision, lower recall) and produces superior object markers as compared to thresholding $\boldsymbol{P}_{region}$ using higher values of $\tau$.

Having described both $\boldsymbol{P}_{region}$, used to delineate object-background boundaries, and $\boldsymbol{P}_{marker}$ used to identify object markers, the topological surface utilized by the watershed algorithm is discussed next. Again, several options exist, with the de facto standard approach utilizing the gradient of the original image. However, since the probability maps themselves form a topological surface, the machine learned output, $\boldsymbol{P}_{region}$ can be once again utilized. Intuitively, the highest intensity values within the $\boldsymbol{P}_{region}$ image correspond to pixels with the highest probability of being part of the target class, hence using the inverted probability map, $1 - \boldsymbol{P}_{region}$, can be advantageous because the aforementioned high probability regions will be flooded first. Unfortunately, more than one local maximum may be present within

large sized objects, thereby motivating the need for markers. To produce a topology amenable to the watershed algorithm, we invert the probability map $\boldsymbol{P}_{region}$, and set the regional minima to correspond with marker locations extracted from the $\boldsymbol{P}_{marker}$ probability map by thresholding via Equation 2.2 on p. 14. Furthermore, to provide additional constraints during the region growing step, potential object boundaries, identified by the $h_{boundary}$ mapping, are used as barriers to prevent growth. In other words, the output of $h_{boundary}$ is used to set the local maxima, whereby the pixels identified by the boundary classifier act as potential watershed pixels and are forced to be flooded last. The original formulation, in [75] did not use the boundary constraints, but as the experimental results in Chapter 6 will reveal, boundary constraints can provide a significant performance boost. In general, perfect $P_{region}$, or perfect $P_{marker}$ plus $P_{boundary}$ will produce optimal results. However, in practice, none of the probability maps will be perfect as can be seen in Figure 2.7. In chapter 6, experimental results will demonstrate that the introduced redundancy is highly beneficial for improving the final object segmentation. To provide additional details, Figure 2.6 on p. 24 presents the pseudo-code describing the on-line phase of the data driven region growing algorithm (DDRG).

Having defined the mappings constituting the data driven region growing framework, we turn our attention to learning the models: $h_{region}$, $h_{boundary}$, and $h_{marker}$. As mentioned in the introductory chapter, manually defining these mappings would be a tedious and difficult process, that would need to be repeated for each new domain. In order for the framework to be of practical use, an automated algorithm is needed to extract image features and map these features to the aforementioned probability maps, $\boldsymbol{P}_{type}$. The next three chapters explore the machine learning possibilities for a fully automated instantiation of the DDRG framework.

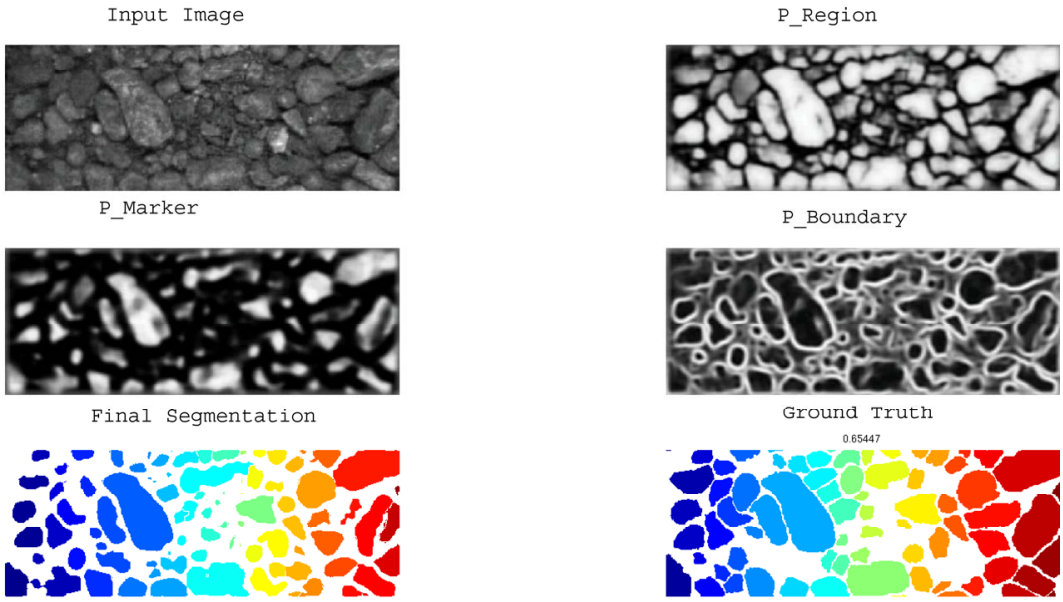



Figure 2.7: Actual components used as input into the Data Driven Region Growing Framework (DDRG).

# Chapter 3

# Learning to Label

This chapter reviews a number of commonly used machine learning techniques for pixel labeling. Particular focus is given to neural networks that form the basis of subsequent discussions on Automated Feature Extraction and Deep Convolutional Networks presented in subsequent Chapters. Chapter 3 also presents the second contribution of this dissertation, namely the Heterogeneous Stacking algorithm. In a manner analogous to the previously presented DDRG framework, Heterogeneous Stacking used Ground Truth manipulation to build a set of base classifiers which are subsequently fused together by a higher level model enabling improved object segmentation quality.

## 3.1 Pixel Classification

A very common data driven approach to image segmentation attempts to learn a pixel classifier that assigns to each pixel the probability of belonging to a given class. Recall that $(i, j)$ index a discrete set of sites on a spatially regular $N \times M$ lattice:

$$S = \{(i, j) | 1 \leq i \leq N, 1 \leq j \leq M\}$$

For each input image $\boldsymbol{I}$ and the corresponding image labeling $\boldsymbol{L}$, let $\boldsymbol{I}(i, j) \in [0, 1]^d$ and $\boldsymbol{L}(i, j) \in \{0, 1\}$ respectively denote the intensity values of image pixels and the corresponding (binary) labels. Throughout this thesis, $\boldsymbol{L}(i, j) = 0$ labels the lattice site $(i, j)$ as background, while $\boldsymbol{L}(i, j) = 1$ denotes the pixel belongs to the target object class. The main objective is to produce a probability map $\boldsymbol{P}$:

$$\boldsymbol{P}(i, j) = p[\boldsymbol{L}(i, j) = 1 \mid \boldsymbol{I}(i, j)] \;\; \forall (i, j) \in S \tag{3.1}$$

with $p[\cdot]$ denoting the probability density function. To obtain the final image segmentation, $\hat{\boldsymbol{L}}$, the probability map $\boldsymbol{P}$ is globally thresholded:

$$\boldsymbol{L}(i, j) = \begin{cases} 0 & \text{if } \boldsymbol{P}(i, j) < \tau \\ 1 & \text{if } \boldsymbol{P}(i, j) \geq \tau \end{cases} \tag{3.2}$$

The process in Equation 3.1 treats individual pixels as i.i.d. (independent identically distributed). Unfortunately, this assumption is rarely satisfied in practice, since most non-trivial domains exhibit complex pixel interactions and dependencies. Therefore, simply using raw pixel values for classification in equation 3.1 results in very poor segmentation. (Otherwise thresholding the input image at every pixel $\boldsymbol{I}(i, j) > \tau$ would produce the desired result. The histogram at the bottom of Figure 3.1 clearly demonstrates the practical shortcomings of this approach.) To overcome this problem, feature extraction techniques are needed to produce a set of feature maps describing local (and possibly global) image characteristics. The specific feature extraction methods used in our research, are discussed in later sections

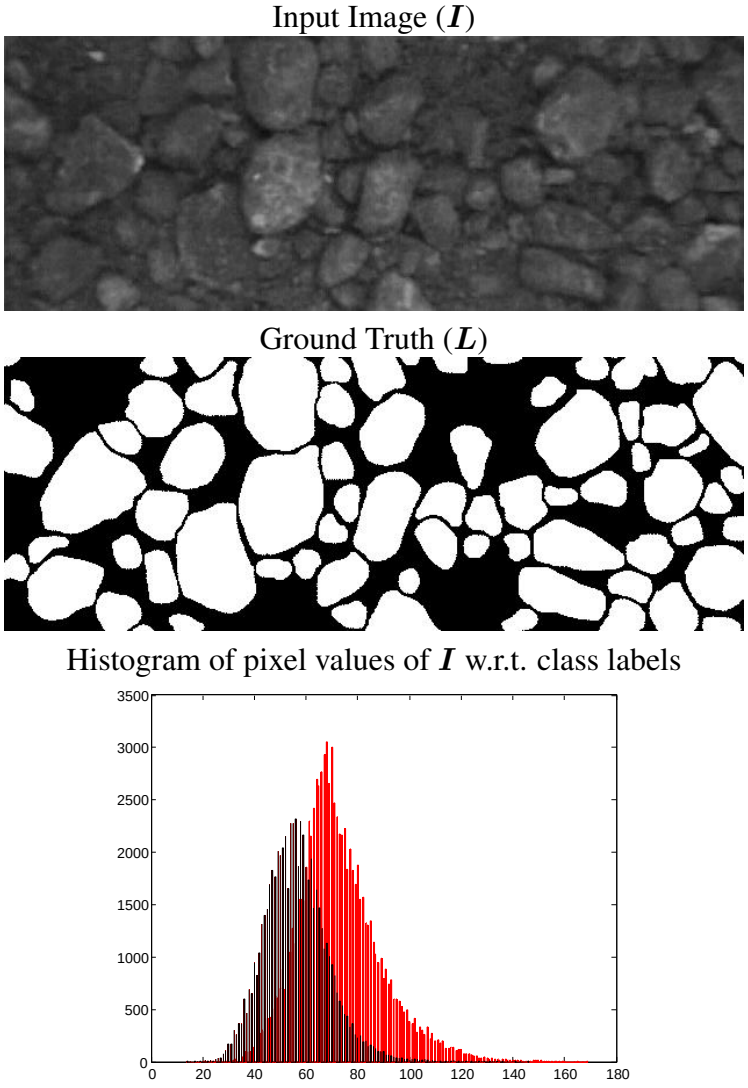



Figure 3.1: **Top:** Input image of a granulous material (in this case frozen oil sand ore) on a conveyor belt. **Middle:** Ground truth image produced by a domain expert. **Bottom:** Histogram of pixel intensities for each class.

of this dissertation. For the moment, let $\boldsymbol{f}(i,j)$ denote the extracted feature vector at each lattice site $(i,j)$. The probability map can now be conditioned on the feature vectors rather than just the raw grayscale or multi-spectral values as follows:

$$\boldsymbol{P}(i,j) = p[\boldsymbol{L}(i,j) = 1 \mid \boldsymbol{f}(i,j)] \;\; \forall (i,j) \in S \tag{3.3}$$

## 3.2 Learning Classifiers

The general form $p[y = l|\boldsymbol{x}]$ in equation 3.3 defines an arbitrary probabilistic binary classifier, where the column vector $\boldsymbol{x} \in \mathbb{R}^d$ consists of $d$ input features, and $y \in \{0, 1\}$ is the (desired) output. Numerous machine learning methods exist for learning such classifiers, given a set of training data $D = \{\boldsymbol{x}_k, y_k\}_{k=1}^n$. Perhaps the simplest such classifier is a linear function approximator obtained by using linear regression:

$$h_\omega(\boldsymbol{x}) = \omega_0 + \boldsymbol{\omega}_1^T \boldsymbol{x} \tag{3.4}$$

The parameters, $\boldsymbol{\omega} = \{\omega_0, \boldsymbol{\omega}_1\}$ can be easily determined by solving:

$$\boldsymbol{\omega} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y} \tag{3.5}$$

where the $d$-dimensional vector $\boldsymbol{x}$ is

$$\boldsymbol{x} = [x(1), ..., x(d)]^T \tag{3.6}$$

the data matrix $\boldsymbol{X}_0$ is then formed by:

$$\boldsymbol{X}_0 = [\boldsymbol{x}_1|...|\boldsymbol{x}_n]^T \tag{3.7}$$

and $\boldsymbol{X}$ symbolizes the mean normalized (i.e., centered) data matrix augmented with a vector of ones, $\mathbf{1}$:

$$\boldsymbol{X} = [\mathbf{1}\ (\boldsymbol{X}_0 - \boldsymbol{\mathcal{M}})],\ \boldsymbol{\mathcal{M}} = [\boldsymbol{\mu}|...|\boldsymbol{\mu}]^T,\ \boldsymbol{\mu} = \frac{1}{n}\sum_{k=1}^{n} \boldsymbol{x}_k \tag{3.8}$$

and $\boldsymbol{y}$ is a vector of target outputs. Note, $h_{\boldsymbol{\omega}}$ henceforth denotes the trained pixel classifier. In general linear regression has two major drawbacks: (i) inverting $\boldsymbol{X^T X}$ becomes an ill conditioned problem if the features in $\boldsymbol{x}$ are collinear (i.e., correlated) [1], and (ii) the output is not a true probability, which must be bound to be in the range $[0, 1]$.

To overcome the second limitation, another widely used algorithm models the class conditional using the Generalized Linear Model [45] with the logistic link function as follows:

$$h_{\boldsymbol{\omega}}(\boldsymbol{x}) = p[y = 1 \mid \boldsymbol{x}] = \frac{1}{1 + e^{-(\omega_0 + \boldsymbol{\omega}_1^T \boldsymbol{x})}} \tag{3.10}$$

[1] Let $E\{\cdot\}$ denote expectation. Then two (zero mean) variables are (linearly) uncorrelated if their covariance, $C_{x,y}$, is zero:

$$C_{x,y} = E\{xy\} = 0 \tag{3.9}$$

here the model parameters, $\boldsymbol{\omega} = \{\omega_0, \boldsymbol{\omega}_1\}$, can be estimated by maximizing the log-likelihood of the training data using standard non-linear optimization routines[2]. The logistic transform,

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.11}$$

ensures that the output is in the range $[0, 1]$, however, the assumption that the input features are uncorrelated is still present and can result in substantial estimation errors for parameters $\boldsymbol{\omega}$. To overcome this problem feature, decorrelation techniques need to be used. These are discussed at some lengths in the context of automated feature extraction in the next chapter.

## 3.3 Neural Networks

There is, however, one particular class of algorithms that can handle correlated inputs. Artificial neural networks (ANN) [110] are a general class of function approximators that can, in theory, approximate arbitrary functions[3]. Technically speaking the aforementioned linear and logistic regression models are special cases of ANNs with no hidden layer. Research on application of Neural networks to image processing spans several decades and had produced a number of interesting algorithms [30].

The standard (2 layer) Multi-Layer Perceptron (MLP) is defined by:

$$h_{\boldsymbol{\omega}}(\boldsymbol{x}) = \boldsymbol{g}(\boldsymbol{W}_2\, \boldsymbol{g}(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2)(\boldsymbol{x}) \tag{3.12}$$

where $\boldsymbol{\omega} = \{\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{b}_1, \boldsymbol{b}_2\}$, and function $g$ is a monotonic non-linear transform (Typically $g$ is either the sigmoid function from equation 3.11 or the hyperbolic tangent). The matrices $\boldsymbol{W}_1, \boldsymbol{W}_2$ connect the input layer to the hidden layer, and the hidden layer to the output layer (respectively), while the bias vectors $\boldsymbol{b}_1, \boldsymbol{b}_2$ are analogous to $\omega_0$ in equations 3.4 and 3.10. For the sake of compactness we will henceforth incorporate the bias terms into the weight matrices and assume the input is extended with a constant term as defined in Equation 3.8. A simple ANN is depicted in Figure 3.2.

The class of MLP's is by no means limited to two layers. We define a neural net with an arbitrary number of hidden layers $l$ by:

$$h_{\boldsymbol{\omega}}(\boldsymbol{x}) = \left(\boldsymbol{g}_{l+1} \circ \boldsymbol{W}_{l+1} \circ \boldsymbol{g}_l \circ \boldsymbol{W}_l \circ ... \circ \boldsymbol{g}_1 \circ \boldsymbol{W}_1\right)(\boldsymbol{x}) \tag{3.13}$$

where $(f \circ g)(x) = f(g(x))$, is a functional composition operator. Note that a weight matrix $\boldsymbol{W}$ is just a linear operator and hence we use $W(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x}$ to denote

[2] The details of the optimization procedure can be found in [45] and [122].

[3] For computational complexity results pertaining to neural networks see [93].

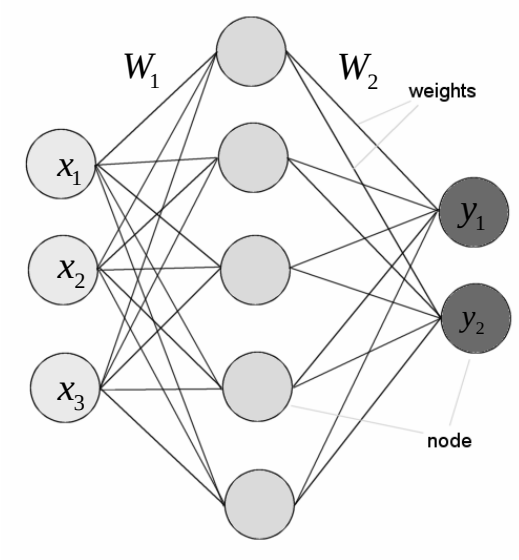



Figure 3.2: Depiction of a standard $3-5-2$ neural network containing 3 inputs, 5 hidden nodes, and 2 outputs. Figure modified from [1].

the matrix-vector product. The parameters, $\boldsymbol{\omega} = \{\boldsymbol{W}_1, ...\boldsymbol{W}_l\}$, can be learned in numerous ways. Here we present the simplest one, namely stochastic gradient descent, for minimizeing the mean squared error loss, $E_{mse}$:

$$E_{mse}(\boldsymbol{y}, \boldsymbol{h}_{\boldsymbol{\omega}}(\boldsymbol{x})) = \|\boldsymbol{y} - \boldsymbol{h}_{\boldsymbol{\omega}}(\boldsymbol{x})\|_{\ell_2}^2 \tag{3.14}$$

where $\|\cdot\|_{\ell_2}$ is the Euclidean norm[4]. The stochastic weight updates are defined by:

$$\boldsymbol{W}_{new}(i,j) = \boldsymbol{W}(i,j) - \eta \frac{\partial E_{mse}(\boldsymbol{y}, \boldsymbol{h}_{\boldsymbol{\omega}}(\boldsymbol{x}))}{\partial \boldsymbol{W}(i,j)} \tag{3.15}$$

where $\boldsymbol{W}(i,j)$ indexes a single matrix element much like $\boldsymbol{I}(i,j)$ indexes a single pixel element, since both are similarly defined on regularly spaced lattices, $\frac{\partial E_{mse}(\boldsymbol{y}, \boldsymbol{h}_{\boldsymbol{\omega}}(\boldsymbol{x}))}{\partial \boldsymbol{W}(i,j)}$ is the partial derivative of the loss function with respect to weight $\boldsymbol{W}(i,j)$, $\eta$ is the learning rate and $\boldsymbol{y}$ is the desired output vector. If the learning rate, $eta = \frac{c}{n}$ is annealed as a function of pattern presentations $n$, then stochastic gradient descent will (provably) converge to a global minima [68]. In practice, such an annealing schedule may be too fast. However, stochastic gradient descent has proven to be the method of choice, when large redundant training sets are utilized.

[4]The Euclidean norm or the $\ell_2$-norm is defined for a $d$-dimensional vector $\boldsymbol{x}$ as: $\|\boldsymbol{x}\|_{\ell_2} = \left(\sum_{i=1}^{d} x(i)^2\right)^{\frac{1}{2}}$. In general, the $\ell_p$-norm is defined as $\|\boldsymbol{x}\|_{\ell_p} = \left(\sum_{i=1}^{d} x(i)^p\right)^{\frac{1}{p}}$.

### 3.3.1 Regularization Techniques

Before describing most common regularization techniques, let us define an entry-wise $\ell_p$-norm for matrix $\boldsymbol{W}$ as:

$$\|\boldsymbol{W}\|_{\ell_p} = \left( \sum_i \sum_j |\boldsymbol{W}(i,j)|^p \right)^{\frac{1}{p}} \tag{3.16}$$

#### $\ell_2$-Regularization

In order to prevent overfitting, the typical approach is to add extra constraints over the entries of the weight matrices $\boldsymbol{W}_l$. Weight decay is often used in conjunction with error minimization by penalizing the Frobenius norm:

$$E(\boldsymbol{y}, \boldsymbol{h}_{\boldsymbol{\omega}}(\boldsymbol{x}), \lambda) = \|\boldsymbol{y} - \boldsymbol{h}_{\boldsymbol{\omega}}(\boldsymbol{x})\|_{\ell_2} - \frac{1}{2}\lambda \|\boldsymbol{\omega}\|_{\ell_2} \tag{3.17}$$

The weight updates are then defined as:

$$\boldsymbol{W}_{new}(i,j) = \boldsymbol{W}(i,j) - \eta \left( \frac{\partial E_{mse}(\boldsymbol{y}, \boldsymbol{h}_{\boldsymbol{\omega}}(\boldsymbol{x}))}{\partial \boldsymbol{W}(i,j)} + \lambda \boldsymbol{W}(i,j) \right) \tag{3.18}$$

Reed et al. [105] prove that using $\ell_2$-regularizaton is equivalent to: (a) training with noisy inputs, (b) convolutional output smoothing, and (c) scaling the non-linear transfer function $\boldsymbol{g}$, to reduce the slope. In all cases the (equivalent) effect is to reduce the magnitude of the weights $\boldsymbol{W}(i,j)$. From a Baysian point of view, $\ell_2$-Regularizaton induces a Gaussian prior over the weights.

#### $\ell_1$-Regularization

Another way to prevent overfitting is $\ell_1$-regularization [125] or weight shrinkage also used in conjunction with error minimization as follows:

$$E(\boldsymbol{y}, \boldsymbol{h}_{\boldsymbol{\omega}}(\boldsymbol{x}), \lambda) = \|\boldsymbol{y} - \boldsymbol{h}_{\boldsymbol{\omega}}(\boldsymbol{x})\|_{\ell_2} - \lambda \|\boldsymbol{\omega}\|_{\ell_1} \tag{3.19}$$

The weight updates are then defined as:

$$\boldsymbol{W}_{new1}(i,j) = \boldsymbol{W}(i,j) - \eta \frac{\partial E_{mse}(\boldsymbol{y}, \boldsymbol{h}_{\boldsymbol{\omega}}(\boldsymbol{x}))}{\partial \boldsymbol{W}_{(i,j)}} \tag{3.20}$$

$$\boldsymbol{W}_{new2}(i,j) = sign(\boldsymbol{W}_{new1}(i,j)) \max(0, \boldsymbol{W}_{new1}(i,j) - \eta\lambda) \tag{3.21}$$

From a Bayesian point of view, $\ell_1$-regularization induces a Laplacian prior over the weights. In contrast to weight decay, $\ell_1$-regularization, can drive the weights completely to zero, rather than simply make their magnitudes small. In turn this produces several interesting phenomena.

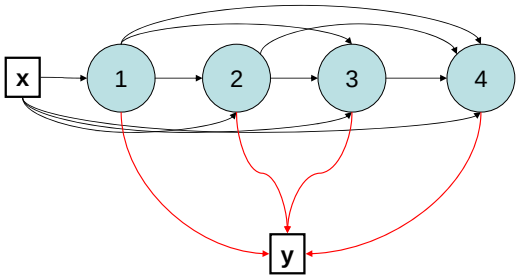



Figure 3.3: An example of a neural network built by cascade correlation algorithm with four hidden nodes.

First and foremost, this type of regularization enables feature selection. If the $\ell_1$-norm of $j^{th}$ column vector is zero (i.e., $\|\boldsymbol{W}(\cdot, j)\|_{\ell_2} = 0$), then all the entries are zero, then the $j^{th}$ feature, $\boldsymbol{x}(j)$, is never used and can therefore be removed. Analogously, the topology of the whole neural network can be controlled by letting $\ell_1$-regularization prune unnecessary nodes in a manner analogous to feature selection. Once again, if a given column vector is zero for a weight matrix $\boldsymbol{W}_l$, the output of the corresponding hidden unit in layer $l-1$ is not used by hidden units at the $l^{th}$ layer.

### 3.3.2 Cascade Correlation

The cascade correlation algorithm is designed to incrementally grow a neural network. It starts with a network containing just one hidden node, and iteratively adds one node at a time to the current architecture. As input, each new node is connected to all base inputs *and* the outputs of all previous nodes. Therefore, each hidden layer contains just one node and there are as many layers as there are nodes. At each iteration of training, only the weights connected to the newly added node are updated.

To address the issues of creating extremely deep and narrow networks, with large fan-in, a modified variant of cascade correlation was proposed in [4]. In this case, the algorithm creates networks with hidden layers containing multiple nodes. At each iterative step, the modified algorithm creates two networks, one with a new unit added to the current layer and another network where the unit is added to a new hidden layer. The network with the higher error after training is discarded and the process repeats it self. Thus, the algorithm either starts a new hidden layer or makes the current hidden layer wider. Empirically, this process creates shallower networks with smaller fan-in while retaining performance analogous to their deep-and-narrow counterparts.

Another modification to cascade correlation proposed removing the "jump con-

nections" linking inputs with deeper hidden layers. These connections create a non-standard network topology making implementation more difficult and increase the fan-in of nodes at deeper layers. To remedy the situation, [98] proposes to create a "strictly layered" topology prohibiting the use of "jump connections" that connect non-adjacent layers. The number of inputs to a given node, the fan-in, is therefore only dependant on the number of nodes at the preceding layer.

## 3.4 Model Fusion

From a Bayesian perspective, a model represents a random draw from some underlying (and unknown) distribution, $p_{\boldsymbol{\omega}}$. This is partly due to samples $\boldsymbol{x}_k$, being random draws from $p_{\boldsymbol{x}}$. Hence, to ensure robustness and generalization accuracy, parameters $\boldsymbol{\omega}$ need to be integrated over using some prior distribution. This, however, is usually intractable and is approximated in practice by learning a set of classifiers $\Omega = \{h_{\boldsymbol{\omega_1}}, ..., h_{\boldsymbol{\omega_n}}\}$, each optimized over a different subset of the training data. The output of each classifier is subsequently merged by uniform averaging as in bagging [15] by:

$$H_\Omega(\boldsymbol{x}) = \frac{1}{n}\sum_{k}^{n} h_{\boldsymbol{\omega_k}}(\boldsymbol{x}) \tag{3.22}$$

To simplify the notation, we will refer to $H_\Omega$ simply as $h$ in the remainder of the thesis.

### 3.4.1 Generalized Pixel Labeling

Using equation 3.22 to model the probability map elements in equation 3.3 we get:

$$\begin{aligned} \boldsymbol{P}(i,j) &= p[\boldsymbol{L}(i,j) = 1 \mid \boldsymbol{f}(i,j)] \\ &= \frac{1}{n}\sum_{k}^{n} h_{\boldsymbol{\omega_k}}(\boldsymbol{f}(i,j)) \\ &= h(\boldsymbol{f}(i,j)) \end{aligned} \tag{3.23}$$

Provided relevant features $\boldsymbol{f}(i,j)$ have been identified, and the chosen machine learning technique, used to build the conditional probability model in equation 3.3, is capable of utilizing the extracted features, the outlined approach can achieve a high pixel classification accuracy. Unfortunately, as depicted in Figure 2.4, even if the method exhibits good generalization performance, objects of the same class that are in close spatial proximity to one another will be merged together into a single connected component. Hence, while the machine learned classifier may have a high pixel classification score, due to the unresolved object-object boundaries (i.e., under-segmentation), the resulting object labeling can still be very poor.

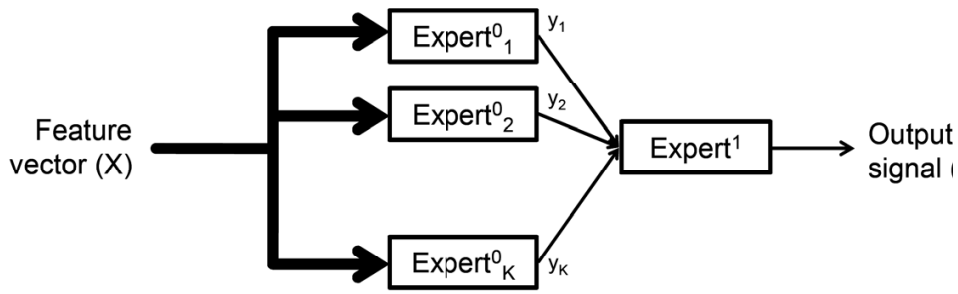



Figure 3.4: Stacked Generalization. Input Feature Vectors are feed into $k$ base classifiers/function approximators. The output of the base classifiers is then feed into another classifier/function approximator to improve accuracy.

### 3.4.2 Stacked Generalization

In [126], Wolpert introduced *stacked generalization*, which utilized the output of several base level ($\mathfrak{L}_0$) classifiers as inputs to the higher level ($\mathfrak{L}_1$) classifier, thereby improving classification accuracy. Figure 3.4 depicts the process. From a different perspective, one can view stacking as learning a gating function to control a mixture-of-experts [59], which in this case are the $\mathfrak{L}_0$ classifiers. The mixture-of-experts algorithms attempt to partition the input space into different regions or categories. In contrast to model averaging, the stacked classifier is trained and can therefore adjust to the peculiarities of the base classifiers. Notice that cascade correlation, presented in the previous section can be viewed as a special case of stacking, designed for neural networks. Under this point of view each new layer learns to fuse the output of the preceding layer(s).

### 3.4.3 Heterogeneous Stacking

Recall the view of stacking as learning a gating function to control a mixture-of-experts, which in turn partition the input space into different regions or categories. An alternate approach presented here is to explicitly partition the output space and subsequently train (a set of) classifiers on each newly created target concept. To combine these **heterogeneous** sources of information, a second set of classifiers is employed, analogous to stacking. To train the $\mathfrak{L}_0$ modules, observe that even simple objects like the rocks presented in Figure 3.1 are not homogeneous, but instead contain several components that can be readily extracted by manipulating the ground truth in a manner analogous to producing $\boldsymbol{L}_{eroded}$ labels. Figure 3.6 presents four label images produced by applying the following morphological operations to the

$$\boldsymbol{I} \mapsto \boldsymbol{f}^{\{0\}} \overset{h^{\{0\}}}{\longmapsto} \boldsymbol{P}^{\{0\}} \mapsto \boldsymbol{f}^{\{1\}} \overset{h^{\{1\}}}{\longmapsto} \boldsymbol{P}^{\{1\}} \mapsto ... \mapsto \boldsymbol{P}^{\{\lambda\}} \mapsto \boldsymbol{f}^{\{ws\}} \overset{ws}{\longmapsto} \boldsymbol{L}^{\{final\}}$$

Figure 3.5: Generic set of mappings describing the process of Heterogenous Stacking with $\lambda + 1$ levels. The last level represents the application of the watershed algorithm, abbreviated as *ws*.

original label image $\boldsymbol{L}$:

$$\boldsymbol{L}_{eroded} = \boldsymbol{L} \ominus B \tag{3.24}$$

$$\boldsymbol{L}_{dilated} = \boldsymbol{L} \oplus B \tag{3.25}$$

$$\boldsymbol{L}_{e'} = \boldsymbol{L} - \boldsymbol{L}_{eroded} \tag{3.26}$$

$$\boldsymbol{L}_{d'} = \boldsymbol{L}_{dilated} - \boldsymbol{L} \tag{3.27}$$

The transformations denote morphological erosion, dilation and two difference operators resembling top-hat and bottom-hat operations. As before, $\boldsymbol{L}_{eroded}$ identifies object markers, while $\boldsymbol{L}_{e'}$ and $\boldsymbol{L}_{d'}$ identify inner and outer object boundaries (respectively). In turn, boundary information indicates where markers and object regions (i.e., $\boldsymbol{L}$) *cannot* be found. Hence, these newly extracted target concepts are complementary to each other and to the original ground truth. Consequently, the $\mathfrak{L}_1$ gating network needs to fuse the output of $\mathfrak{L}_0$ classifiers together rather than select the output of a single base classifier as in de-facto mixtures-of-experts algorithm. From this point of view, the proposed method resembles ensemble learning algorithms, e.g., bagging [15] and boosting [35], which are inherently cooperative in nature. However, these methods introduce diversity into the ensemble by re-sampling the training set as does stacked generalization. In contrast, heterogenous stacking aims to modify the label image $\boldsymbol{L}$ and otherwise keep the training set unchanged. *Random* label flips have been previously explored in [104, 16, 83]. Of course once the i.i.d. assumption has been made, as was done in the aforementioned references, there is nothing more 'intelligent' one can do with the training data other than to try and regularize the learning algorithm via the aforementioned random label permutations. In contrast, image pixels, for any non-trivial domain, are definitively not i.i.d. (c.f., Figure 3.1) and are, therefore amenable to much more interesting label modification schemes. To the best of our knowledge, the research in this thesis and is the first to propose explicit and knowledge directed modification of the ground truth image.

Having defined all target concepts, $\boldsymbol{L}_{type}$, where $type \in \{region, eroded, dilated, e', d'\}$, the corresponding probability maps are created by generalizing equation 3.3 as fol-

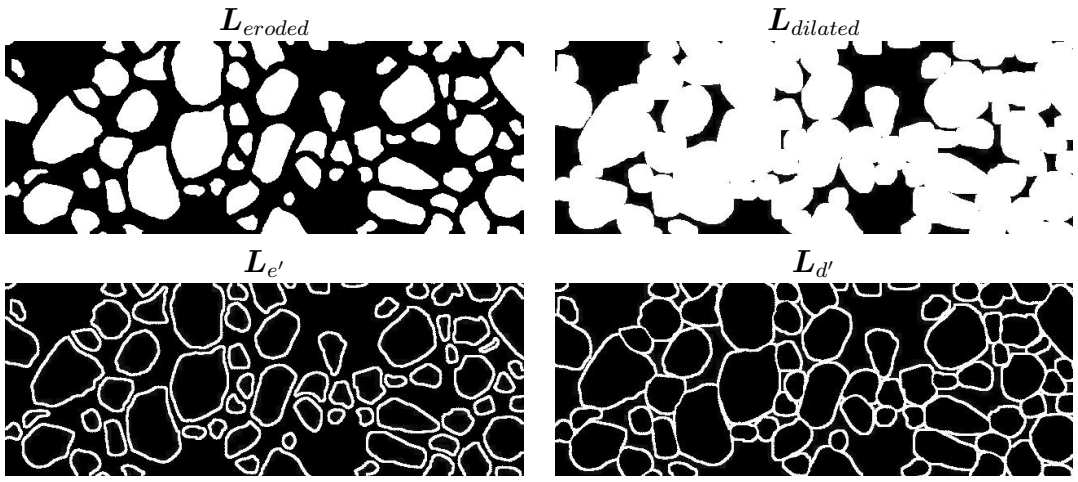



Figure 3.6: New target creation via morphological operations on the original ground truth ($\boldsymbol{L}$). See equations 3.24-3.27 for definitions.

lows:

$$\begin{aligned} \boldsymbol{P}^{\{0\}}_{type}(i,j) &= p[\boldsymbol{L}_{type}(i,j) \mid \boldsymbol{f}^{\{0\}}(i,j)] \\ &= h^{\{0\}}_{type}(\boldsymbol{f}^{\{0\}}(i,j)) \end{aligned} \tag{3.28}$$

Noting that this set of probability maps forms a multi-dimensional image, we simplify the notation by letting $\boldsymbol{P}^{\{0\}} = \{\boldsymbol{P}^{\{0\}}_{type}\}$. Recently, Ting and Witten [119], have empirically demonstrated that using the raw probability maps rather than the thresholded classification labels as input to $\mathfrak{L}_1$ classifier(s) improves performance. As our experimental results will demonstrate, for non i.i.d. data, one can go further and interleave feature extraction with learning to further improve performance. Once again, this effectively allows us to take advantage of the rich domain structure present within images and the resulting probability maps. Consequently the second round of feature extraction can be implemented via the mapping:$\boldsymbol{P}^{\{0\}} \mapsto \boldsymbol{f}^{\{1\}}$, where $\boldsymbol{f}^{\{i\}}$ denotes the $i^{th}$ level of feature extraction. Subsequently the extracted features can be utilized to train a set of $\mathfrak{L}_1$ classifiers $h^{\{1\}}_{type}$. The final labeling $\boldsymbol{L}^{\{final\}}$ can then be produced by creating a topology usable by the watershed algorithm from the probability maps $P^{\{1\}}$ and applying the watershed algorithm. The process was described in the previous section. Within the stacking framework, the topology creation process can be viewed as a feature extraction step mapping $P^{\{1\}} \mapsto \boldsymbol{f}^{\{ws\}}$, while the watershed process can be viewed as an unsupervised classifier. The heterogeneous stacking process can now be succinctly summarized by a sequence of mappings presented in Figure 3.5. Experimental results demonstrating the efficacy of Heterogeneous Stacking are presented in Chapter 6.

## 3.5 Random Field Methods

Despite the apparent computational complexity of mapping $\boldsymbol{I} \mapsto \boldsymbol{L}$, as discussed in the introduction chapter, research into Markov/Conditional/Disciminative Random Field methods [18, 65, 64] has produced a number of heuristic approaches for computing $\boldsymbol{L}$ in a more holistic manner. Following [64], random field methods cast the image segmentation problem as learning a noncausal model of the form:

$$p[\boldsymbol{L}] = \prod_{c \in C} h_c(\boldsymbol{L}_c) \tag{3.29}$$

where $C$ is the set of pixel cliques[5], and $h_c$ are (learned) clique potentials. Within the Markov Random Field (MRF) framework, the posterior over the labels is expressed using Bayes' rule as:

$$p[\boldsymbol{L}|\boldsymbol{I}] \propto p[\boldsymbol{L}, \boldsymbol{I}] = p[\boldsymbol{L}]p[\boldsymbol{I}|\boldsymbol{L}] \tag{3.30}$$

The generative MRF model, aims to learn the joint probability of the data and labels, $p[\boldsymbol{L}, \boldsymbol{I}]$, by modeling the prior, $p[\boldsymbol{L}]$ as an MRF (usually taken to be a homogeneous isotropic Ising model) and assumes that the data likelihood term has a factorized form:

$$p[\boldsymbol{I}|\boldsymbol{L}] = \prod_{(i,j) \in S} p[\boldsymbol{I}(i,j)|\boldsymbol{L}(i,j)] \tag{3.31}$$

In turn, this implies that the image pixels are conditionally independent from each other, given the corresponding labels. Using the Hammersley-Clifford Theory [8] and the above assumption allows the density of the MRF to be modeled as a Gibbs distribution:

$$p[\boldsymbol{L}] = \frac{1}{Z} exp\left(\sum_{c \in C} h_c(\boldsymbol{L}_c)\right)$$

where $Z$ is the partition function. To incorporate data and spatial dependencies the posterior MRF is modeled as:

$$p[\boldsymbol{L}|\boldsymbol{I}] =$$

$$\frac{1}{Z} \exp\left[\sum_{(i,j) \in S} \underbrace{\ln\left(p[\boldsymbol{f}^{\boldsymbol{I}}(i,j)|L(i,j)]\right)}_{\text{association potential}} + \beta \sum_{(i,j) \in S} \left(\sum_{(i',j') \in \partial(i,j)} \underbrace{\boldsymbol{L}(i,j)\boldsymbol{L}(i',j')}_{\text{interaction potential}}\right)\right] \tag{3.32}$$

where $\beta$ is the interaction parameter.

[5] Recall, a clique is a fully connected subgraph of the original graph as discussed in the previous chapter.

To generalize Equation 3.32, Conditional/Discriminative random field methods incorporate additional data terms for modeling the posterior. For example, the DRF model in [64] is given by:

$$p[\boldsymbol{L}|\boldsymbol{I}] =$$

$$\frac{1}{Z}\exp\left[\sum_{(i,j)\in S}\underbrace{h_{\boldsymbol{\omega}_1}(\boldsymbol{L}(i,j),\boldsymbol{f}^{\boldsymbol{I}}(i,j))}_{\text{association potential}}+\sum_{(i,j)\in S}\left(\sum_{(i',j')\in\partial(i,j)}\underbrace{h_{\boldsymbol{\omega}_2}(\boldsymbol{L}(i,j),\boldsymbol{L}(i',j'),\boldsymbol{f}^{\boldsymbol{I}}(i,j))}_{\text{interaction potential}}\right)\right] \tag{3.33}$$

where both the associative and interaction potentials incorporate the image features. This enables abrupt transitions (i.e., edges) to be modeled. Parameter estimation can be carried out in many ways, although usually the pseudo-likelihood maximization is used (see [64] for details). Inference (i.e., on-line computation of the posterior for a test input) can be done in many ways however, empirical results in [33] demonstrated that using the Iterated Conditional Modes (ICM) yields reasonable performance. Since it is difficult to maximize the joint probability of an MRF, [9] proposed the deterministic ICM algorithm that sequentially maximizes local conditional probabilities. Given an initial estimate $\boldsymbol{L}_0$ of $\boldsymbol{L}$, the algorithm iterates over:

$$h_{ICM}:[\boldsymbol{f}^{\boldsymbol{I}}(i,j),\boldsymbol{f}^{\boldsymbol{L}_{d-1}}(i,j)]\mapsto\boldsymbol{L}_d(i,j) \tag{3.34}$$

where $\boldsymbol{f}^{\boldsymbol{L}_{d-1}}(i,j)$ is the set of label values from lattice cites neighboring location $(i,j)$ determined during previous iteration $d-1$. Formally, $\boldsymbol{f}^{\boldsymbol{L}_{d-1}}(i,j) = \{L(i',j')|(i',j')\in\partial(i,j)\}$. The initial estimate $\boldsymbol{L}_0$ is usually produced by maximizing the association potential. The algorithm stops once some predefined convergence criteria is satisfied, which is usually based on the difference between $\boldsymbol{L}_{d-1}$ and $\boldsymbol{L}_d$.

While the random field models significantly improve on the basic model from Equation 3.1, where image pixels (and labels) are unconditionally independent, these models have never-the-less been found to be overly restrictive in practice. Specifically, while the MRF/CRF model allows pairwise label interaction, analysis of higher order interactions (see Figure 2.1) quickly becomes intractable. Furthermore, exact computation of the partition function, $Z$ is also usually intractable[6] and in practice is approximated using sampling methods. Employing these 'short-cuts', still makes random field models difficult to train and apply in practice. More recently, higher-order random fields have been developed in [109]. These are briefly discussed in the next chapter. In addition, to eliminate the costly computation of the partition function $Z$, energy minimization based methods have been proposed [70], that learn raw potential function and do not require normalizing terms.

[6] In general computing the partition function, $Z$, is NP-hard [37].

# Chapter 4

# Automated Feature Extraction and Relevant Applications

This chapter focuses on the feature extraction function $\boldsymbol{f}^{\boldsymbol{I}}(i,j)$. The typical goal of feature extraction is the identification of information relevant to a particular task at hand. In contrast to the manual feature design coupled with feature selection, this chapter focuses on fully automated methods for feature extraction and a few relevant applications (e.g., image denoising). These approaches remove the need for manual feature creation altogether, by learning a set of (possibly non-linear) filters, that can extract 'interesting' aspects of a given image.

## 4.1 Neighborhood Analysis

Recall our initial assumption that a pixel $\boldsymbol{I}(i,j)$ does not contain enough information about the label $\boldsymbol{L}(i,j)$ and we must therefore (at the very least) examine neighboring pixels. Let $\boldsymbol{N}_r(i,j)$ denote a symmetric neighborhood, corresponding to an $(2r+1) \times (2r+1)$ subimage centered at, and including, location $(i,j)$[1]. Therefore, a square image patch $\boldsymbol{N}_r(i,j)$ has dimensions $(2r+1) \times (2r+1)$ and contains $(2r+1)^2 = 4r^2 + 4r + 1$ pixels. The neighborhood of pixel $(i,j)$, defines the most basic set of features, $\boldsymbol{f}(i,j) = vec(\boldsymbol{N}_r(i,j))$, for describing the properties of a given pixel. Let $\boldsymbol{x} = vec(\boldsymbol{N}_r(\boldsymbol{I}(i,j)))$, be a column vector representing a vectorized image patch extracted from $\boldsymbol{I}$. Now we can define the data matrix $\boldsymbol{X}_0$ as in Equation 3.7, with each row being a sample of $\boldsymbol{N}_r(\cdot)$ and each column of $\boldsymbol{X}_0$ corresponding to a different feature, and representing one neighbor of $(i,j)$. The centered data matrix $\boldsymbol{X}$ can then be calculated as in Equation 3.8 by subtracting the mean feature vector from each row of $\boldsymbol{X}_0$.

## 4.2 Linear Algebra, SVD and Eigenvalue Decomposition

Following [47], let us define the $\ell_p$ of the vector $\boldsymbol{x} \in \mathbb{R}^n$ by

$$\|\boldsymbol{x}\|_{\ell_p} = \left( \sum_{i=1}^{n} \boldsymbol{x}(i)^p \right)^{\frac{1}{p}} \tag{4.1}$$

where $\boldsymbol{x}(i)$ is the $i^{th}$ component of vector $\boldsymbol{x}$. Then given an $m \times n$ matrix $\boldsymbol{A} \in \mathbb{R}^{n \times m}$, the matrix norm induced by the vector norm in 4.1 is:

$$\|\boldsymbol{A}\|_{\ell_p} = \max_{\boldsymbol{x} \neq 0} \frac{\|\boldsymbol{A}\boldsymbol{x}\|_{\ell_p}}{\|\boldsymbol{x}\|_{\ell_p}} \tag{4.2}$$

[1] To prevent border effects the images are padded based on $r$.

It is well known that the $\ell_2$ norm of a matrix corresponds to the largest singular value $\sigma_{max}$, where the singular value decomposition is given by $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}$ and the matrix $\boldsymbol{\Sigma}$ is an $m \times n$ diagonal matrix with:

$$\sigma_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ \sigma_i \geq 0 & \text{if } i = j \end{cases}$$

Furthermore, the set of singular values, $\sigma(\boldsymbol{A}) = \{\sigma_i\}$, are related to the eigenvalues $\lambda(\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}) = \{\lambda_i\}$, by $\sigma_i = \sqrt[+]{\lambda_i}$, which are the solutions to the eigenvalue/eigenvector problem:

$$\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x} \tag{4.3}$$

for an $n \times n$ matrix $\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}$. The spectrum of $\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}$ is given by $\lambda(A^T A)$, while the spectral radius of $\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}$ is given by $\rho(\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}) = max\ \lambda(\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A})$. For now let us consider a simpler case of a square $n \times n$ matrix $\boldsymbol{A}$ and the corresponding eigenvalue/eigenvector decomposition:

$$\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x} \tag{4.4}$$

The eigenvectors of $\boldsymbol{A}$ describe directions where the effect of the linear transformation given by $\boldsymbol{A}$ is simple to understand. Given a vector $\boldsymbol{x}$ solving equation 4.4, the transformation $\boldsymbol{A}\boldsymbol{x}$ simply multiplies $\boldsymbol{x}$ by eigenvalue $\lambda$. Hence SVD and eigenvector/eigenvalue decompositions provide the means for understanding the behavior of a general linear transformation $\boldsymbol{A}$.

## 4.3 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) [62] is a multivariate statistical technique that calculates an orthogonal basis describing the variability in data. For zero mean data, if two features are orthogonal, then they are uncorrelated. If $E\{\cdot\}$ denotes expectation, then two (zero mean) variables are (linearly) uncorrelated if their covariance, $C_{x,y}$, is zero:

$$C_{x,y} = E\{xy\} = 0 \tag{4.5}$$

By projecting the original correlated variables onto the PCA basis, a new set of uncorrelated variables can be produced, that are linear combinations of the original variables. The basis vectors (a.k.a. principal components) thus represent the major (orthogonal) directions of variability in the dataset and are described by the eigenvectors of $\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}$, where $\boldsymbol{X}$ is the (mean normalized) dataset in matrix form, with data samples represented by rows and each column representing a variable.

More specifically, PCA performs an eigenspace projection, which is calculated by identifying the eigenvectors of the covariance matrix derived from a set of (training) data, denoted as $\boldsymbol{X}$. The eigenvectors corresponding to non-zero eigenvalues

of the covariance matrix form an orthonormal basis that rotates and/or reflects the data in the $N$-dimensional space. The covariance matrix is then computed as:

$$\boldsymbol{C_X} = \boldsymbol{X}^T \boldsymbol{X} \tag{4.6}$$

The eigenvectors of $\boldsymbol{C_X}$, given by the solution of either Equation 4.3 or 4.4, describe the principal directions of variability in the data set $\boldsymbol{X}$. In the case of image processing, data samples $\boldsymbol{x}_i$ represent vectorized image patches, usually sampled at random from a set of images. In turn, the principal components of this data set represent a set of (vectorized) linear, orthogonal filters that can be used to extract a set of features describing an image patch.

From a historical perspective, SVD was independently derived by Beltrami in 1873 and Jordan in 1874. The earliest description of PCA is given by Pearson in 1901 and Hotelling in 1933. Since then it has become a fundamental analytical technique in numerous fields and is the basis of numerous other algorithms, including canonical correlation analysis and partial least squares. PCA for computer vision and image processing was first pioneered by Kirby and Sirvovich [116], who showed that face images (registered and intensity normalized) comprised of thousands of pixels can be characterized by just the first few eigenvectors (corresponding to the eigenvalues with largest magnitudes). In [97] the approach was refined to multiple subspaces each characterizing a facial feature. In essence the researchers manually split the face into its constituent parts such as the eyes, mouth, nose, and created a separate subspace for each part. By factoring the face into constituents, face recognition was made more robust and required even fewer principal components (in total) than the original PCA method. In general, PCA can effectively preprocess the data for use by linear and logistic regression[2].

### 4.3.1 Linear Autoencoders

Like the rest of the automated feature extraction algorithms presented in this chapter, PCA can be reformulated as a special type of neural network. In this case the (column) eigenvectors can be grouped/formed into a weight matrix $\boldsymbol{W}_1$ and learned by the following network topology:

$$\hat{\boldsymbol{x}} = (\boldsymbol{W}_2 \circ \boldsymbol{W}_1)\, \boldsymbol{x} \tag{4.7}$$

In order to have convergence to the true PCA eigenvectors, the weight matrices must be constrained to be orthonormal and $\boldsymbol{W}_2 = \boldsymbol{W}_1^T$. Notice that the features of the input vector, $\boldsymbol{x}$, can be extracted as $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{W_1 x}$ and conversely, the original input vector can be reconstructed via $\hat{\boldsymbol{x}} = \boldsymbol{W_2 f}(\boldsymbol{x})$. Thus, the autoencoder

[2]The use of PCA with linear regression is sometimes called PCR for principal component regression.

for learning PCA parameters is a linear network that constrains the weight matrices such that the norm of each (eigenvector) is unity and that the 'feature extraction matrix', $\boldsymbol{W}_1$ is equivalent to the transpose of 'output synthesis' matrix, $\boldsymbol{W}_2$, i.e., $\boldsymbol{W}_1 = \boldsymbol{W}_2^T$. The network in Equation 4.7 defines what is known as a linear autoencoder, which can be used to approximate the closed form PCA solution. In most cases, the closed form eigenvector decomposition is usually preferred over the autoencoder approach, due to the possibility of estimation errors that may occur as a result of performing gradient descent. Nevertheless, if online gradient descent is used to learn the weight matrices, the amount of data an autoencoder can process is no longer bounded by the amount of RAM, as in the case of the eigenvector decomposition. Furthermore, if the problem at hand is non-stationary, the online autoencoder can continuously adjust to the changing environment by readjusting the eigenvectors stored as weight matrices.

## 4.4 Independent Component Analysis (ICA)

Like PCA, independent component analysis [57] is statistical technique for analysis of multivariate data. Both PCA and ICA try to discover hidden factors underlying the data set. In contrast to PCA, which performs an analysis of the covariance matrix, ICA goes further and analyzes higher order statistics, typically kurtosis, in order to find independent rather that simply decorrelated components. A well known fact in statistics is that if two Gaussian distributions are uncorrelated, then they are independent. That implies that PCA, which finds an orthogonal and hence decorrelated basis for the data set, is optimal for factors that have Gaussian distributions. On the other hand, ICA aims to uncover factors (also referred to as sources) that are both non-gaussian and statistically independent.

Strictly speaking let $\boldsymbol{x}, \boldsymbol{y}$ be observations of random variables $x, y$, represented as vectors. Then

**$\boldsymbol{x}, \boldsymbol{y}$ are linearly independent** If $\nexists a \in \mathbb{R}$ such that $a\boldsymbol{x} = \boldsymbol{y}$ and $\boldsymbol{x} \neq \mathbf{0}, \boldsymbol{y} \neq \mathbf{0}$, where $\mathbf{0}$ is a vector of zeros.

**$\boldsymbol{x}, \boldsymbol{y}$ are orthogonal** If $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x}^T \boldsymbol{y} = 0$, i.e., their dot-product is zero.

**$\boldsymbol{x}, \boldsymbol{y}$ are uncorrelated** If $\langle \boldsymbol{x} - \mu_{\boldsymbol{x}} \mathbf{1}, \boldsymbol{y} - \mu_{\boldsymbol{x}} \mathbf{1} \rangle = 0$, where $\mu_{\boldsymbol{x}}, \mu_{\boldsymbol{y}}$ are the means, and $\mathbf{1}$ is a vector of ones.

The relationship between uncorrelated, orthogonal and linearly independent is illustrated in Figure 4.1. The concept of statistical independence is different from linear independence in linear algebra. From a statistical point of view, if two zero-mean random variables are linearly independent then they are uncorrelated. Note that this

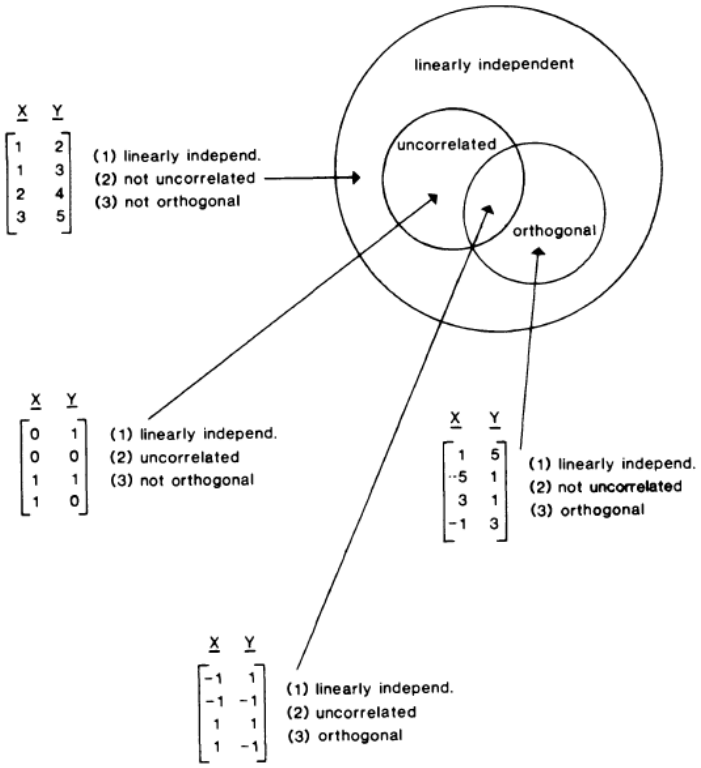



Figure 4.1: Visualization of the relationship between uncorrelated, orthogonal and linearly independent variables. From [107].

only holds for zero mean variables. (See [107] for additional discussion.) However, linear independence does not imply statistical independence. Two random variables are statistically independent iff:

$$p(x, y) = p(x)p(y) \tag{4.8}$$

That is, the joint density, $p(x, y)$, must factor into a product of marginal densities $p(x)$ and $p(y)$ [54]. It can be shown that two independent random variables $x,\ y$, satisfy:

$$E\{g(x)h(y)\} = E\{g(x)\}E\{h(y)\} \tag{4.9}$$

where $g(x),\ h(y)$ are any absolutely integrable functions. By letting $g(x) = x,\ h(y) = y$, equation 4.5 is obtained as a special case of 4.9. Another well known fact [54] is that if two uncorrelated random variables are Gaussian, then they are independent. This is due to the fact that a product of two Gaussian distributions is still a Gaussian, and a distribution can be described by its moments, $\{m_i\}_{i=0}^{\infty}$. However, the third and higher moments of a Gaussian distribution are zero. Hence, if $x$ and $y$ are

uncorrelated Gaussian random variables:

$$\begin{aligned} E\{m_i(x)m_j(y)\} &= E\{g(x)h(y)\} \\ &= 0 \\ &= E\{g(x)\}E\{h(y)\} \\ &= E\{m_i(x)\}E\{m_j(y)\},\ \forall i,j \geq 3 \end{aligned}$$

where $m_i(\cdot)$ is the $i^{th}$ moment. Thus PCA is optimal for linear mixtures of Gaussian variables. In other words, if the latent factors are gaussian, only the covariance of the data matrix needs to be analyzed. However, most real world data including natural images, are **not** composed of linear, Gaussian mixtures, rather the distributions are very leptokurtic, i.e., the typical components comprising real world data are super-Gaussian, having a higher kurtosis than a Gaussian random variable with the same mean and variance (see Figure 4.3 for examples). Kurtosis ("excess") is defined as $\kappa(x) = \frac{E\{x^4\}}{[E\{x^2\}]^2} - 3$, and is zero for a Gaussian distribution. Therefore, to find the true latent factors, independent components need to be identified.

The basic model for ICA assumes that a number of unknown (non-gaussian, and statistically independent) sources $\boldsymbol{s}$ are **linearly** mixed by a mixing matrix $\boldsymbol{A}$ into a set of observed mixtures $\boldsymbol{x}$:

$$\boldsymbol{x} = \boldsymbol{A}\boldsymbol{s} \tag{4.10}$$

Conversely, to identify the sources $\boldsymbol{s}$, ICA computes a decomposition matrix $\boldsymbol{A}^\dagger$ such that:

$$\boldsymbol{s} = \boldsymbol{A}^\dagger\boldsymbol{x} \tag{4.11}$$

where $\boldsymbol{A}^\dagger = \boldsymbol{A}^{-1}$ if the mixing and decomposition matrices are square and non-singular; otherwise it is the pseudo-inverse of $\boldsymbol{A}$. To uncover the independent components, the data is first whitened/sphered in order to remove first and second order statistical dependencies. This step can be easily done by using PCA. It can be shown that by projecting the data onto normalized eigenvectors, the covariance of the new data is set to identity. The second (iterative) step of ICA is to minimize fourth order dependencies by finding a set of projections that mutually maximize one of many non-linear contrast functions measuring the degree of sparseness or independence between the projections. Research into Independent Component Analysis (ICA) has produced a myriad of algorithms [54] capable of extracting linear, non-gaussian components and can be readily applied to the problem at hand. Most of the algorithms are iterative, and are based on maximizing a contrast function measuring the degree of non-Gaussianity. Example measures of non-Gaussianity include: kurtosis, entropy, Fisher score, and many others [54].

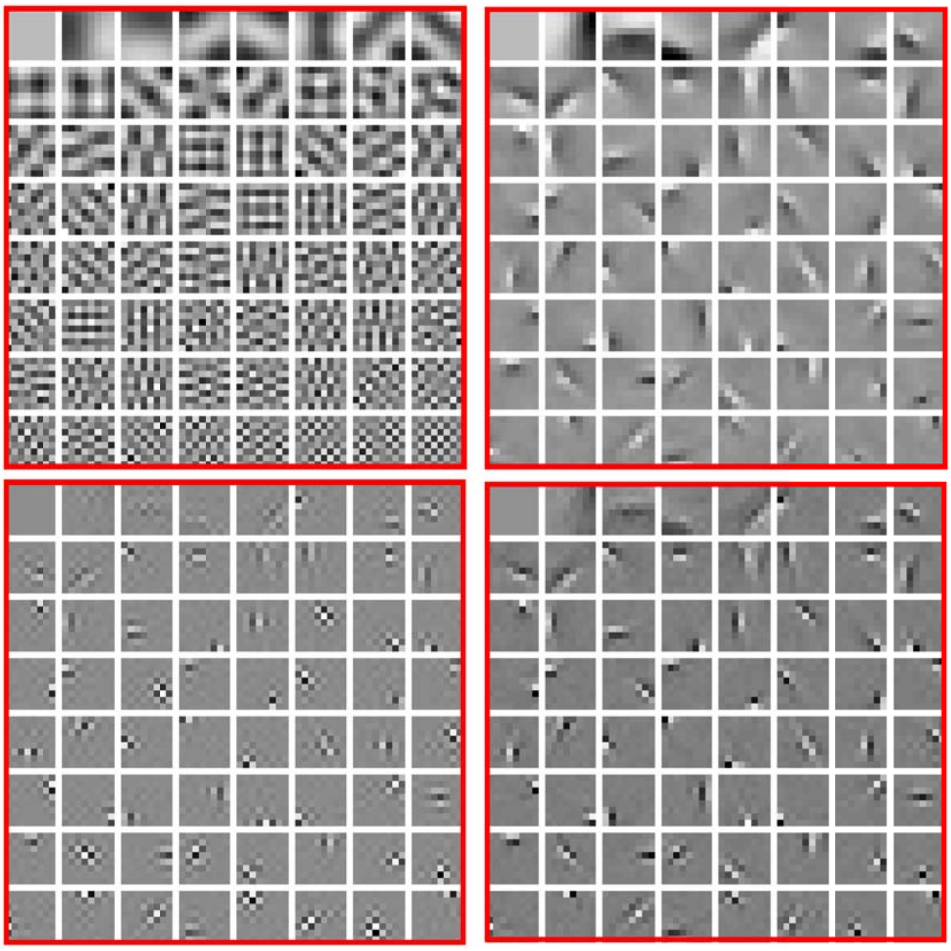


Figure 4.2: Visualization of $8 \times 8$ filters. **Top Left:** PCA basis, **Top Right:** ICA basis ($\grave{\boldsymbol{a}}_\alpha$),**Bottom Left:** Analogously reshaped ICA filters ($\grave{\boldsymbol{a}}^\dagger_\alpha$), **Bottom Right:** orthogonalized ICA with filters and basis being identical, as in the case of PCA.

### 4.4.1 Convolutional Encoding/Decoding of Images

This section describes an efficient implementation of ICA based image encoding/decoding and is extensively used in the remainder of the dissertation. Let $\boldsymbol{x} = vec(\boldsymbol{N}^{\boldsymbol{I}}_r(i,j))$ be a column vector representing a vectorized image patch extracted from $\boldsymbol{I}$ and $\boldsymbol{X}$ be a centered matrix of samples.

Once the matrix $\boldsymbol{A}^\dagger$ has been learned from $\boldsymbol{X}$, features can be efficiently extracted by reshaping the rows of $\boldsymbol{A}^\dagger$ into filters, rotating the filters by $180°$, and subsequently convolving an input image with the newly created filter bank[3]. (Alternatively one can forego the $180°$ rotation and use spatial correlation instead of

[3]Typically the input image is z-scaled (normalized) by subtracting the mean and dividing by the standard deviation. Furthermore, the local mean is then subtracted from each $n \times n$ patch. The local mean normalization can be efficiently implemented via convolution as well.

convolution.) We denote by $\grave{\boldsymbol{a}}_\alpha^\dagger$ the filters created from $\boldsymbol{A}^\dagger$. The set of filters is denoted by $\Phi = \{\grave{\boldsymbol{a}}_1^\dagger, ..., \grave{\boldsymbol{a}}_k^\dagger\}$. Hence, for a given image $\boldsymbol{I}$, the feature maps $\boldsymbol{f}_\alpha^{\boldsymbol{I}}$ can be produced via convolution by:

$$\boldsymbol{f}_\alpha^{\boldsymbol{I}} = \boldsymbol{I} * \grave{\boldsymbol{a}}_\alpha^\dagger \tag{4.12}$$

The feature vector $\boldsymbol{f}^{\boldsymbol{I}}(i,j) = \boldsymbol{s}(i,j)$ is the set of latent variables describing the $n \times n$ pixel neighborhood centered at site $(i,j)$ (i.e., an $N_r(i,j)$ patch where $n = (2r+1)$).

Furthermore, given a set of feature maps, $\boldsymbol{f} = \{\boldsymbol{f}_\alpha\}_{\alpha=1}^k$, the original image can be also efficiently reconstructed using convolution. Before presenting convolutional decoding, a few preliminaries are in order. Observe that the $N_r(i,j)$ patch is generated by reshaping the vector output of $\boldsymbol{A}\boldsymbol{s}(i,j)$ into an $n \times n$ patch, where $n = 2r+1$. Let $\alpha$ index the individual components of $\boldsymbol{s} = [s_1, ..., s_k]$, and drop spatial indices $(i,j)$. Then $\boldsymbol{A}\boldsymbol{s} = \boldsymbol{A}_1 s_1 + ... + \boldsymbol{A}_k s_k = \sum_\alpha \boldsymbol{A}_\alpha s_\alpha$, where $\boldsymbol{A}_\alpha$ is a $\alpha^{th}$ column vector. Let $\Psi = [\grave{\boldsymbol{a}}_1, ..., \grave{\boldsymbol{a}}_k]$ be the set of filters created from the columns of matrix $\boldsymbol{A}$. Then

$$\widehat{N_r(i,j)} = \sum_\alpha^k \grave{\boldsymbol{a}}_\alpha s_\alpha(i,j)$$

represents the reconstruction of $N_r(i,j)$. Intuitively, if $(i',j') \in \partial(i,j)$, then patches $N_r(i',j')$ and $N_r(i,j)$ overlap. To produce the final reconstructed image $\hat{\boldsymbol{I}}$, the method of **sliding windows**, averages out the output for all the patches overlapping $(i,j)$. For interior pixels, there are $n^2 = (2r+1)^2$ overlapping patches that are averaged over to produce the final output. A straightforward implementation of ICA based image reconstruction would involve starting with $\hat{\boldsymbol{I}}(i,j) = 0,\ \forall (i,j) \in S$. Then adding the output of each patch to $\hat{\boldsymbol{I}}(i,j)$ one at a time, as

$$N_r^{\hat{\boldsymbol{I}}}(i,j) = N_r^{\hat{\boldsymbol{I}}}(i,j) + \sum_\alpha^k \grave{\boldsymbol{a}}_\alpha s_\alpha(i,j)\ \forall (i,j) \in S$$

The final step would be to normalize $\hat{\boldsymbol{I}}$ based on the number of patches that overlap $(i,j)$. Rather than multiplying $\grave{\boldsymbol{a}}_\alpha$ with a scalar $\boldsymbol{f}_\alpha = s_\alpha(i,j)$, convolutional decoding convolves the spatial feature map $\boldsymbol{f}_\alpha$ with the filter $\grave{\boldsymbol{a}}_\alpha$[4]. In this implementation, the output is given by:

$$\hat{\boldsymbol{I}} = \boldsymbol{f} * \boldsymbol{\Psi} = \sum_\alpha^k \boldsymbol{f}_\alpha * \grave{\boldsymbol{a}}_\alpha \tag{4.13}$$

where $\boldsymbol{f} * \boldsymbol{\Psi}$, denotes tensor based convolution. Once again the final step is to normalize each location $(i,j)$ based on the number of overlapping patches. In order

[4]Equivalently, correlation can be used: $\hat{\boldsymbol{I}} = \sum_\alpha^k \boldsymbol{f}_\alpha \star \grave{\boldsymbol{a}}_\alpha^-$, where $\star$ denotes spatial correlation and $\grave{\boldsymbol{a}}^-$ is a filter rotated $180°$.

to understand convolutional decoding let us consider a single $3 \times 3$ feature map $\boldsymbol{f}$ and a $3 \times 3$ filter $\grave{\boldsymbol{a}}$:

$$\boldsymbol{f} = \begin{array}{|c|c|c|}\hline f_{1,1} & f_{1,2} & f_{1,3} \\ \hline f_{2,1} & f_{2,2} & f_{2,3} \\ \hline f_{3,1} & f_{3,2} & f_{3,3} \\ \hline \end{array} \quad \grave{\boldsymbol{a}} = \begin{array}{|c|c|c|}\hline a_{1,1} & a_{1,2} & a_{1,3} \\ \hline a_{2,1} & a_{2,2} & a_{2,3} \\ \hline a_{3,1} & a_{3,2} & a_{3,3} \\ \hline \end{array} \quad \grave{\boldsymbol{a}}^{-} = \begin{array}{|c|c|c|}\hline a_{3,3} & a_{3,2} & a_{3,1} \\ \hline a_{2,3} & a_{2,2} & a_{2,1} \\ \hline a_{1,3} & a_{1,2} & a_{1,1} \\ \hline \end{array}$$

Next consider the contribution to $\hat{\boldsymbol{I}}(2,2)$ made when filter $\grave{\boldsymbol{a}}$ is positioned over location $(1,1)$, which is $f_{1,1}a_{3,3}$. Recall that location $(1,1)$ produces as output a patch $N_1(1,1) = f(1,1)\grave{\boldsymbol{a}}$ the bottom left location of $N_1(1,1)$ corresponds to location $\hat{\boldsymbol{I}}(2,2)$. The full contribution is then $\hat{\boldsymbol{I}}(2,2) = f_{1,1}a_{3,3} + f_{1,2}a_{3,2} + f_{1,3}a_{3,1} + ... + f_{3,1}a_{1,3} + f_{3,2}a_{1,2} + f_{3,3}a_{1,1}$. But this is precisely the convolution with $\grave{\boldsymbol{a}}$ and, equivalently spatial correlation with $\grave{\boldsymbol{a}}^{-}$, which is $\grave{\boldsymbol{a}}$ rotated $180^{\circ}$ about the center pixel. Hence, **feature extraction** and **output synthesis** can be efficiently performed via convolutional operations based on Equations 4.12 and 4.13. This approach will is utilized for a number of algorithms presented in the remainder of this chapter and is also the fundamental building block of Stacked Convolutional Regression. Additional insight can be gained by observing that averaging over multiple overlapping patch outputs is similar to model averaging presented previously. In this case, the output for a given location (i,j) is given under different context. Each neighbor $(i', j') \in \partial(i,j)$ proposes its own estimate for location $(i,j)$ which is subsequently integrated with other estimates into a single output. One immediate observation is that PCA/ICA algorithms inherently ignore interactions between overlapping patches, I.e., these algorithms implicitly consider image patches independent of one another. Clearly this is an incorrect assumption, which several algorithms try to address. These are discussed in the context of High Order Random Field Methods at the end of the chapter.

### 4.4.2 ICA based Neural Networks

As in the case of PCA, a single hidden layer autoencoder can be used for learning ICA parameters. Letting the first weight matrix equal the de-mixing (feature extraction) matrix $\boldsymbol{A}^{\dagger} = \boldsymbol{W}_1$ and the reconstruction matrix equal to the second weight matrix, $\boldsymbol{A} = \boldsymbol{W}_2$, we get an autoencoder neural network:

$$\begin{aligned} \hat{\boldsymbol{x}} = & \quad (\boldsymbol{W}_2 \circ \boldsymbol{W}_1)\,\boldsymbol{x} && (4.14) \\ = & \quad \boldsymbol{W}_2(\boldsymbol{W}_1\boldsymbol{x}) && (4.15) \\ = & \quad \boldsymbol{W}_2\boldsymbol{s} && (4.16) \end{aligned}$$

As previously, the first layer of this neural net extracts latent components $\boldsymbol{s}$, from the input vector $\boldsymbol{x}$, while the second layer reconstructs the input vector $\hat{\boldsymbol{x}}$, from the latent components. The constraint of $\boldsymbol{W}_2 = \boldsymbol{W}_1^{\dagger}$ is still required. Hence, we could

try to learn the mixing/demixing matrices by minimizing the reconstruction error, $MSE(\boldsymbol{x}, \hat{\boldsymbol{x}})$. Unfortunately this approach can only produce a representation of the data congruent to PCA. To produce ICA-like components we need to introduce a non-linearity, $\boldsymbol{g}_1$, to capture higher order moments of the input matrix (see [57] for details).

$$\hat{\boldsymbol{x}} = (\boldsymbol{W}_2 \circ \boldsymbol{g}_1 \circ \boldsymbol{W}_1)\, \boldsymbol{x}$$

Later sections of this chapter further discuss the link between neural nets and PCA/ICA. For now, observe that once learned, this type of neural network also acts as a bank of filters that can be applied to an image. The first set of weights $\boldsymbol{W}_1$ extracts a set of feature maps, $\boldsymbol{f} = \{\boldsymbol{f}_\alpha\}$, while the second set of weights, once properly reshaped, is used to synthesize the approximation to the initial input. I.e., Equations 4.12 and 4.13 can be used with neural networks as well.

To briefly review, in contrast to using a monolithic set of hand-crafted features, ICA (like PCA) learns a new feature extraction matrix $\boldsymbol{A}^\dagger$ for each new domain in an unsupervised and automated way. Furthermore, the features are independent of one another, resulting in improved estimates of linear and logistic regression parameters $\boldsymbol{\omega}$ during the learning stage. Conveniently, since the rows (resp. columns) of $\boldsymbol{A}^\dagger$ (resp. $\boldsymbol{A}$) can be reshaped into image patches, the learned filters are can be readily visualized as in Figure 4.2, which depicts the filters learned by PCA and ICA used for extracting orthogonal and independent components. Furthermore, filters learned by ICA resemble Gabor filters, and more importantly respond to stimuli in a manner similar to cells found in V1, an area of the brain responsible for initial processing of visual information [92, 120]. Experimental results utilizing ICA, in conjunction with heterogenous stacking (c.f., Section 3.4.3), are presented in Chapter 6.

### 4.4.3 Sparse Code Shrinkage for (Image) Denoising

A simple method for image de-noising via ICA uses a set of "clean" images to learn the independent components. Subsequently, when a noisy image is projected onto the ICA basis, the resulting independent components can be shrunk (towards zero) to reduce noise. The (optimal) shrinkage function depends on the distribution of the independent component as shown in Figure 4.3.
The typical noise model used is:

$$y = s + v \tag{4.17}$$

where $y$ is the noisy observation, $s$ is the signal corresponding to the true independent component, and $v \sim \mathcal{N}(0, \sigma)$ is zero mean Gaussian noise with variance $\sigma^2$. In [55] the following denoising functions were proposed:

$$\widehat{s} = y - \sigma^2 f'(y) \tag{4.18}$$

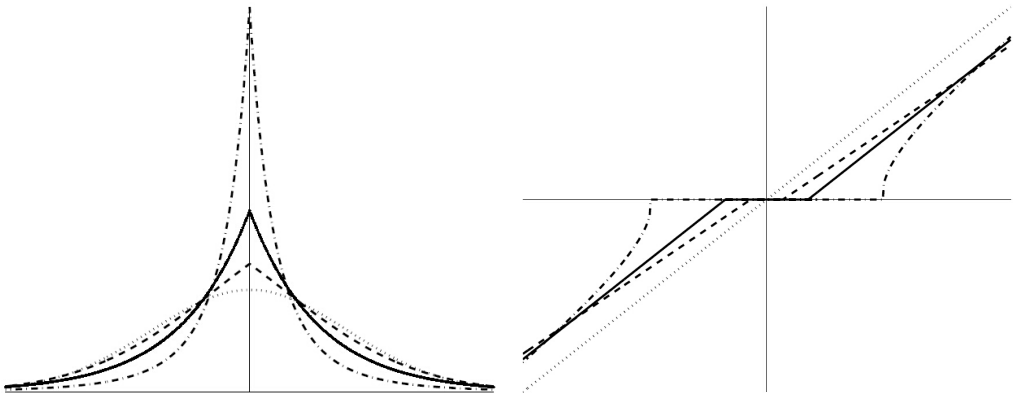


Figure 4.3: **Left:** Plots of densities corresponding to different models of sparse components. *Solid line:* Laplace density. *Dashed line:* a typical moderately sparse(super-Gaussian) density. *Dash-dotted line:* a typical strongly sparse(super-Gaussian) density. *Dotted line:* Gaussian density. **Right:** Plots of (optimal) shrinkage functions for each distribution above. *Solid line*: shrinkage corresponding to Laplace density. *Dashed line:* shrinkage corresponding to moderately sparse(super-gaussian) density. *Dash-dotted line:* shrinkage corresponding to strongly sparse(super-gaussian) density. *Dotted line:* $y = x$ for reference. Figure and text from [55].

and

$$\widehat{s} = sign(y)max(0, |y| - \sigma^2|f'(y)|) \tag{4.19}$$

where $f(y) = -log(p(y))$, the negative log density of the distribution of $y$ and $f'(y)$ is the corresponding derivative. The shrinkage functions are designed to reduce the entropy of a distribution (i.e., making it even sparser). The shrinkage functions work in exactly the same way as regularization does within neural networks by driving the magnitude of components/weights towards zero. Notice the similarity of the shrinkage functions to the regularization terms of Equations 3.18 and 3.21. In [55], the researchers show that the more kurtotic the distribution of a random variable is, the easier it is to denoise, since small component values almost surely denote noise and can be set to zero. Thus, by using ICA, sparse, super-gaussian components can be identified that in turn are easily de-noised and converted back into the image patches. In order to denoise images rather than patches two potential approaches exist. One approach is to divide the image into non-overlapping tiles and denoise each tile independent of others. Unfortunately, this introduces significant blocking artifacts where the tiles meet one another. In addition the procedure is not invariant to translation since the output for a given pixel location would depend on its relative position within the tile.

In contrast, the aforementioned **sliding window** approach analyzes each loca-

tion, $(i, j)$, after padding the image appropriately. I.e., given an $N_r(i, j)$ patch, the image is padded by $r$ pixels on all sides. For interior image regions, the net effect is that location $(i, j)$ is actually analyzed $(2r + 1)^2$ times[5]. The final output is then produced by taking an average of all outputs for a given location $(i, j)$. Unlike tiling, this technique is translation invariant, and eliminates blocking artifacts. There are however border effects, but this is a general issue in image processing. As previously described convolutional decoding can be used to implement the sliding window algorithm in an efficient manner.

ICA denoising methods are closely related to wavelet shrinkage and wavelet coring [115]. More recently researchers extended these basic models to include a Markov random field [109], which specifies a prior on the spatial distribution of components $\boldsymbol{S}$ which is discussed later in the chapter.

### 4.4.4 ICA for Feature Extraction from Color and Stereo Images

In [51], researchers applied the FastICA algorithm [58] to color and stereo image data. By concatenating the color/stereo channels into a single vector and subsequently running ICA on the new joint vectors, the approach was able to learn color and stereo filters. More specifically, for two stereo image patches, vectorized into $\boldsymbol{x}_1, \boldsymbol{x}_2$, and in correspondence with one another, the joint vector, $\boldsymbol{z}$ is produced by:

$$\boldsymbol{z} = \begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{pmatrix} \tag{4.20}$$

Then finding the latent factors $\boldsymbol{A}$, by solving

$$\begin{pmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{pmatrix} = \boldsymbol{z} = \boldsymbol{A}\boldsymbol{s} = \begin{pmatrix} \boldsymbol{A}_1 \\ \boldsymbol{A}_2 \end{pmatrix} \boldsymbol{s} \tag{4.21}$$

results in a joint representation of the stereo channels. The matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ can be once again visualized by reshaping each row into an $n \times n$ filter. The result of applying ICA to stereo images is depicted in Figure 4.4.

### 4.4.5 ICA for Regression

Hyvrinen et al. [56] proposed independent component regression (ICR), which elegantly demonstrates the intricate relationship between ICA, artificial neural networks [46] and projection pursuit regression [52]. More specifically, the goal is to identify a **joint latent space** that can succinctly model both input and output in a manner similar to modeling stereo image patches. To accomplish this feat, first

[5]The borders, e.g., location (1,1) would only be analyzed $(r + 1)^2$ times, or if padding is not used, just one output would be produced.

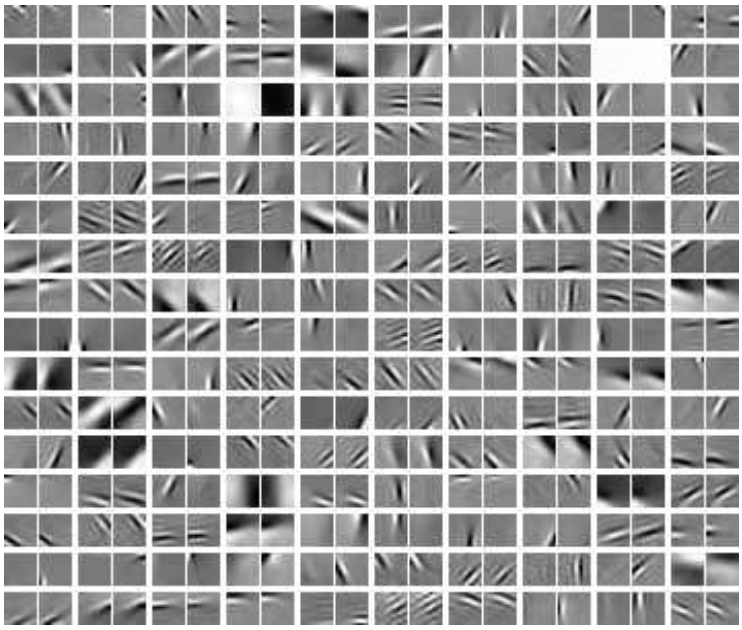


Figure 4.4: "ICA basis of stereo images. Each pair of patches represents one basis vector $a_i$ of the estimated mixing matrix $\boldsymbol{A}$. Note the similarity of these features to those obtained from standard image data. In addition, these exhibit various degrees of binocularity and varying relative positions and phases." Figure and Text from [51].

consider a set of vectorized image patches $\boldsymbol{X}$ as defined in equation 3.8 and the corresponding vectorized patches of labels $\boldsymbol{Y}$ defined in a similar manner. Then, let $\boldsymbol{Z}$ be the concatenated input-output matrix given by:

$$\boldsymbol{Z} = \begin{pmatrix} \boldsymbol{X} \\ \boldsymbol{Y} \end{pmatrix} \tag{4.22}$$

Applying ICA (after removing linear dependencies between $\boldsymbol{X}$ and $\boldsymbol{Y}$) to $\boldsymbol{Z}$ identifies the *joint* latent space. Throughout the text we refer to this and related techniques as associative component analysis. One related and well known technique is partial least squares (PLS) (e.g., [84]) which extracts and associates decorrelated input/output components. As such PLS performs associative PCA, simultaneously extracting principal components from input and output and performing linear regression between the two. In contrast, ICR is a non-linear procedure. The following formulation follows [56]. By performing ICA on $\boldsymbol{z}$, formed by concatenating input $\boldsymbol{x}$ and output $\boldsymbol{y}$ vectors, a joint latent space is found given by:

$$\begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A}_{in} \\ \boldsymbol{A}_{out} \end{pmatrix} \boldsymbol{s} \tag{4.23}$$

where, once again, $\boldsymbol{s}$ is a vector of joint independent components. The output, $\boldsymbol{y}$ can now be approximated as:

$$\tilde{\boldsymbol{y}} = E\{\boldsymbol{y}|\boldsymbol{x}\} = \boldsymbol{A}_{out} \int_{\boldsymbol{s}=\boldsymbol{A}_{in}^T\boldsymbol{x}} \boldsymbol{s}p(\boldsymbol{s}) \approx \boldsymbol{A}_{out}\boldsymbol{g}(\boldsymbol{A}_{in}^T\boldsymbol{x}) \tag{4.24}$$

To understand equation 4.24, first observe that $\boldsymbol{A}_{in}^T\boldsymbol{x} = \tilde{\boldsymbol{s}}$, is a (noisy) approximation to the joint latent factors $\boldsymbol{s}$. The function $\boldsymbol{g}$ can then be viewed as denoising $\tilde{\boldsymbol{s}}$. The denoised components $\boldsymbol{g}(\tilde{\boldsymbol{s}})$ are then transformed via $\boldsymbol{A}_{out}$ into the final output $\tilde{\boldsymbol{y}}$. A simpler version of this idea was successfully applied to image denoising and presented in a preceding section of this chapter. In addition, [55] shows that the more super-gaussian the distribution $p(s_i)$ is, the better the denoising results are (provided that the noise is distributed as a zero mean Gaussian with known variance). This is not surprising, since the essence of the algorithm is to shrink each $s_i$ towards zero. The more leptokurtic a distribution, $p(s_i)$ is, the less entropy is present and hence small (in absolute value) non-zero components are more likely to be noise. As a result, sparse code shrinkage via soft thresholding works better for highly kurtotic distributions. The method is closely related to Bayesian Wavelet Coring [115].

Turning our attention back to the soft thresholding function $\boldsymbol{g}(\boldsymbol{u}) = [g_1(u_1), ..., g_n(u_n)]$ from equation 4.24, [53] shows that letting each $g_k(u_k) = -tanh(u_k) + cu_k$, where $c$ is a global constant common to all $k$, corresponds to a model based neural network (with a single hidden layer). In this case each hidden node of the neural net is performing feature extraction via $\boldsymbol{W}_1 = \boldsymbol{A}_{in}$, denoising via $\boldsymbol{g}_1 = -tanh(u_k) + cu_k$, and reconstruction using $\boldsymbol{W}_2 = \boldsymbol{A}_{out}$ . In essence, each component $u_k$ is shrunk by $-tanh(u_k)$ in order to compensate for noise during the feature extraction process resulting from the missing data, $\boldsymbol{y}$.

## 4.5 Autoencoders and Non-Linear PCA

An auto-encoder is an artificial neural network used for learning efficient encodings. As the name implies, the aim of an auto-encoder is to learn an encoding or a representation of the data. Auto-encoders, depicted in Figure 4.5, use one or more hidden layers and have the input layer identical to the output layer. Formally an autoencoder, with $l$ layers is given by the function:

$$\hat{\boldsymbol{x}} = (\boldsymbol{g}_l \circ \boldsymbol{W}_l \circ ... \circ \boldsymbol{g}_1 \circ \boldsymbol{W}_1)\, \boldsymbol{x} \tag{4.25}$$

where $\hat{\boldsymbol{x}}$ is the reconstruction of input $\boldsymbol{x}$. As mentioned, the goal is to learn a representation of the data matrix $\boldsymbol{X}$. More specifically, an autoencoder can be thought of as a feature extraction method. The output of any hidden layer, $\boldsymbol{f}^k(\boldsymbol{x}) =$

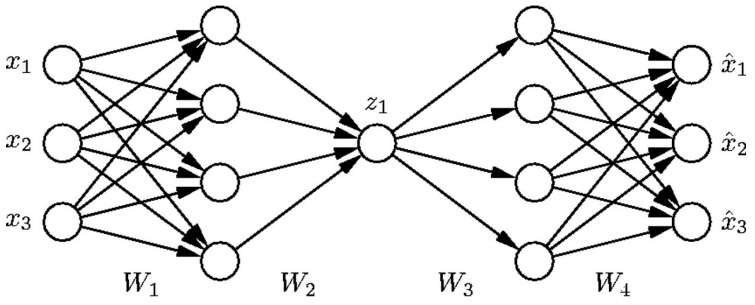



Figure 4.5: Nonlinear PCA. A typical auto-associative neural network. The network's output is forced to be equal to the input $\boldsymbol{x}$. Illustrated is a $[3-4-1-4-3]$ network architecture with biases omitted for clarity. Three-dimensional samples $\boldsymbol{x}$ are compressed into a scalar component $z$ by the extraction part. The reconstruction part synthesizes $\hat{\boldsymbol{x}}$ from $z$. The samples are usually noise-filtered representations of $\boldsymbol{x}$. Text and Figure modified from [112].

$(\boldsymbol{g}_k \circ \boldsymbol{W}_k \circ ... \circ \boldsymbol{g}_1 \circ \boldsymbol{W}_1)\,\boldsymbol{x}$, $1 \leq k < l$, can be thought of as features representing $\boldsymbol{x}$, while the remaining layers $k+1, ..., l$ can be viewed as the reconstruction layers that synthesize an approximation of $\boldsymbol{x}$ from the features, $\boldsymbol{f}^k(\boldsymbol{x})$.

If the transfer functions, $\boldsymbol{g_i}$, are linear then the auto-encoder has the representational power of PCA, regardless of depth. With a single non-linear layer of hidden units, and some simple additional constraints, it is possible to produce an ICA like representation of the data. In this case the feature extraction is performed by $\boldsymbol{W}_1 = \boldsymbol{A}^\dagger$, while the reconstruction is performed by $\boldsymbol{W}_2 = \boldsymbol{A}$.

In contrast to simple shallow architectures, deep non-linear architectures attempt to learn hierarchical structure, and hold the promise of being able to first learn simple concepts, and then successfully build up more complex concepts by composing together the simpler ones. However, it is extremely difficult to optimize the weights in non-linear autoencoders that have multiple hidden layers and millions of parameters. With large initial weights, autoencoders typically find poor local minima and with small initial weights, the gradients in the early layers are tiny, making it infeasible to train autoencoders with many hidden layers. This inability to learn is the main reason this potentially powerful nonlinear dimensionality reduction algorithm has not found many practical applications. However, if the initial weights are close to a good solution, gradient descent works well, but finding such initial weights requires a very different type of algorithm that learns one layer of features at a time.

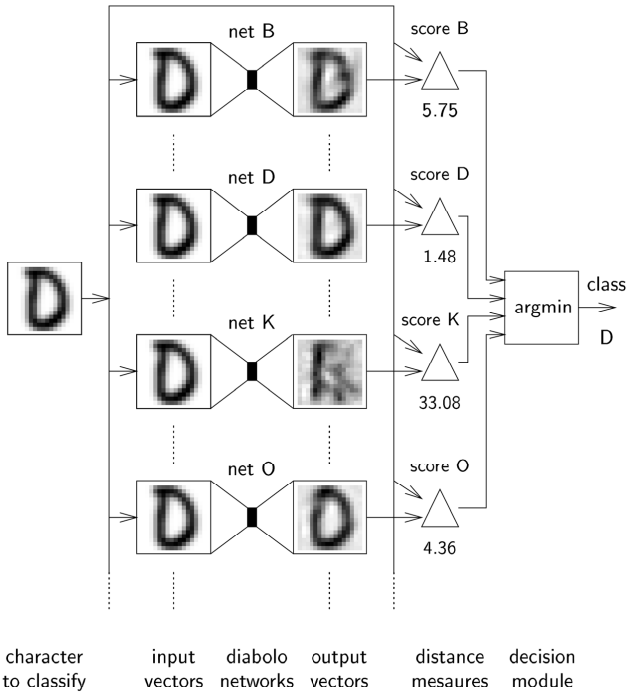



Figure 4.6: Architecture of the Diabolo Networks. Each autoencoder is trained to minimize the reconstruction error of one specific class of samples. At runtime the reconstruction error of each autoencoder is used to decide the class label for a given sample. Figure from [113].

### 4.5.1 Diabolo Networks - Discriminative Autoencoders

In [113], a set of autoencoders was used in a discriminative manner. Each autoencoder is trained to minimize reconstruction error of samples belonging to one of $k$ classes. During the on-line phase, the algorithm applies each autoencoder and measures the reconstruction error (MSE) each autoencoder produces. The label corresponding to the autoencoder with the lowest error is then assigned to a given test sample. Figure 4.6 depicts the diabolo classifier used for character recognition.

To further improve performance, each autoencoder can utilize information from other classes. Recall the gradient descent procedure defined by Equation 4.5.1. In discriminative mode, the autoencoder employs both gradient *descent*, to minimize MSE for the target class samples, and gradient *ascent* to increase the reconstruction error of the non-target class samples. Thus the weight update rule of the $k^{th}$

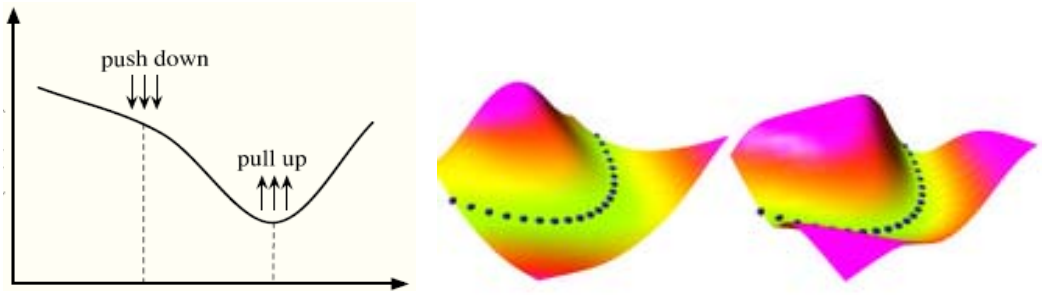



Figure 4.7: Energy Based Learning. **Top:** The push/pull effect on the energy surface. **Bottom:** The evolution of the energy surface with the black dots representing the training data. Figure modified from [70].

autoencoder becomes:

$$\boldsymbol{W}_{new}(i,j) = \begin{cases} \boldsymbol{W}(i,j) - \eta_1 \frac{\partial E_{mse}(\boldsymbol{y}, \boldsymbol{h}_{\boldsymbol{\omega}}(\boldsymbol{x}))}{\partial \boldsymbol{W}_{(i,j)}} & \text{if } y = k \\ \\ \boldsymbol{W}(i,j) + \eta_2 \frac{\partial E_{mse}(\boldsymbol{y}, \boldsymbol{h}_{\boldsymbol{\omega}}(\boldsymbol{x}))}{\partial \boldsymbol{W}_{(i,j)}} & \text{if } y \neq k \end{cases} \tag{4.26}$$

The concept of 'tugging' up or down on an energy function is also related to the wake-sleep and contrastive divergence algorithms [49] used to train a Restricted Boltzmann Machine presented in the next section. The algorithms maximize probability of training points, while minimizing the likelihood of samples confabulated during the 'dream' phase. The idea was generalized by energy based models (EBM) of [70]. In contrast to probabilistic graphical models[6], where a probability is associated with each configuration, energy based models simply assign an un-normalized energy and eliminate the need for computing the partition function. In the generative case, the goal of (EBM) is to minimize the energy of training data and nearby points, while raising the energy of points far from the training data as depicted in Figure 4.7. Similar to the random field models, in the discriminative case, one must search for the configuration with the lowest energy by querying the generative model or following the energy gradient. In [89] an energy based model [71] was used in a manner analogous to heterogeneous stacking and was also shown to be effective at 'cleaning up' label noise.

### 4.5.2 Deep Greedy Layer-wise Learning

As previously mentioned, for networks with many (more than four) hidden layers, conventional learning procedures fail. However, the problem can be remedied

[6]Technically probabilistic graphical models are a subclass of energy based models.

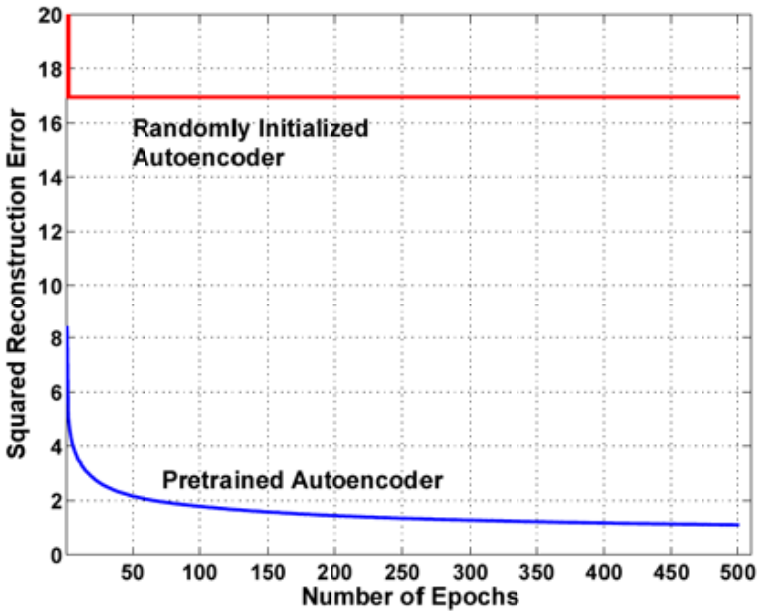



Figure 4.8: Learning Deep Autoencoders. The deep 784-400-200-100-50-25-6 autoencoder makes rapid progress after pretraining but no progress without pretraining. From [48].

by using initial weights that approximate the final solution. The process to find these initial weights is often called pretraining. The effectiveness of pretraining is demonstrated in Figure 4.8.

The pretraining technique, developed by Hinton et al. [48], for training many-layered "deep" auto-encoders involves treating each neighboring set of two layers as a Restricted Boltzmann Machine for pre-training to approximate a good solution and then using a backpropagation technique to fine-tune the whole network. Hinton et al. [48] describe a way to perform fast, greedy learning of deep, directed belief networks by training a stack of Restricted Boltzmann Machines one layer at a time. In addition, Bengio et al. [7], proposed a similarly greedy algorithm, but directly based on a single hidden layer autoencoders, called stacked autoencoders. Ranzato et al. [103] developed an energy-based hierarchical algorithm, based on a sequence of sparsified autoencoders/decoders. In related direction, several studies have compared models such as these, as well as non-hierarchical/non-deep learning algorithms, to the response properties of neurons in area V1. In [120] van Hateren and van der Schaaf demonstrated that the filters learned by independent components analysis (ICA) on natural image data match very well with the classical receptive fields of V1 simple cells. Filters learned by sparse coding also give responses similar to V1 simple cells.

The aforementioned greedy learning algorithms for pretraining autoencoders,

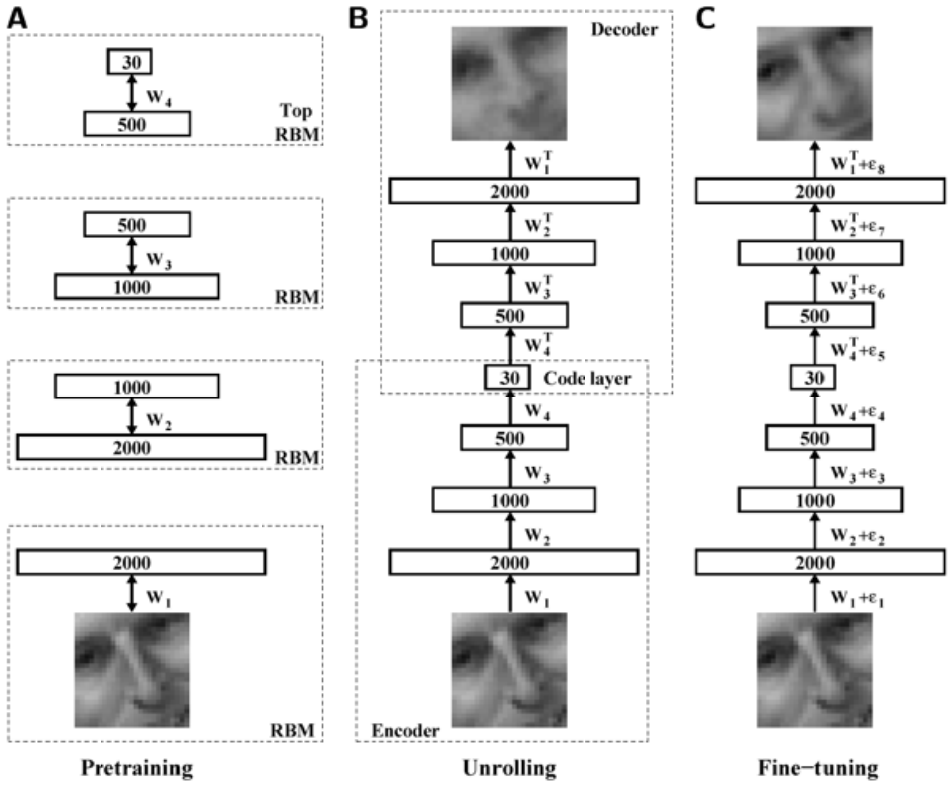



Figure 4.9: Learning Deep Autoencoders using greedy layer-wise training and fine-tuning. From [48].

progressively discover low-dimensional representations. As an interesting side-note, observe the similarities between pre-training, cascade correlation and stacked generalization. Both use previous output as input to a new layer of machine learning algorithms. After the pretraining stage, the model is unfolded as shown in the middle column of Figure 4.9, to produce encoder and decoder networks that initially use the same weights as learned during pretraining. Subsequently, a global fine-tuning stage uses backpropagation through the whole autoencoder to fine-tune the weights for optimal input reconstruction. The key idea is that these greedy learning algorithms will perform a global search for a good, sensible region in the parameter space. Therefore, with just pretraining, a good data reconstruction model can be obtained. Backpropagation is better at local fine-tuning of the model parameters than global search. So further training of the entire autoencoder using backpropagation can result in a good local optimum.

To formalize the approach a succinct description for each of the three phases, pre-training, unrolling, and fine-tuning, using the notation defined thus far, is as

follows:

**Pre-training** starts by training an autoencoder with a single hidden layer, either using contrastive divergence as in the case of RBMs [48] or back propagation as in the case of stacked autoencoders [6]. Let $\boldsymbol{f}^k(\boldsymbol{x})$ represent the hidden layer output of the $k^{th}$ autoencoder with respect to original input $\boldsymbol{x}$. That is $\boldsymbol{f}^k(\boldsymbol{x}) = \left(\boldsymbol{g} \circ \boldsymbol{W}_1^k\right) \boldsymbol{f}^{k-1}(\boldsymbol{x})$, with $\boldsymbol{f}^0(\boldsymbol{x}) = \boldsymbol{x}$, and the $k^{th}$ autoencoder given by:

$$\widehat{\boldsymbol{f}^{k-1}(\boldsymbol{x})} = \left(\boldsymbol{W}_2^k \circ \boldsymbol{g} \circ \boldsymbol{W}_1^k\right) \boldsymbol{f}^{k-1}(\boldsymbol{x}) \tag{4.27}$$

subject to the constraint that $\boldsymbol{W}_1^k = \boldsymbol{W}_2^{k^T}$. This constraint has already been discussed in previous section describing PCA and serves to produce a single encoding/decoding matrix. Furthermore, in contrast to the autoencoder for PCA this type of network has a non-linear transfer function $\boldsymbol{g}$.

**Unrolling** is performed once all individual layers have been learned. The deep autoencoder is assembled from the constituent parts, namely the previously learned single layer autoencoders:

$$\hat{\boldsymbol{x}} = (\boldsymbol{W}_2^1 \circ \boldsymbol{g} \circ \boldsymbol{W}_2^2 \circ \boldsymbol{g} \circ ... \circ \boldsymbol{g} \circ \boldsymbol{W}_2^k \circ \boldsymbol{g} \circ \boldsymbol{W}_1^k \circ ... \circ \boldsymbol{g} \circ \boldsymbol{W}_1^2 \circ \boldsymbol{g} \circ \boldsymbol{W}_1^1)\boldsymbol{x} \tag{4.28}$$

**Fine-Tuning** Once the deep net has been constructed, backpropagation as in Equations 4.5.1, 3.18 or 3.21 can be used to fine tune all the weights in all the layers. The constraint $\boldsymbol{W}_1^k = \boldsymbol{W}_2^{k^T}$ is now relaxed (i.e., ignored) to improve reconstruction.

As a final thought we quote Yoshua Bengio [6] on this utmost critical topic:

> Deep architectures and the greedy layer-wise strategy exploits the principle, apparently also exploited by humans (Piaget, 1952), that one can more easily learn high-level abstractions if these are defined by the composition of lower-level abstractions, with the property that these lower-level abstractions are useful by themselves to describe the data, and can thus be learned before the higher level abstractions are learned.

## 4.6 Convolutional Networks

Convolutional neural networks (CNN) are deep multi-layer artificial neural networks with a brain-inspired architecture motivated by vision tasks. Unlike conventional neural networks, they have local connectivity. In every hidden layer of

conventional neural nets, each input is connected to each node of the subsequent layer, i.e., they are fully connected. In contrast, local connectivity constrains the input field of each hidden layer neuron to a small window of neurons at the previous layer. CNN's are further constrained by the concept of parameter sharing, whereby neurons with exactly the same input weights but a different spatial input fields are spatially replicated across a $C_{map}$ that preserves the topology of the input image, see Figure 4.10. The same concept can be applied to one-dimensional data (sequences), yielding so-called time-delay neural networks, as well as to higher-dimensional structures. What is important about convolutional neural networks is that they can be trained to be extremely robust to a large number of invariance factors in object recognition, such as of translation, rotation, slant, scaling, edge thickness, illumination, clutter (and to some degree occlusion). Since these factors can be composed and present a combinatorial explosion of variations, it appears hopeless to attack such tasks with template-based or local kernel algorithms. On the other hand, convolutional neural networks have been found to be extremely successful in object recognition and object detection tasks in vision.

### 4.6.1 Neocognitron

One of the first hierarchical multilayered neural networks, capable of learning, was the neocognitron, proposed by K. Fukushima [36]. It has been used for handwritten character recognition and other pattern recognition tasks. Figure 4.10 shows a typical architecture of the neocognitron network. The lowest stage is the input layer consisting of a two-dimensional array of cells, which represent the "photoreceptors" of the retina. Connections between cells of adjoining layers are retinotopically[7] organized. Each cell receives input connections that lead from cells situated in a spatially localized area on the preceding layer. The rest of the hierarchy is composed of alternating layers of "S-cells" and "C-cells". (The network shown in Figure 4.10, contains an additional contrast-extracting layer, which is usually manually coded). S-cells work as feature-extractors. Functionally, they resemble simple cells of the primary visual cortex and their input connections are modified

[7]Retinotopy describes the spatial organization of the neuronal responses to visual stimuli. In many locations within the brain, adjacent neurons have receptive fields that include slightly different, but overlapping portions of the visual field. The position of the center of these receptive fields forms an orderly sampling mosaic that covers a portion of the visual field. Because of this orderly arrangement, which emerges from the spatial specificity of connections between neurons in different parts of the visual system, cells in each structure can be seen as forming a map of the visual field (also called a retinotopic map, or a visuotopic map). Retinotopic maps are a particular case of topographic organization. Many brain structures that are responsive to visual input, including much of the visual cortex and visual nuclei of the brainstem (such as the superior colliculus) and thalamus (such as the lateral geniculate nucleus and the pulvinar), are organized into retinotopic maps, also called visual field maps. From `http://en.wikipedia.org/wiki/Retinotopy`

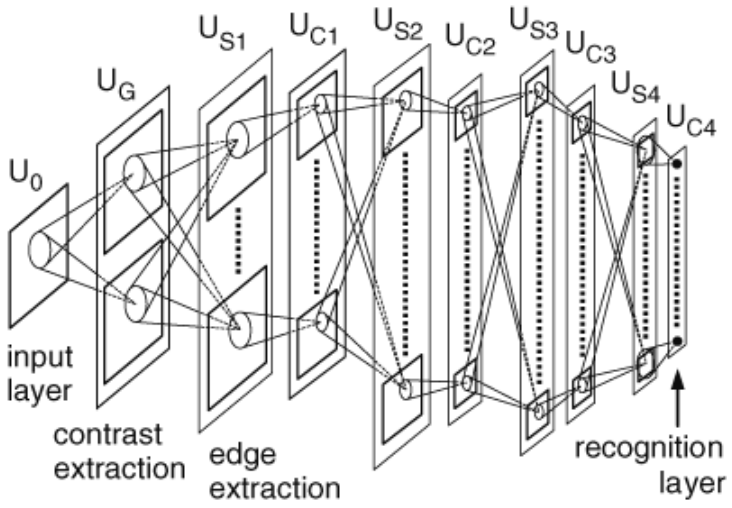



Figure 4.10: Architecture of a Neocognitron From `http://www.scholarpedia.org/article/Neocognitron`.

through learning. After learning, each S-cell selectively responds to a specific feature present within its receptive field. Generally speaking, local features, such as edges or lines in particular orientations, are extracted in lower stages resembling the learned filters of ICA. More global features, such as parts of learning patterns, are extracted in higher stages.

C-cells, which resembles complex cells in the visual cortex, are inserted in the network to allow for positional errors in the features of the stimulus. The input connections of C-cells, which come from S-cells of the preceding layer, are fixed and invariable. Each C-cell receives excitatory input connections from a group of S-cells that extract the same feature, but from spatially different positions. The C-cell responds if at least one of the S-cells within its receptive field is active. Thus, the C-cell's response is locally invariant to spatial translation of the input pattern. From a technical perspective, the C-cells execute a blurring (low pass filtering) operation, because the responses of preceding S-cells are summed by C-cells. Since the receptive fields of C-cells overlap each C-cell in essence averages out the outputs of spatially localized S-cells.

Each layer of S-cells or C-cells is divided into sub-layers, called "cell-planes", according to the features to which the cells responds. As usual the cells in each cell-plane are arranged in a two-dimensional lattice such as the one defined in Equation 2.1. A cell-plane is a group of cells that are arranged retinotopically and share the same set of input connections. In other words, the connections to a cell-plane have a translational symmetry. As a result, all the cells in a cell-plane have receptive fields of an identical characteristic, but the locations of the receptive fields differ from

cell to cell. The modification of S-cell connections during the learning process is also constrained by the sharing of connections. The final layer of the network, of C-type cells acts as a linear classifier that outputs the class of the input image (e.g., the image is the digit '1').

The original neocognitron used unsupervised learning to modify the weights of S-cells and was able to extract 'interesting' features from the data. Unfortunately it did not produce good performance (on the character recognition task).

### 4.6.2 LeNet

In contrast to the neocognitron, LeNet, also a Convolutional Neural Network architecture proposed by Yann LeCun [69], utilized supervised back-propagation to learn the weights for S-cells. In addition, these networks, depicted in Figure 4.11, have different C-cells. The C-cells in the CNN downsample the input maps rather than just filter them. This results in smaller and smaller feature maps further up the hierarchy. In addition, conventional layers have been added at the very top of the hierarchy in order to further improve object recognition performance.

Unfortunately, the major difference between neocognitron and the CNN architecture maybe notational rather than conceptual. The description of the neocognitron [36] uses the concept of C-cells and S-cells to denote Simple and Complex cells, where by the simple cells learn while the complex cells perform a fixed spatial averaging operation. On the other hand, the description of the CNN uses C-maps and S-maps to mean Convolutional and Sub-sampling layers, thereby reversing the notational logic. Despite the confusing notation, after training both systems function in a similar manner.

In both cases, due to the weight sharing constraints, the systems produce a deep neural network that utilizes a set of learned weights very similar to those learned by PCA/ICA and autoencoders. As the name implies, these weights can be reshaped into filter banks and convolved with input images in a manner analogous to methods presented in Section 4.4.1. In essence, rather than working on single vectorized image patches, CNNs work on images. Despite the apparent complexity, convolutional networks can be described using the neural network notation developed thus far. Consider a set of learned convolutional weights, $\omega = \{\boldsymbol{W}_1, \boldsymbol{W}_2, ...\}$. The first set of weights, $\boldsymbol{W}_1$, can be reshaped much like the ICA filters into 2-D filters, $\Phi_1 = \{\grave{\boldsymbol{w}}_1, \grave{\boldsymbol{w}}_2, ...\}$. Each convolution produces a set of feature maps, $\boldsymbol{f}_\alpha = \boldsymbol{I} * \grave{\boldsymbol{w}}_\alpha$. The individual feature maps are subsequently transformed by the transfer function $\boldsymbol{g}$, sub-sampled, and composed into a multidimensional feature map (i.e., a tensor), $\boldsymbol{f} = [\boldsymbol{f}_1, \boldsymbol{f}_2, ..., \boldsymbol{f}_\alpha, ...]$. The second set of weights, $\boldsymbol{W}_2$, is now reshaped into a set of multidimensional filters $\Phi_2$, which are convolved with the feature map $\boldsymbol{f}$. The process repeats itself until all the convolutional and sub-sampling operators have been applied. The remaining feature map, $f_{last}$ is reshaped into a vector, $\boldsymbol{x}$ which

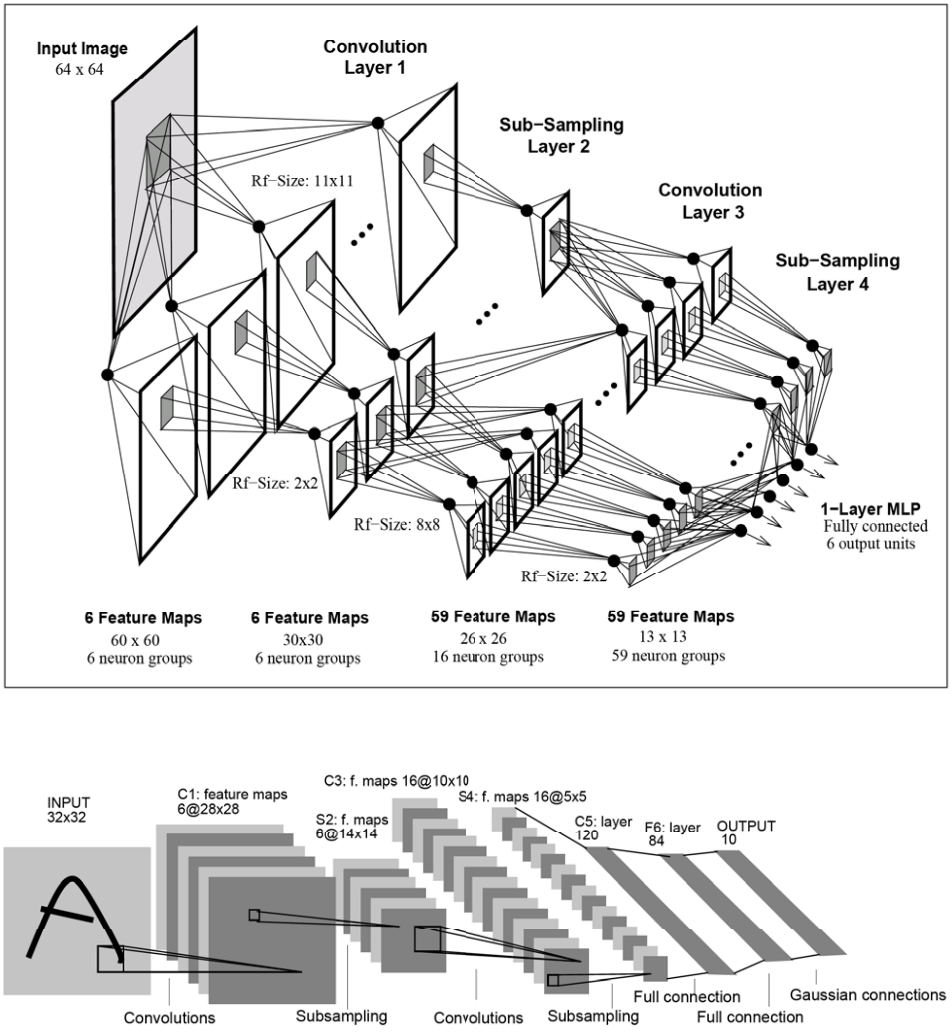



Figure 4.11: Architecture of Convolutional Neural Networks. **Top:** From [32]. **Bottom:** From [69]. The key difference between the top and bottom architectures is that the top architecture performs a multidimensional convolution in layer 3. On the other hand, the bottom network, the original LeNet5, performs standard convolution in layers 3.

is fed into a conventional neural net to produce the final output $\boldsymbol{y}$. In this fashion an image, for example containing a digit, can be classified. In terms of image/object segmentation, CNN's have been applied with promising results in [60] and [89].

## 4.7 High Order Random Fields for image analysis and denoising

As mentioned in the previous chapter, a major limitation of random field methods is the practical restriction to pairwise clique potential within the interaction term of Equations 3.32 and 3.33. The limitation stems from the combinatorial explosion of potential clique configurations possible as the size of the cliques gets larger. To overcome this problem a number of research endeavors have tried to create higher order random field methods using a limited but useful subset of clique configurations. Early attempts made use of hand-selected filter banks in order to efficiently describe an $n \times n$ patch thereby circumventing the need to integrate over all potential clique configurations. A brief review of these methods and further references can be found in [108]. Within the random field framework, the selection of relevant cliques is analogous to manual feature selection. More recently, research on automated feature extraction, such as PCA, ICA and a myriad a variants has been integrated into the random field framework. The combined methodology, enables simultaneous parameter learning and selection of relevant filters constituting a subset of cliques. One of the first such systems, called Field of Experts [109], is based on the product of experts (PoE) methodology of [50, 49].

Like ICA, the PoE methodology models image patches as a set of latent features. Prior to describing the PoE model, recall that by assumption the ICA components are independent. Therefore, the likelihood of observing a vectorized image patch $\boldsymbol{x}$ is given by product of the empirical distributions:

$$p[\boldsymbol{x}] \propto \prod_k g_k(s_k) = \prod_k g_k(\boldsymbol{A}_k^\dagger x)$$

where the $k^{th}$ independent component $s_k = \boldsymbol{A}_k^\dagger \boldsymbol{x}$, is the dot-product of the input $\boldsymbol{x}$ and the $k^{th}$ row of the 'un-mixing' matrix $\boldsymbol{A}^\dagger$, and $g_k$ is the empirical likelihood of observing the value $s_k$ for the $k^{th}$ independent component. From a practical point of view, given an $n \times n$ image patch, ICA in general cannot find a set of $n^2$ fully independent components, thereby making the above model an approximation.

In contrast to ICA, the PoE approach models the likelihood function, $g_k$, in addition to the filters $\boldsymbol{J}_k^T \approx \boldsymbol{A}_k^\dagger$. The generative product of student-t experts (PoT), from [117] is given by:

$$p_\omega[\boldsymbol{x}] = \frac{1}{Z_\omega} \prod g_k(\boldsymbol{J}_k^T \boldsymbol{x}, \alpha_k) \tag{4.29}$$

where the learned parameters are $\boldsymbol{\omega} = \{\boldsymbol{J}, \boldsymbol{\alpha}\}$, $Z_\omega$ is the partition function, and $g_k$ is modeled as a student-t expert:

$$g_k(\boldsymbol{J}_k^T \boldsymbol{x}, \alpha_k) = \left(1 + (\frac{\boldsymbol{J}_k^T \boldsymbol{x}}{2})^2\right)^{\alpha_k}, \ \alpha_k \geq 0 \tag{4.30}$$

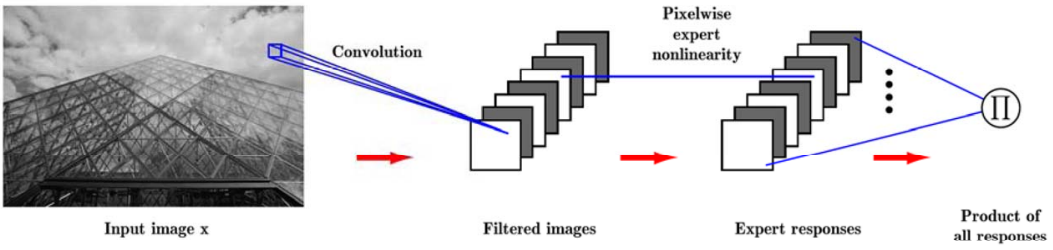



Figure 4.12: Fields of Experts architecture visualized as a specialized Convolutional Neural Network. From [108]

Observe, each expert functions in a manner analogous to a single hidden node of a neural network, whereby the linear filter response is transformed by a non-linearity, $g_k$. A major advantage of the PoE model over ICA is its ability to create over-complete representations, where the number of components exceeds the dimensionality of $\boldsymbol{x}$. The over-complete case enables the PoE to model filter dependencies and is therefore regarded as a more expressive model. Observe that an autoencoder, in particular a deep autoencoder can create over and under complete representations as well, and can also model the aforementioned filter dependencies.

One of the limitations of both ICA and PoE approaches is that they model image patches rather than whole images. As a result, the interactions between label patches are ignored, i.e., these algorithms view image patches as independent. To remedy the situation, the Field of Experts model [109] proposed to learn an MRF model over the PoT model via:

$$p_\omega[\boldsymbol{I}] = \frac{1}{Z_\omega} \prod_l \prod_k g_k(\boldsymbol{J}_k^T \boldsymbol{x}_l, \alpha_k) \tag{4.31}$$

where $l$ indexes over the image patches $\boldsymbol{x}_l$ extracted from image $\boldsymbol{I}$. This model accounts for interactions of overlapping patches and is (arguably) better suited for modeling images. Equation 4.31 actually implements a special type of convolutional neural network, depicted in Figure 4.12. The first layer of this architecture convolves the input image with filters created from columns of $\boldsymbol{J}$. The next layer applies a pixel-wise non-linearity $g_k$. The last layer multiplies all outputs of all experts for all spatial locations to produce a single scalar that, after being normalized by the partition function $Z$ represents the likelihood of observing a given image. To learn the parameters of Equation 4.31, a variant of contrastive divergence is used in a manner similar to training the PoE model [117]. Once the model for image prior $p[\boldsymbol{x}]$ is learned, it can be applied to a number of low level image processing tasks, such as denoising and in-painting. For denoising a gradient ascent procedure is used. Given an observed, noisy image $\boldsymbol{I}^{t_0}$, the goal is to recover the true image

$\boldsymbol{I}^\star$ via an iterative process given by:

$$\boldsymbol{I}^{t+1} = \boldsymbol{I}^t + \eta \left[ \beta \sum_{k=1}^{K} \boldsymbol{J}_k^- * g'(\boldsymbol{J}_k * \boldsymbol{I}^t, \alpha_k) + \frac{1}{\sigma_2}(\boldsymbol{I}^{t_0} - \boldsymbol{I}^t) \right] \tag{4.32}$$

where $\sigma$ is the variance of the corrupting noise, $\eta$ controlls the gradient step size and $\beta$ controls relative importance of the gradient of the FoE prior with respect to the observed image. Observe that the partition function is not involved in the denoising process, making the computational demand relatively light. One drawback of this model is the need to specify the noise variance, which may not always be known. Notice that the gradient of the FoE prior resembles the ICA based denoising procedure using convolutional encoding/decoding. Therefore the main difference between patch based and image based models is the type of filters they learn (at least in the context of image denoising).

# Chapter 5

# From Pixels To Patches To Structures

This chapter presents the third and fourth major contributions of this work, namely the Output Decomposition Mixture Of Experts and the Stacked Convolutional Regression algorithms. The latter approach is fully automated requiring minimal human intervention and is therefore well suited for use within the Data Driven Region Growing Framework presented in Chapter 2.

## 5.1 Output Decomposition

In this section we propose to first decompose the ground truth into output features. Subsequently, input features are mapped to output features by employing an ensemble of function approximators. Once output features have been produced they are synthesized into the final image labeling. The output decomposition scheme allows each function approximator to focus on a different aspect of the pixel labeling problem. The goal of the approach is to operate in between low level pixel labels and high level objects. As an example, consider the rock objects depicted in Figure 5.1. The contour of each rock is composed of differently oriented edges. For each oriented edge we can train a specific detector and then synthesize the object contour by combing the outputs of each detector. Hence by decomposing the ground truth into a set of object parts describing a patch of ground truth labels, we can break a difficult pixel labeling problem into a set of easier structure labeling problems.

### 5.1.1 From pixel labeling to structure labeling

As described in Chapter 3, classical pixel labeling attempts to find the following mapping:

$$h_{pl} : \boldsymbol{I}(i,j) \mapsto \boldsymbol{L}(i,j) \tag{5.1}$$

by computing the probability of pixel $(i,j)$ belonging to the target class (as in Equation 3.1 and then thresholding the probability map at level $\tau$ via Equation 3.2).
Equation 3.1 treats individual pixels as i.i.d., an assumption rarely satisfied in practice, since most non-trivial domains exhibit complex pixel interactions. To overcome this problem, contextual pixel labeling defines a feature extraction function that computes local (and possibly global) contextual features, $\boldsymbol{f}^{\boldsymbol{I}}(i,j)$, for each pixel $(i,j)$. Subsequently the newly formed feature vectors are used to learn the mapping:

$$h_{cpl} : \boldsymbol{f}^{\boldsymbol{I}}(i,j) \mapsto \boldsymbol{L}(i,j) \tag{5.2}$$

To further improve pixel classification accuracy, recursive contextual pixel classification [129] and, more recently, random field methods (e.g., Markov / Conditional / Discriminative Random Fields [65, 64]) have been designed to account for label interactions as well as input pixel interactions. These systems, employing Iterated

Conditional Modes for inference, first use the regular contextual pixel labeling, as in Equation 5.2, to produce an initial labeling $\boldsymbol{L}_0$. Subsequently a recursive procedure iteratively computes $\boldsymbol{L}_d$ as follows:

$$h_{rcpl} : [\boldsymbol{f}^{\boldsymbol{I}}(i,j), \boldsymbol{f}^{\boldsymbol{L}_{d-1}}(i,j)] \mapsto \boldsymbol{L}_d(i,j) \tag{5.3}$$

where $\boldsymbol{f}^{\boldsymbol{L}_{d-1}}(i,j)$ is a function extracting features from $\boldsymbol{L}_{d-1}$ at lattice site $(i,j)$. Typically the features extracted from $\boldsymbol{L}$ are very simple, usually just a neighborhood centered about $(i,j)$ (i.e., cliques). In contrast to the approaches defined by mappings in 5.1, 5.2 and 5.3, the following algorithm tackles the problem from a different point of view. The algorithm, named Output Decomposition Mixture of Experts, explicitly extracts contextual features from **both** input images and ground truth images, and subsequently learns a mapping from the former to the latter:

$$h_{\text{OD-MoE}} : \boldsymbol{f}^{\boldsymbol{I}}(i,j) \mapsto \boldsymbol{f}^{\boldsymbol{L}}(i,j) \tag{5.4}$$

as depicted in Figure 5.1. The essence of the algorithm lies in extracting output features, $\boldsymbol{f}^{\boldsymbol{L}}$, that allow the synthesis of output $\boldsymbol{L}$. Once the input/output decomposition scheme has extracted the input and output features, we utilize machine learning algorithms to train a set of models that instantiate the mapping in Equation 5.4. At runtime, the outputs of individual models are fused together to produce the final segmentation $\tilde{\boldsymbol{L}}$.

### 5.1.2 Output Decomposition based Mixture-of-Experts

For simplicity, let us consider a single feature extraction function, extracting $k$ features for each site $(i,j)$, and applicable to both the input image $\boldsymbol{I}$ and the output labels $\boldsymbol{L}$. Using this function we produce a set of input feature maps $\{\boldsymbol{\Phi}_t^{\boldsymbol{I}}\}_{t=1}^k$ and a set of output feature maps $\{\boldsymbol{\Phi}_t^{\boldsymbol{L}}\}_{t=1}^k$. For lattice site $(i,j)$, the feature vectors are therefore given by:

$$\boldsymbol{f}^{\boldsymbol{I}}(i,j) = [\Phi_1^I(i,j), ..., \Phi_k^I(i,j)] \in \mathbb{R}^k$$

$$\boldsymbol{f}^{\boldsymbol{L}}(i,j) = [\Phi_1^L(i,j), ..., \Phi_k^L(i,j)] \in \mathbb{R}^k$$

Using these input/output feature maps we train a set of function approximators $H = \{h_1, ..., h_k\}$ as:

$$h_t : \boldsymbol{f}^{\boldsymbol{I}}(i,j) \mapsto \tilde{\Phi}_t^L(i,j) \tag{5.5}$$

that map input feature vectors, to individual output components. Pictorially, each induced mapping $h_t$ corresponds to an expert depicted by solid arrows in Figure 5.1.

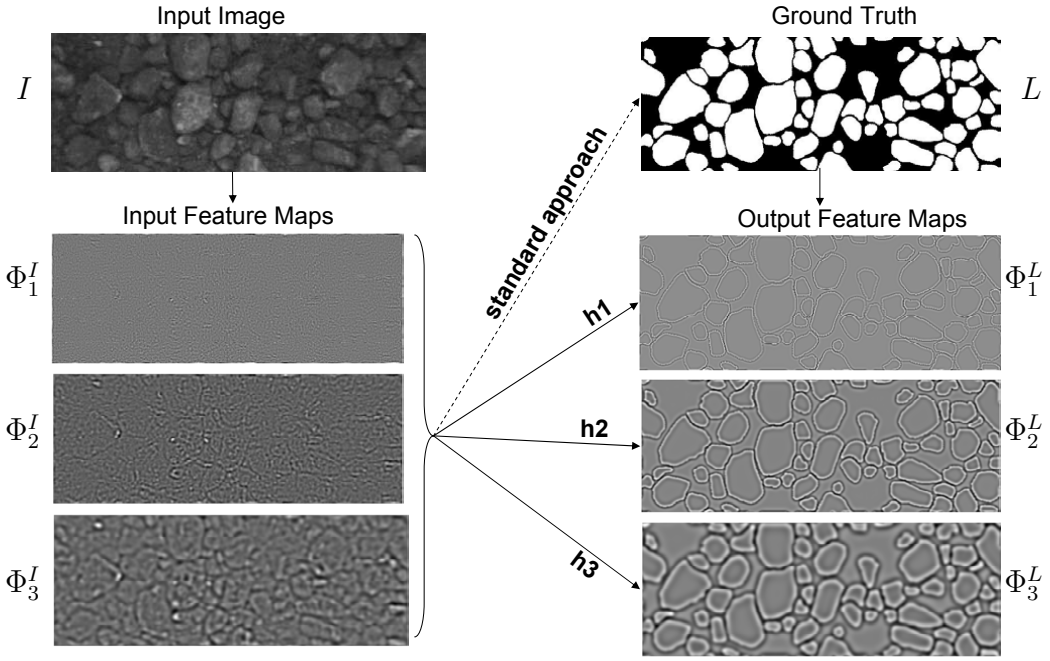



Figure 5.1: Output Decomposition based Mixture of Experts approach. **Left:** Input Image, $\boldsymbol{I}$, and the extracted feature maps, $\Phi_1^I, \Phi_2^I, \Phi_3^I$. **Right:** Corresponding Ground Truth image, $\boldsymbol{L}$, and the extracted feature maps, $\Phi_1^L, \Phi_2^L, \Phi_3^L$. For demonstrational purposes we applied the Difference of Gaussians (DoG) decomposition scheme to both $\boldsymbol{I}$ and $\boldsymbol{L}$. The following mappings are represented: (dashed arrow) from input features to pixel labels as in the case of the **standard approach** defined by Equation 5.2; (solid arrows) from input features to output features as in the case of $h_1, h_2, h_3$ corresponding to the OD-MoE approach defined in Equation 5.5.

### Frequency based Feature Extraction and Output Synthesis

The discrete 2-D Fourier transform [39] of a function $g(i,j)$ and its inverse, each defined on lattice $S$ from Equation 2.1 on p. 10, are given by:

$$G(u,v) = \frac{1}{\sqrt{NM}} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} g(i,j) e^{-(\sqrt{-1})2\pi(\frac{ui}{N}+\frac{vj}{M})}$$

$$g(i,j) = \frac{1}{\sqrt{NM}} \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} G(u,v) e^{(\sqrt{-1})2\pi(\frac{ui}{N}+\frac{vj}{M})}$$

To simplify notation we define:

$$\mathcal{F}[g] = G \; ; \; \mathcal{F}^{-1}[G] = g$$

to denote the Fourier Transform and its inverse. Next, we define a set of frequency filters $\{G_t(u,v)\}_{t=1}^{k}$ with the constraint:

$$\sum_{t=1}^{k} |G_t(u,v)| = 1, \forall (u,v) \in S$$

This constraint enables input reconstruction to be given as simply the summation of filter responses for a given frequency $(u,v)$. The frequency feature coefficients of an arbitrary function $q(i,j)$, defined over lattice $S$, are then simply the point-wise product of filter $G_t$ with the frequency representation of the function, $Q \odot G_t = Q(u,v)G_t(u,v),\ \forall (u,v) \in S$, with $Q = \mathcal{F}[q]$. The spatial feature maps of $q(i,j)$ are therefore defined as:

$$\Phi_t^q = \mathcal{F}^{-1}[Q \odot G_t], t = \{1, ..., k\}$$

Finally, for this specific decomposition scheme, the reconstruction (i.e., **the output synthesis**) function is defined in the Fourier domain as:

$$\tilde{Q} = \sum_{t=1}^{k} \mathcal{F}[\Phi_t^q] \odot G_t^{\boldsymbol{\alpha}} \tag{5.6}$$

where $\boldsymbol{\alpha} = 0$ is the default approach[1]. By setting $\boldsymbol{\alpha} \geq 1$, the algorithm is able to perform online filtering. Experimental results in Section 6.3, will demonstrate the ability of this filtering procedure to attenuate noise resulting from function approximation.

**Runtime OD-MoE**

At runtime, the following steps are performed :

1. Extract image features by convolving the input image with a filter bank $\{G_t\}_{t=1}^{k}$:

   $$\boldsymbol{\Phi}_t^{\boldsymbol{I}} = \mathcal{F}^{-1}\left[\mathcal{F}[\boldsymbol{I}] \odot G_t\right]\ \forall t = \{1, ..., k\}$$

   or equivalently:

   $$\boldsymbol{\Phi}_t^{\boldsymbol{I}} = \boldsymbol{I} * \mathcal{F}^{-1}[G_t]\ \forall t = \{1, ..., k\}$$

   where $*$ is the convolution operator.

2. Perform function approximation by applying the trained ensemble of function approximators defined by Equation 5.5

   $$\tilde{\boldsymbol{\Phi}}_t^{\boldsymbol{L}}(i,j) = h_t(\boldsymbol{f}^{\boldsymbol{I}}(i,j))\ \ \forall (i,j) \in S$$

[1] In the case of $\boldsymbol{\alpha} = 0$, the summation can be performed in the spatial domain as $\tilde{q} = \sum_t \Phi_t^q$

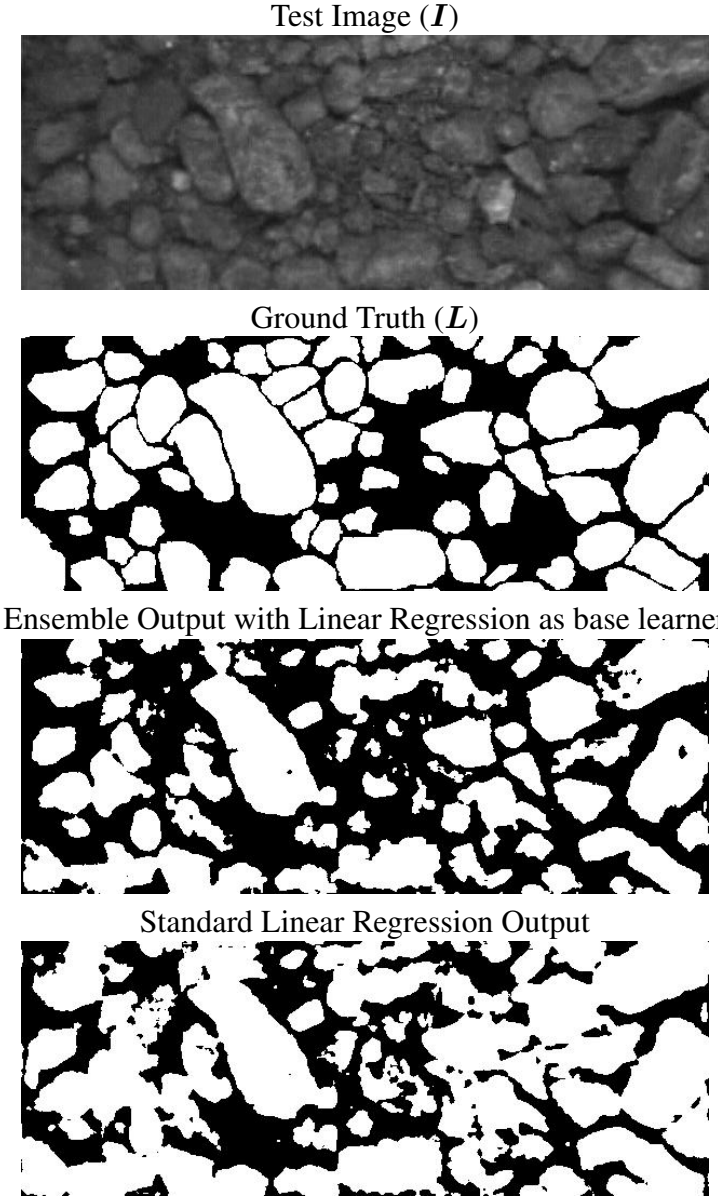



Figure 5.2: Output Decomposition test result versus regular linear regression.

3. Produce the output labels by combining the output of function approximators:

$$\tilde{\boldsymbol{L}} = \mathcal{F}^{-1}\left[\sum_{t=1}^{k} \mathcal{F}[\tilde{\Phi}_t^L] G_t^{\boldsymbol{\alpha}}\right]$$

The improved boundary detection and object separation can be readily observed in Figure 6.6, which depicts part of the detailed experimental results presented in the next chapter.

## 5.2 A Unifying Perspective for Generative Learning

Previous chapter presented window based automated feature extraction algorithms. We have demonstrated that PCA, ICA, PoE and Autoencoders are all specific cases of neural networks. Furthermore, the Field of Experts (FoE) algorithm corresponds to a special convolutional neural network. However, based on the Convolutional Encoding/Decoding algorithm of Section 4.4.1, one can show that all the aforementioned algorithms can in fact be represented as convolutional neural networks without the down-sampling layers. Recall that given an encoding matrix $\boldsymbol{W}_1$, the feature maps can be created via

$$\boldsymbol{f}_{\alpha}^{\boldsymbol{I}} = \boldsymbol{I} * \grave{\boldsymbol{w}}_{1,\alpha} \tag{5.7}$$

where $\grave{\boldsymbol{w}}_{1,\alpha}$ is the filter created from row $\alpha$ of matrix $\boldsymbol{W}_1$. Similarly we can reconstruct the image from the feature maps using the decoding matrix $\boldsymbol{W}_2$ either using the sliding window method or convolutional decoding:

$$\hat{\boldsymbol{I}} = \sum_{\alpha}^{k} \boldsymbol{f}_{\alpha}^{\boldsymbol{I}} * \grave{\boldsymbol{w}}_{2,\alpha} \tag{5.8}$$

where $\grave{\boldsymbol{w}}_{2,\alpha}$ is the filter created from column $\alpha$ of matrix $\boldsymbol{W}_2$. The image denoising process can then be abstractly represented by:

$$\hat{\boldsymbol{I}} = \sum_{\alpha}^{k} g_k(\boldsymbol{I} * \grave{\boldsymbol{w}}_{1,\alpha}) * \grave{\boldsymbol{w}}_{2,\alpha} \tag{5.9}$$

where $g_k$ represents the denoising function, which can be either hand-coded (as in ICA) or learned as in the case of PoE/FoE (presented in previous chapter). In turn, Equation 5.9 represent a convolutional neural network depicted in Figure 5.3. In summary, patch based neural networks can be transformed into an equivalent convolutional neural network and (in special cases) vise versa. As a concrete example consider an autoencoder mapping for learning a representation for $21 \times 21$ image patches. The patch-based mapping is then given by

$$N_{10}^{\boldsymbol{I}}(i,j) \mapsto \widehat{N_{10}^{\boldsymbol{I}}(i,j)}$$

Equivalently, one can train a convolutional network containing two convolutional layers:

$$N_{20}^{\boldsymbol{I}}(i,j) \mapsto \widehat{N_{0}^{\boldsymbol{I}}(i,j)}$$

which maps $41 \times 41$ patches to single pixels (recall $\boldsymbol{I}(i,j) = N_0^{\boldsymbol{I}}(i,j)$).

Thus one has a choice, either learn an autoencoder or an equivalent convolutional networks that performs regression. This fundamental duality not only offers

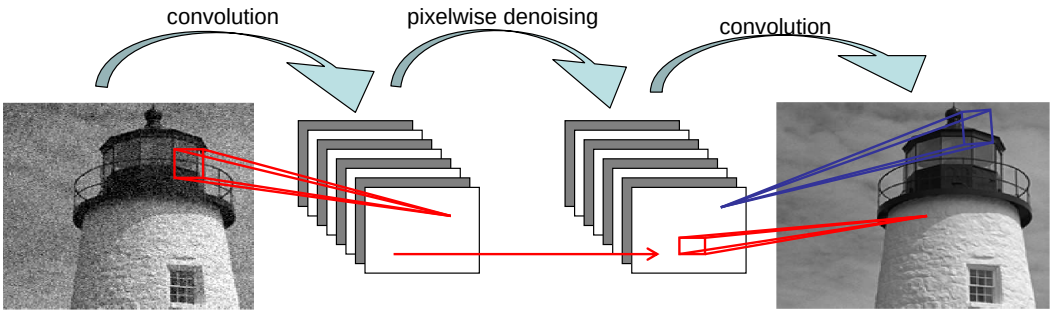



Figure 5.3: Architecture of a Convolutional Denoising Network. For the last convolutional layer, blue depicts the sliding window decoding method, while the red denotes the convolutional decoding equivalent. See Section 4.4.1 for more details.

a unique insight into the workings of both conventional and convolutional networks but can have practical benefits as well. These are explored within the context of discriminative learning in the next chapter.

### Gradient Descent

Based on the above discussion regarding the duality of patch-based networks and convolutional networks one must assume a similar duality between conventional gradient descent and convolutional gradient descent. In light of this, one may question the actual differences between the output of PoE and FoE presented previously. In [108], Roth presents an argument that PoE samples independent image patches while FoE samples independent images during the training process. This contrast in training methodology holds when comparing general patch based and convolutional based learning.

However, if one samples every patch in a given image, then the differences in learning procedures between patch based and convolutional networks vanish, at least for batch learning. Convolutional backpropagation simply allows one to forgo the need to extract a large redundant set of patches, while retaining the information used to calculate the gradient $\frac{\partial E_{mse}(\boldsymbol{x}, \boldsymbol{h}_{\boldsymbol{\omega}}(\boldsymbol{x}))}{\partial \boldsymbol{W}}$.

On the other hand, in the case of stochastic gradient descent, there are differences between patch-based and convolutional networks. The general contrast between stochastic versus batch gradient descent, is succinctly presented in [68]. One specific property of stochastic gradient descent is that it tends to converge much more rapidly, especially in the presence of non i.i.d. samples. Clearly overlapping image patches are not i.i.d. Images, however are independent[2] of one another provided they do not contain overlapping parts. Thus, in the case of patch

[2]While images may be independent of one another, there is still no guarantee they are identically distributed.

based stochastic gradient descent, network parameters would be updated after a presentation of a single randomly selected image patch. For convolutional networks, stochastic gradient descent entails weight updates after presentation of a single image. Therefore, one can conclude that stochastic gradient descent would be much more beneficial for patch-based while for convolutional networks the improvement would depend on the number of images available for training.

Often (but not always) the fastest converging learning method is mini-batching, whereby the gradient and subsequent weight updates are based on the presentation of a random subset of data. As the sample size increases, mini-batch learning converges to batch learning, while a decrease in sample size allows mini-batch learning to approximate stochastic gradient descent. For appropriate batch size, the algorithm can inherent beneficial properties of both stochastic and batch learning since the mini-batch gradient approximation is more accurate than that of stochastic gradient, while convergence on a redundant data set is still much faster than batch learning.

In light of the above discussion, we conjecture that if for a given image $\boldsymbol{I}$ all patches, $\{N_r(i,j)\}_{\forall(i,j)\in S}$ were used to learn the filters, both PoE and FoE[3] would converge to a very similar set of filters.

## 5.3 A Unified View for Discriminative Learning

The proposed output decomposition Mixture of Experts method has shed light on the importance of analyzing a neighborhood of pixel labels via feature extraction. Building on this idea, let us adopt a restricted view object segmentation as a mapping from image neighborhoods, $N^I_{r_1}(i,j)$, to label neighborhoods $N^L_{r_2}(i,j)$ via three functions:

$$f: \quad N^I_{r_1}(i,j) \quad \mapsto \boldsymbol{f}^I(i,j) \tag{5.10}$$

$$h: \quad \boldsymbol{f}^I(i,j) \quad \mapsto \boldsymbol{f}^L(i,j) \tag{5.11}$$

$$F: \quad \boldsymbol{f}^L(i,j) \quad \mapsto N^L_{r_2}(i,j) \tag{5.12}$$

where $f$ is the feature extraction function, $h$ is the classifier (or the set of classifiers as in the mixtures of specialists case), and $F$ is the (inverse) output synthesis function. As mentioned in previous chapters, manual feature extraction is a tedious, time consuming, and error prone task. The creation of the two feature extraction functions (one for images and another one for labels) seems to be even more difficult. Furthermore, if the concept of output decomposition is to be embedded into the data driven region growing framework, potentially six unique feature extraction functions may need to be defined, two for each of the thee mappings, $h_{region}$,

[3]Recall PoE stands for Product of Experts, while FoE stands for Fields of Experts

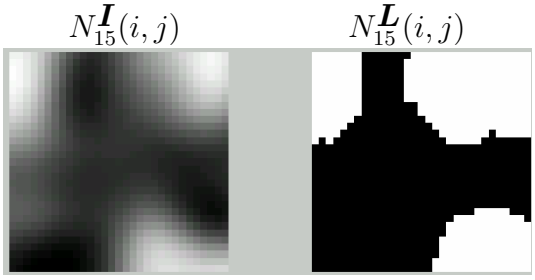



Figure 5.4: Example of $31 \times 31$ input/output training patches in correspondence with one another.

$h_{marker}$, $h_{boundary}$. This further exacerbates the problems with manually defining feature extraction functions and motivates the use of automated feature extraction methods described in the previous chapter.

Ideally, one should learn a specific set of features for each domain, for each mapping within DDRG, and for each input/output pair. To that end, previous section examined algorithms for automated feature extraction. We have also demonstrated the equivalence of several patch based methods (PCA/ICA/Autoencoders) to convolutional networks composed of two convolutional layers. Furthermore this duality between patch based and convolutional neural networks can be easily extended for use within the discriminative setting.

Consider a conventional patch based (i.e., non-convolutional) neural network with $l$ hidden layers implementing the mapping:

$$N_r^{\boldsymbol{I}}(i,j) \mapsto \boldsymbol{L}(i,j) \tag{5.13}$$

Letting $\boldsymbol{x} = vec(\boldsymbol{N}_r^{\boldsymbol{I}}(i,j))$ and $y = \boldsymbol{L}(i,j)$ the conventional neural net is given by Equation 3.13:

$$y_{\boldsymbol{\omega}}(\boldsymbol{x}) = \left(\boldsymbol{g}_{l+1} \circ \boldsymbol{W}_{l+1} \circ \boldsymbol{g}_l \circ \boldsymbol{W}_l \circ ... \circ \boldsymbol{g}_1 \circ \boldsymbol{W}_1\right)(\boldsymbol{x}) \tag{5.14}$$

Analogous to the discussion in Section 5.2, this corresponds to a convolutional network with a single convolutional layer and $l$ conventional pixelwise layers[4]. Now consider the mapping:

$$N_r^{\boldsymbol{I}}(i,j) \mapsto N_r^{\boldsymbol{L}}(i,j) \tag{5.15}$$

whereby the output is a label patch rather than a single pixel label. Figure 5.4 depicts the input and output patches. The conventional network is then given by:

$$\boldsymbol{y}_{\boldsymbol{\omega}}(\boldsymbol{x}) = \left(\boldsymbol{g}_{l+1} \circ \boldsymbol{W}_{l+1} \circ \boldsymbol{g}_l \circ \boldsymbol{W}_l \circ ... \circ \boldsymbol{g}_1 \circ \boldsymbol{W}_1\right)(\boldsymbol{x}) \tag{5.16}$$

[4]The meaning of 'layer' is different for conventional ANN than for CNN's. Recalling the discussion from previous chapter, a conventional linear autoencoder, e.g., ICA network, with a single hidden layer has an equivalent convolutional counterpart consisting of two convolutional layers.

where the network structure is similar to Equation 5.16 with the exception that $\boldsymbol{y} = vec(\boldsymbol{N}_r^{\boldsymbol{L}}(i,j))$. Now the equivalent convolutional network is given by:

$$\begin{aligned} \boldsymbol{f}_k^{\boldsymbol{I}} &= \boldsymbol{I} * \grave{\boldsymbol{w}}_{1,k},\ k = \{1, ...K\} \\ \boldsymbol{f}^{\boldsymbol{L}}(i,j) &= \left(\boldsymbol{g}_{l-1} \circ \boldsymbol{W}_{l-1} \circ ... \circ \boldsymbol{g}_2 \circ \boldsymbol{W}_2\right) \boldsymbol{f}^{\boldsymbol{I}}(i,j) \\ \hat{\boldsymbol{L}} &= \sum_{k=1}^{K} \boldsymbol{f}_k^{\boldsymbol{L}} * \grave{\boldsymbol{w}}_{l,k} \end{aligned} \tag{5.17}$$

From the point of view of sliding windows 4.4.1, patch labeling defined by Equation 5.16 produces many candidate labels for a given cite $(i,j)$. Recall that the neighborhood function $N_r(i,j)$ is symmetric, i.e., $(i',j') \in N_r(i,j) \implies (i,j) \in N_r(i',j')$. Hence, for a patch of radius $r$, cite $(i,j)$, will be labeled $(2r+1)^2$ times. This can be quite beneficial since the final label $\boldsymbol{L}(i,j)$ will incorporate predictions based on different context and will actually utilize information from a much larger $N_{2r}(i,j)$ image patch. For example, letting $r = 10$, results in Equation 5.13 mapping from/to $21 \times 21$ pixel patches. However, as noted above, the final label $\boldsymbol{L}(i,j)$ will actually incorporate data from a much larger $41 \times 41$ image patch. In turn, the averaging process can be viewed as a mixture-of-experts, where each expert uses different context for labeling location $(i,j)$. Therefore, under this formulation, the larger the output patch the larger the number of experts that label a particular cite $(i,j)$ (and the more regularized the output will be).

Since the formulation of the CNN model in Equation 5.17 is equivalent (and assuming conventional backpropagation will produce similar results) to convolutional backpropagation, the CNN model must also extract 'features' from the output labels. This result also ties in with Independent Component Regression presented in Section 4.4.5, whereby ICA and discriminative neural networks with one hidden layer are shown to be approximately equivalent. Under this model Regression is performed by running ICA on a joint vector $\boldsymbol{z} = [\boldsymbol{x}^T\ \boldsymbol{y}^T]^T$. The result of ICR is a matrix $\boldsymbol{A}$ approximately satisfying:

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{A}_x \\ \boldsymbol{A}_y \end{bmatrix} \boldsymbol{s}$$

where $\boldsymbol{s}$ is a vector of joint independent components. Online prediction of $\boldsymbol{y}$ is then performed as follows:

$$\hat{\boldsymbol{y}} = \boldsymbol{A}_y\, \boldsymbol{g}(\boldsymbol{A}_{\boldsymbol{x}}^{\dagger}\boldsymbol{x}) \tag{5.18}$$

where $\boldsymbol{g}$ is the denoising function taken to be a component-wise hyperbolic tangent due to assumption of a kurtotic distribution on $\boldsymbol{s}$. Clearly the result holds for deeper patch based networks and presents a new point of view for greedy layer-wise learning approaches presented in the previous section. In this case the feature extractors $\boldsymbol{A}_x, \boldsymbol{A}_y$ stay the same while the denoising function becomes more and more

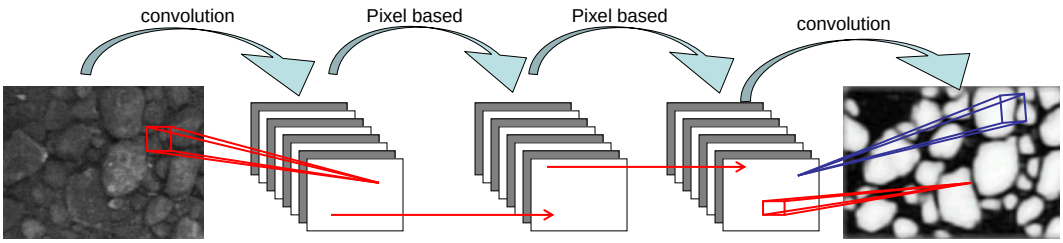



Figure 5.5: Convolutional Regression Network with three hidden layers (and four weight matrices). The columns of first and rows of last weight matrices are reshaped into filters for convolution, while the second and third weight matrices are used in a conventional pixel-wise manner. For the last convolutional layer, the blue depicts the sliding window decoding method, while the red depicts the convolutional decoding equivalent. See Section 4.4.1 for more details.

complex as a result of learning a deeper and deeper network via greedy layer-wise strategy. Furthermore, if $\boldsymbol{x}$ $\boldsymbol{y}$, are vectorized input/output patches then Equation 5.18 can be rewritten as a convolutional network as well, indicating that this type of convolutional neural network performs a joint feature extraction on the input/output vector $\boldsymbol{z}$.

This result casts a new light on the previously presented Output Decomposition Mixture of Experts. If OD-MoE uses two sets of filter banks then it is in fact an instance of a convolutional neural network whereby the two filter banks correspond to the first and last (convolutional) layers and the Mixture of Experts function approximators correspond to the inner layers or equivalently the denoising function $\boldsymbol{g}$.

## 5.4 Stacked Convolutional Regression Networks

This section presents an incremental strategy for learning deep (convolutional) neural networks, similar in spirit to both the greedy layer-wise learning, and to cascade correlation. The proposed algorithm, called stacked convolutional regression (SCR), directly maps image patches to label patches, and contains several novel features: (1) It uses a cascade correlation like approach for (2) learning a convolutional neural network in order to simultaneously analyze and map corresponding image and label patches as discussed in the previous section.

### 5.4.1 Network Growth and Training

Recalling the previously defined notation, let $\boldsymbol{x} = vec(\boldsymbol{N}_r^{\boldsymbol{I}}(i,j))$ and $\boldsymbol{y} = vec(\boldsymbol{N}_r^{\boldsymbol{L}}(i,j))$, respectively denote vectorized patches extracted from image $\boldsymbol{I}$ and

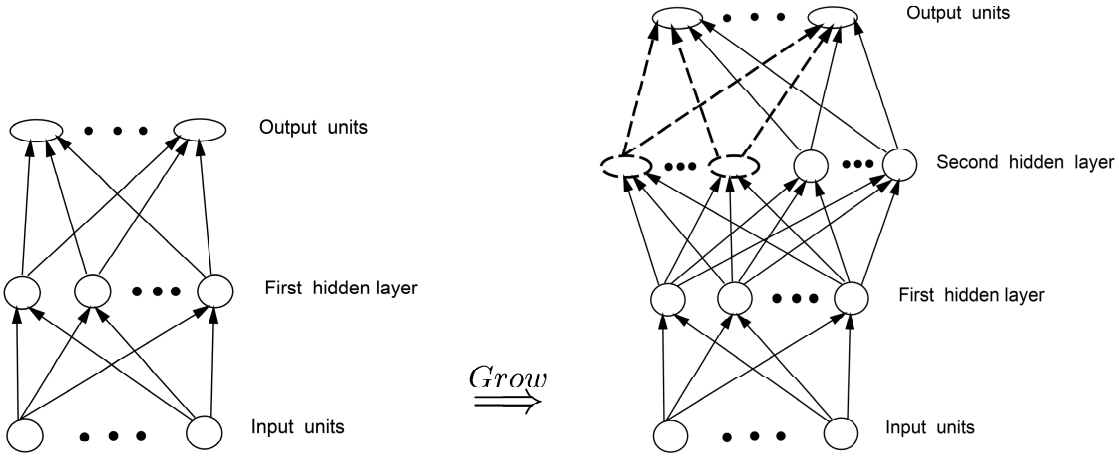



Figure 5.6: Stacked Convolutional Regression. At each iteration, the algorithm adds, and subsequently trains, one hidden layer. Figure modified from [98]

label image $\boldsymbol{L}$ depicted in Figure 5.4. The incremental learning algorithm progressively trains larger and larger networks using the previously learned weights as the initial starting points. The training algorithm starts with a standard neural network, $h_{\boldsymbol{\omega}_1}$ containing a single hidden layer and randomly initialized weight matrices $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$. Once trained, a new network, $h_{\boldsymbol{\omega}_2}$, is constructed containing two hidden layers and three weight matrices, $\boldsymbol{W}_3, \boldsymbol{W}_2, \boldsymbol{W}_1^{h_{\boldsymbol{\omega}_1}}$. The weight matrix $\boldsymbol{W}_1^{h_{\boldsymbol{\omega}_1}}$ is initialized with previously learned values from $h_{\boldsymbol{\omega}_1}$, while the other two matrices are randomly initialized. The following equations summarize the growing process.

$$h_{\boldsymbol{\omega}_1} = (\boldsymbol{g}_2 \circ \boldsymbol{W}_2 \circ \boldsymbol{g}_1 \circ \boldsymbol{W}_1)(\boldsymbol{x}) \quad (5.19)$$

$$h_{\boldsymbol{\omega}_2} = \left(\boldsymbol{g}_3 \circ \boldsymbol{W}_3 \circ \boldsymbol{g}_2 \circ \boldsymbol{W}_2 \circ \boldsymbol{g}_1 \circ \boldsymbol{W}_1^{h_{\boldsymbol{\omega}_1}}\right)(\boldsymbol{x}) \quad (5.20)$$

$$h_{\boldsymbol{\omega}_3} = \left(\boldsymbol{g}_4 \circ \boldsymbol{W}_4 \circ \boldsymbol{g}_3 \circ \boldsymbol{W}_3 \circ \boldsymbol{g}_2 \circ \boldsymbol{W}_2^{h_{\boldsymbol{\omega}_2}} \circ \boldsymbol{g}_1 \circ \boldsymbol{W}_1^{h_{\boldsymbol{\omega}_2}}\right)(\boldsymbol{x}) \quad (5.21)$$

$$\vdots$$

$$h_{\boldsymbol{\omega}_l} = \left(\boldsymbol{g}_{l+1} \circ \boldsymbol{W}_{l+1} \circ ... \circ \boldsymbol{g}_3 \circ \boldsymbol{W}_3^{h_{\boldsymbol{\omega}_{l-1}}} \circ \boldsymbol{g}_2 \circ \boldsymbol{W}_2^{h_{\boldsymbol{\omega}_{l-1}}} \circ \boldsymbol{g}_1 \circ \boldsymbol{W}_1^{h_{\boldsymbol{\omega}_{l-1}}}\right)(\boldsymbol{x}) \quad (5.22)$$

Training of each $h_{\boldsymbol{\omega}_k}$, can be done using any one of a multitude of gradient descent techniques, in particular those described in Chapter 3 and in Section 5.2. For the experimental results presented in the next chapter mini-batch gradient descent was used. The Growth process is illustrated in Figure 5.6. Note that all layers of $h_{\boldsymbol{\omega}_k}$ are modified by gradient descent. In contrast, cascade correlation modifies only the current, randomly initialized node, while greedy-layer-wise learning has two phases: (i) pre-training, and (ii) fine-tuning. On the other hand, the SCR algorithm incorporates both fine tuning and initialization into a single training iteration. While the newly created topmost layers are being initialized, fine tuning is performed at

the previously initialized layers. Figure 5.7 presents the learned filters at the first and last convolutional layers. Experiments (not presented) that split the pre training and fine tuning stages did not produce significantly different results.

Furthermore, examining the input/output filters for networks with only one hidden layer in Figure 5.7 reveals an interesting symmetry. The input and output filters are in correspondence with one another. This is in fact predicted by the ICR theorem in Section 4.4.5. The input/output filters for networks with only one hidden layer represent the joint latent feature extractors of $\boldsymbol{z} = [\boldsymbol{x}^T \ \boldsymbol{y}^T]^T$. As the networks grows, this correspondence 'diffuses' throughout the network.

### 5.4.2 Online Inference

The duality of patch-based and convolutional neural networks, enables us to pick and choose how to test and how to train. We chose to use patch based training but then transform the model into a fully convolutional network for online testing.

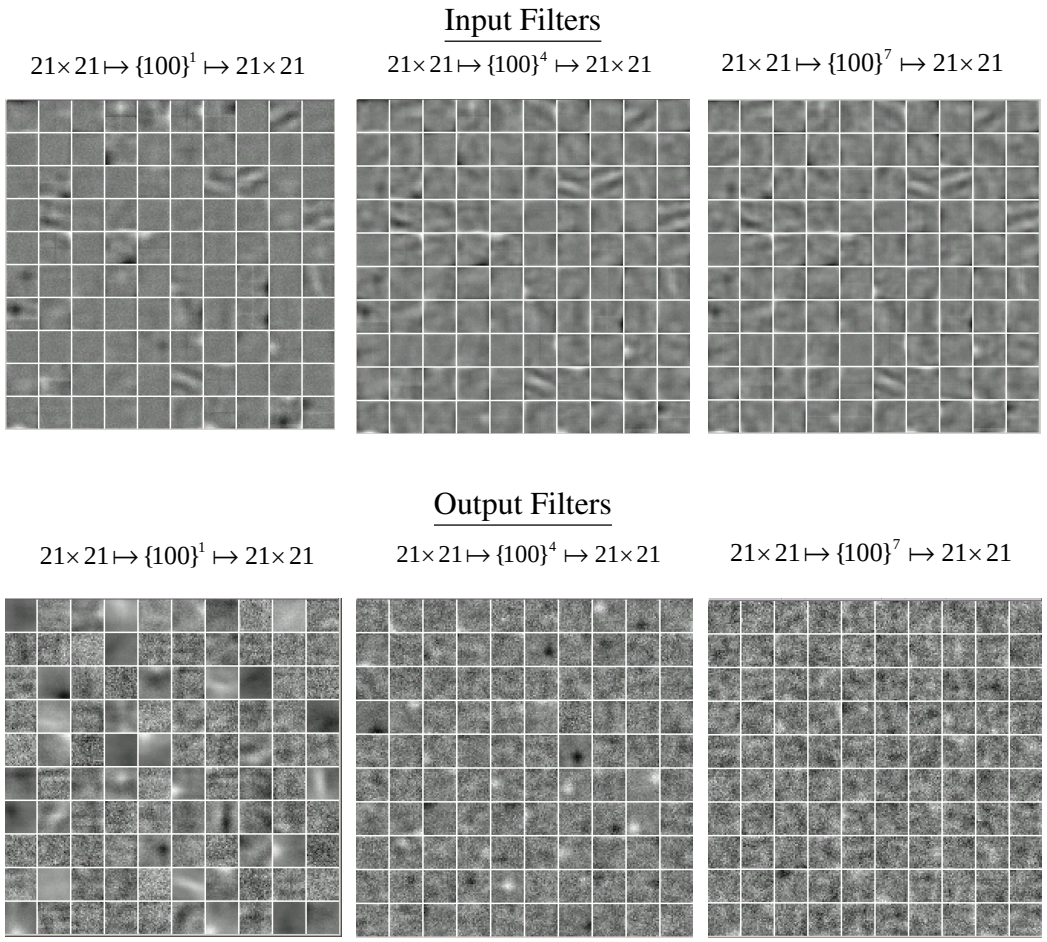



Figure 5.7: Evolution of Input/Output filters as a function of network depth. Both input and output patches are $21 \times 21$ pixels (i.e., $N_{10}(i, j)$). The size of each hidden is kept at 100 nodes. The number of hidden layers in a network is given by $h$ in $21 \times 21 \mapsto \{100\}^h \mapsto 21 \times 21$. **Top:** Evolution of the input filters. As the network grows, filters in the initial layer mature and sharpen. Some 'blank filters' develop only after the network has grown beyond a certain depth. **Bottom:** Re-learning of output synthesis filters. As the network grows the representation at the output layer dramatically changes. Initially, when the network is composed of a single hidden layer, the output filter is in correspondence with the input filters. As the training progresses the hidden layer absorb some of the representational complexity, leaving the output layer to re-learn simpler, less defined filters.

# Chapter 6

# Experimental Results

This chapter presents extensive experimental evaluation of several systems implementing DDRG, the **Data Driven Region Growing** framework for object delineation.

The first generation of the DDRG framework utilized only the $\boldsymbol{L}_{region}$ and $\boldsymbol{L}_{marker}$. The initial system [75], Classification Driven Watershed Segmentation (CDWS), implemented these mappings via traditional contextual pixel labeling, as in Equation 5.2, using a set of hand crafted features and logistic regression. Experiments focused on evaluating the efficacy of learning markers and region growing topology produced by the logistic regression. The second system, Heterogeneous stacking for classification driven watershed segmentation (HS-CDWS) [76], attempted to automate the feature extraction process by employing Independent Component Analysis. In conjunction the research proposed the heterogenous stacking concept, whereby the output of several classifiers, each trained on different targets, was combined by a second level classifier. This was the first system that was capable of instantiating the DDRG framework in a fully automated fashion. However, the performance was worse than the previous system utilizing hand-crafted features. In order to achieve state of the art performance, the third system utilized Stacked Convolutional Regression (SCR), as presented in the previous chapter. This system is capable of automatically learning all three mappings, $\boldsymbol{L}_{region}$, $\boldsymbol{L}_{marker}$, $\boldsymbol{L}_{boundary}$ and produced superior object segmentation results.

## 6.1 Classification Driven Watershed Segmentation

The Classification Driven Watershed Segmentation (CDWS) system represents the first generation of data driven region growing methods. It utilized two sets of logistic regression classifiers, based on Equation 3.10, to produce $\boldsymbol{L}_{region}$ and $\boldsymbol{L}_{marker}$. The feature extraction function was manually coded and is described in the next subsection.

### 6.1.1 Feature Extraction

Currently many different approaches for feature extraction have been proposed in the literature, with texture features being most relevant [43, 90, 102]. Common descriptions of texture include: (a) co-occurrence matrices [44], (b) local binary patterns [91], (c) random field methods [20]. In contrast, our feature extraction resembles Viola's team work in [13, 118] and to some extent that of [38], whereby a sequence of linear filters was used to produce the feature maps. In contrast, we allow more general algorithms to extract feature maps such as those depicted in Figure 6.1. The feature extraction procedure creates a multi-channel image, $\boldsymbol{f}$, whereby each pixel vector, $\boldsymbol{f}(i,j)$, corresponds to a training/test sample. Further-

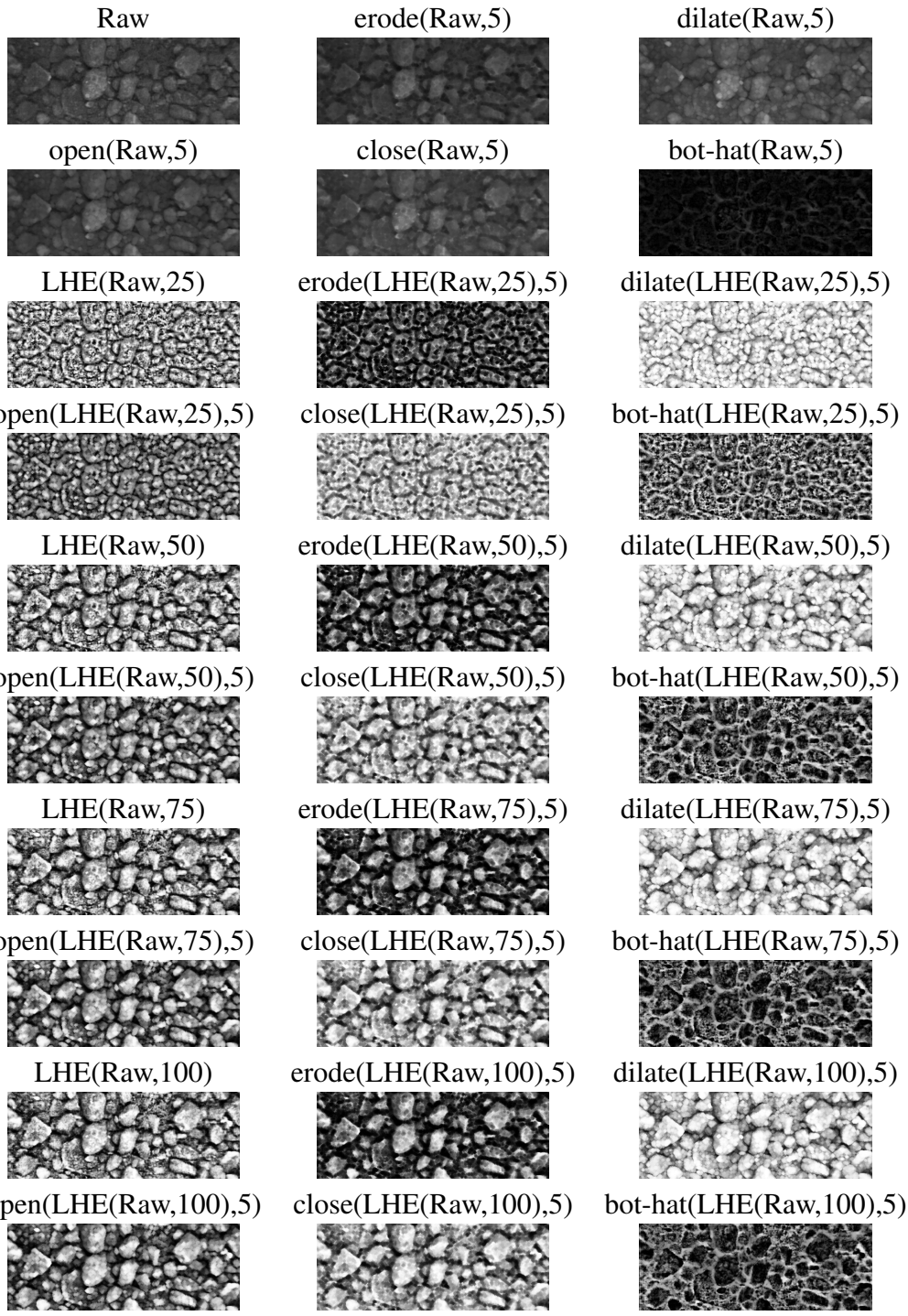



Figure 6.1: CDWS Features. The first 30 of the 150 extracted feature maps (at the highest resolution). Local Histogram equalization (LHE) used window sizes of $\{25, 50, 75, 100\}$. Morphological operations used a $5 \times 5$ square kernel.

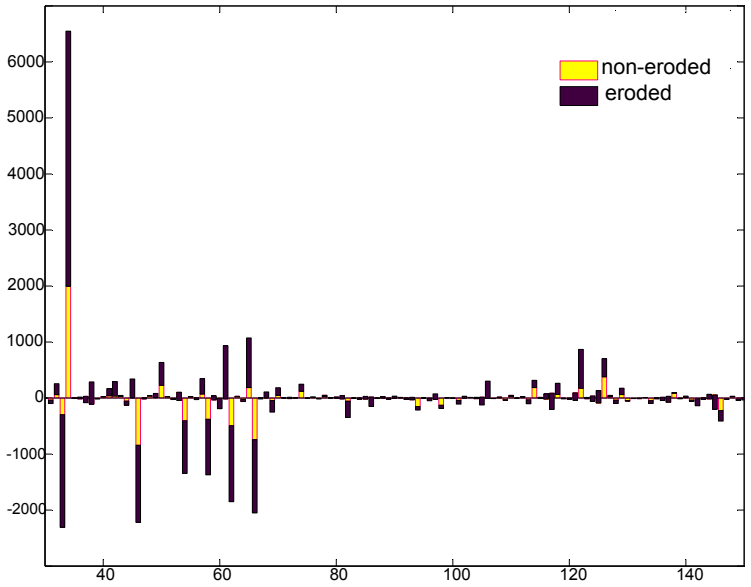



Figure 6.2: Weights (indexed on the X-axis) learned by the logistic regression algorithm using (a) standard ground truth, (b) eroded ground truth. Note that the weights for the first 30 features have been omitted since they were mostly zero in both cases.

more, notice in Figure 6.1 that a large number of simple and redundant feature maps, $f_i,\ i \in [1, ..., k]$, is produced, thereby removing the need to spend copious amounts of time creating a small set of highly domain specific features. In fact, the proposed approach does the exact opposite, it tries to create a large number of weak features and expects the classifier to weight them according to their relevance during training. In general, many of the features may turn out to be irrelevant for a given application. However, our approach begins with a large set of features in order to be application-independent, at the expense of increased training complexity. Figure 6.2 presents the typical feature weights learned by the logistic regression algorithm from equation 3.10 when trained using (a) standard ground truth, (b) using eroded ground truth. We have omitted the weights for the first thirty features, which were zero in both cases, indicating that the high resolution features are essentially considered irrelevant for this classification task. The fact that the high resolution features are not used in classification, implicitly indicates that using raw pixel values as input features for classification will result in poor performance, and further motivates the need for feature extraction.

### 6.1.2 Experimental Procedure

In order to compare CDWS against the state-of-the-art methods and variants of CDWS, a granulometry expert manually segmented nine images containing oil sand ore (see Figure 3.1 for an example). Each image was $236 \times 637$ pixels. For all experiments, a leave-one-out (LOOCV) testing strategy was employed, whereby the system was trained on eight of the nine images and the remaining image was used for testing. The procedure was repeated with every image being the test image once.

In order to train the necessary classifiers, for each image 30 feature maps were extracted, shown in Figure 6.1, at five resolutions for a total of 150 feature maps. In other words, for each pixel a feature vector with 150 components was created. Both $h_{region}$ and $h_{marker}$, i.e., the classifiers that respectively produce $P_{region}$ and $P_{marker}$, were trained as follows. To reduce computational and memory requirements one classifier was trained on one of the nine images in the database using logistic regression[1] [45]. For the LOOCV testing strategy, the probability maps ($P_{region}$ and $P_{marker}$) were created by averaging out probability maps produced by the eight classifiers that did not "see" the current test image. Formally, for input image $\boldsymbol{I}_i$:

$$P_{type} = \frac{1}{n-1} \sum_{j \neq i}^{n} h_{type,\boldsymbol{\omega}_j}(\boldsymbol{I}_i)$$

where $type \in \{marker, region\}$. In essence, $h_{type} = \{h_{type,j}\}$ is an ensemble of classifiers. The outputs of the ensemble members are averaged together to produce the probability map. The procedure is identical for creating both types of classifiers. To produce object markers and boundaries for CDWS, the probability maps are always thresholded at $\tau_1 = \tau_2 = 0.5$ (see Figure 2.6). **The metrics used for evaluation are presented in Appendix C on p. 138**.

### 6.1.3 Experiment 1

To establish whether the proposed modifications to the watershed algorithm indeed have merit, several variants of the CDWS algorithm were constructed. In order to ascertain if eroding the ground-truth to train a specialized maker identification classifier is indeed beneficial, the first variant of CDWS extracted markers from $P_{region}$ using several threshold values. For comparison, the proposed CDWS algorithm, as presented in Figure 2.6, was evaluated using various sizes of erosion kernels applied to the ground truth.

[1] Other classifiers, such neural networks, decision trees, naive Bayes (tested on smaller sets) performed similarly. However, logistic regression was much faster in both training and testing. All experiments were conducted using PrTools [29] Matlab toolbox.

In addition, to ascertain the benefits of using $1 - P_{region}$ as the topological function utilized by the watershed algorithm, several alternative methods were implemented which used (a) the gradient of the raw image, (b) gradient of $P_{marker}$, (c) gradient of $P_{region}$, (d) the negative of the raw image, (e) $1 - P_{marker}$, and (f) the CDWS algorithm utilizing $1 - P_{region}$.

Experimental results for the CDWS variants are presented in Figure 6.3. By comparing the top graph of Figure 6.3 to the middle graph, it is clear that using markers extracted from $P_{marker}$ significantly improves the label score. Furthermore, CDWS appears less sensitive to the size of the erosion kernel, than to the value of the threshold used for extracting markers from $P_{region}$. Also note that simply thresholding $P_{region}$, i.e., not utilizing the watershed algorithm, to produce the final object labeling results in a label score of approximately $0.3$. In addition, flooding the inverted object-boundary probability results in superior performance than the other approaches as shown by the bottom graph of Figure 6.3.

### 6.1.4 Experiment 2

To compare the proposed CDWS algorithm against standard granulometry based algorithms the WipFrag and OSA systems were procured (see Appendix A for their description). Table 6.2 presents the experimental results. In addition, for each system, performance was examined using several (mutually exclusive) post-processing techniques. Since noisy segmentation and labeling is inevitable, the final output was filtered using size filters that remove particles smaller than a predefined size. By producing results using several filter sizes (applied to both the output images and the ground truth images) the object size that is most problematic for each system can be quantitatively determined. An alternative post processing step, carried out instead of the first, used morphological opening (also applied to both the output images and the ground truth images) with the aim of removing bridge pixels connecting two objects. The aim of this experiment was to determine if the proposed CDWS system can indeed do more than just remove bridge pixels, which can be easily accomplished by the simplest of the post processing techniques. As results in Table 6.3 indicate, regardless of the postprocessing method, the CDWS algorithm still outperforms both WipFrag and OSA. Furthermore, the worst output of CDWS (no post processing) is better than the best output of either WipFrag or OSA as indicated by **all** evaluation criteria[2].

[2]In terms of computational cost, a direct comparison at present time is difficult. Both OSA and WipFrag are industrial systems implemented in C, and have been extensively optimized to allow for real-time image processing. On the other hand, CDWS is implemented in Matlab without any optimization and takes about 5 min to process an image. The main cost in CDWS is feature extraction. However, by examining Figure 6.2 we can see that very few features receive high weights and in turn, are actually relevant to the task at hand. Hence, by adding a feature selection stage, to select only the relevant features rather than using all 150 features, the runtime performance of the system

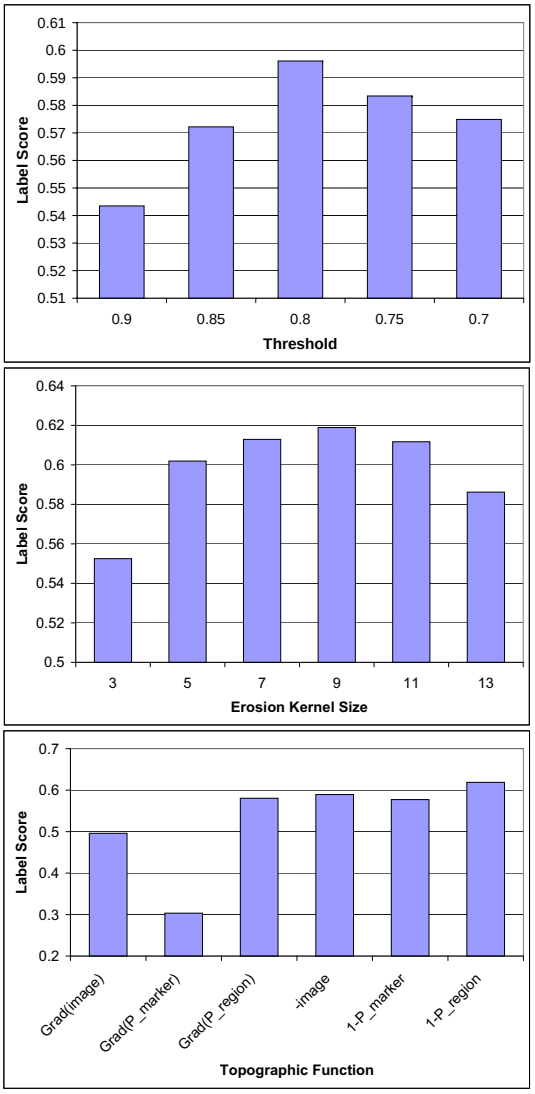



Figure 6.3: **Top:** Effectiveness of markers extracted by thresholding $P_{region}$ at various values of $\tau$ (cf. equation 2.4). **Middle:** Effectiveness of markers extracted by thresholding $P_{marker}$ at $\tau = 0.5$). Output of the marker classifier ($h_{marker}$) was varied by applying erosion with different sizes of structuring element, $B$, to the ground truth image $L$, during training (see equation 2.5 for details). Comparison with the top table indicates that using a classifier for marker identification improves performance. The latter approach appears less sensitive to the size of the erosion kernel than the former approach with respect to threshold level. **Bottom:** Watershed segmentation using different Topographic Functions. In each case the same set of markers, extracted from $P_{marker}$ at $\tau = 0.5$, was used. Using $1 - P_{region}$, i.e., the inverted object-background probability map, clearly produces the best results.

can be significantly improved.

Table 6.1: **Top:** Effectiveness of markers extracted by thresholding the object-boundary probability map produced by the region classifier. **Middle:** Markers extracted by thresholding (at 0.5) the marker probability map. The output of the marker classifier was varied by using erosion with kernels of different sizes to modify the ground truth at training time. Comparison between the top and bottom tables indicates that using a marker identification classifier improves performance. The latter approach appears less sensitive to the size of the erosion kernel than the former approach with respect to threshold level. **Bottom:** Watershed segmentation using different flood images. Using $1-\boldsymbol{P}_{region}$, i.e., the inverted object-background probability map, produces the best results.

| marker threshold | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| 0.9 | 0.7446 | 0.8271 | 0.8594 | 0.8487 | 0.5435 |
| 0.85 | 0.7469 | 0.8286 | 0.86 | 0.8511 | 0.5722 |
| **0.8** | **0.7481** | **0.8295** | **0.8603** | **0.8524** | **0.5961** |
| 0.75 | 0.7483 | 0.8296 | 0.8601 | 0.8529 | 0.5834 |
| 0.7 | 0.7492 | 0.8301 | 0.8602 | 0.854 | 0.5749 |

| gt erosion | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| 3 | 0.7602 | 0.8382 | 0.8657 | 0.8626 | 0.5525 |
| 5 | 0.7603 | 0.8385 | 0.8667 | 0.8618 | 0.6019 |
| 7 | 0.76 | 0.8384 | 0.8671 | 0.8609 | 0.6129 |
| 9 | 0.7602 | 0.8385 | 0.8669 | 0.8614 | 0.6189 |
| 11 | 0.7598 | 0.8383 | 0.8668 | 0.861 | 0.6117 |
| 13 | 0.7586 | 0.8373 | 0.8661 | 0.8601 | 0.5862 |

| flood image | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| Grad(image) | 0.7326 | 0.8213 | 0.87 | 0.8232 | 0.496 |
| Grad(pmap_marker) | 0.6968 | 0.7972 | 0.8631 | 0.7841 | 0.3034 |
| Grad(pmap_standard) | 0.7435 | 0.8299 | 0.8786 | 0.8292 | 0.5806 |
| image | 0.6972 | 0.7971 | 0.8614 | 0.786 | 0.2564 |
| 1-pmap_marker | 0.7557 | 0.8357 | 0.8677 | 0.855 | 0.5773 |
| 1-pmap_standard | 0.7602 | 0.8385 | 0.8669 | 0.8614 | 0.6189 |

Table 6.2: Performance of OSA, WipFrag and CDWS systems.

| System | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| OSA | 0.6838 | 0.7825 | 0.8357 | 0.7907 | 0.5522 |
| WipFrag | 0.5913 | 0.6493 | 0.6587 | 0.8529 | 0.3574 |
| CDWS | **0.7602** | **0.8385** | **0.8669** | **0.8614** | **0.6189** |

Table 6.3: Results for OSA (**top**), WipFrag (**middle**) and CDWS(**bottom**), comparing two different post processing approaches. First approach filtered out particles smaller than (**SF**). Second approach used morphological opening with a square kernel of size (**OF**). Regardless of the postprocessing type used CDWS outperforms both OSA and WipFrag.

OSA

| Post Processing | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| none | 0.6838 | 0.7825 | 0.8357 | **0.7907** | 0.5522 |
| SF = 100 | 0.6921 | 0.7925 | 0.8548 | 0.7849 | 0.5692 |
| SF = 200 | **0.6979** | **0.7997** | 0.8678 | 0.7813 | 0.5802 |
| SF = 400 | 0.6829 | 0.7973 | 0.861 | 0.7678 | **0.5871** |
| SF = 800 | 0.6481 | 0.7971 | 0.8331 | 0.7451 | 0.5866 |
| OF = 3 | 0.6839 | 0.7871 | 0.8553 | 0.774 | 0.5647 |
| OF = 5 | 0.6834 | 0.7921 | 0.8814 | 0.753 | 0.578 |
| OF = 7 | 0.6709 | 0.7893 | 0.9083 | 0.7199 | 0.5853 |
| OF = 9 | 0.6326 | 0.7689 | **0.9254** | 0.6667 | 0.5572 |

WipFrag

| Post Processing | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| none | **0.5913** | 0.6493 | 0.6587 | **0.8529** | 0.3574 |
| SF = 100 | 0.5857 | 0.6475 | 0.6599 | 0.8392 | 0.3635 |
| SF = 200 | 0.5831 | 0.6545 | 0.6706 | 0.8175 | 0.3779 |
| SF = 400 | 0.5674 | 0.6662 | 0.6819 | 0.7728 | 0.4016 |
| SF = 800 | 0.5399 | **0.6903** | 0.6756 | 0.7276 | **0.4223** |
| OF = 3 | 0.5897 | 0.6546 | 0.668 | 0.8345 | 0.371 |
| OF = 5 | 0.5848 | 0.6576 | 0.6773 | 0.8113 | 0.3791 |
| OF = 7 | 0.5759 | 0.6626 | 0.6952 | 0.7714 | 0.3974 |
| OF = 9 | 0.566 | 0.6718 | **0.7261** | 0.7206 | 0.4163 |

CDWS

| Post Processing | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| none | 0.7602 | 0.8385 | 0.8669 | 0.8614 | 0.6189 |
| SF = 100 | 0.762 | 0.841 | 0.8725 | 0.8583 | 0.6246 |
| SF = 200 | 0.7624 | 0.8426 | 0.8775 | 0.854 | 0.628 |
| SF = 400 | 0.7524 | 0.8414 | 0.8702 | 0.8484 | 0.6325 |
| SF = 800 | 0.7344 | 0.8483 | 0.8572 | 0.8378 | 0.6359 |
| OF = 3 | 0.7599 | 0.8402 | 0.8774 | 0.8509 | 0.6383 |
| **OF = 5** | 0.7555 | 0.8392 | 0.8886 | 0.8352 | **0.6444** |
| OF = 7 | 0.7433 | 0.8336 | 0.9004 | 0.8105 | 0.6419 |
| OF = 9 | 0.721 | 0.8215 | 0.9113 | 0.776 | 0.6349 |

### 6.1.5 Experiment 3 - Color Image Segmentation

To demonstrate the general applicability of the algorithm, the CDWS system was applied on aerial images of forest plantations depicted in Figure 6.4 (see Appendix

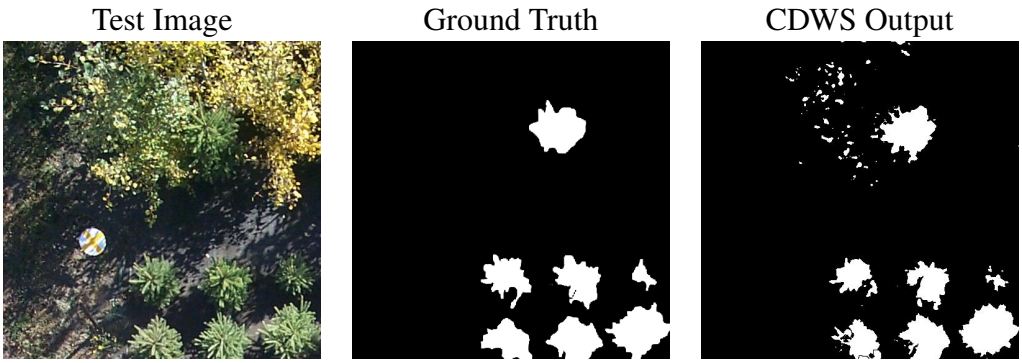



Figure 6.4: An aerial image taken over a mixed vegetation plot along with the desired interpretation of spruce canopies labeled in white and the output of CDWS shown on the left. Notice the non-uniform lighting conditions as well as the variability in shape and size of the target objects. Furthermore, the ground-truth image at times only labels the 'sun-crown' (i.e., the portion of the crown lit up by the sun) as opposed to the full tree canopy. These are some of the challenges faced by an image interpretation system within the forestry domain.

Table 6.4: CDWS performance on aerial forest images

| | I/U | acc | prec | recall | label score |
|---|---|---|---|---|---|
| **mean** | **0.70** | **0.92** | **0.83** | **0.84** | **0.47** |
| stdev | 0.12 | 0.04 | 0.15 | 0.10 | 0.12 |

B for an overview of the domain). First, the 18 ($512 \times 512$ pixel) RGB images were converted to HSV color space. Next, for each HSV image the previously described feature extraction scheme was applied to each of the H, S, V channels separately (producing 450 features). The same LOOCV experimental procedure as described in section 6.1.2 was used to evaluate the performance. Results are shown in Table 6.4. Unfortunately previous results for this domain have reported only the intersection-over-union (I/U) scores and are therefore difficult to compare with results in Table 6.4. Using Adaptive Object Recognition system (ADORE) in [72], an LOOCV strategy over 34 ($256 \times 256$ pixel) images, achieved an I/U score of $0.54$. In [73], using ADORE coupled with Hierarchical Markov Random Fields from [17], the LOOCV procedure achieved an I/U score of $0.61$ over 3 ($1536 \times 1024$ pixel) images. On their own, hierarchical MRF produced an I/U score of $0.48$ using the same 3 ($1536 \times 1024$ pixel) images. In contrast, the proposed CDWS system achieved an I/U score of 0.70 over 18 ($512 \times 512$ pixel) images, a considerable improvement over the much more sophisticated Reinforcement Learning and Multi-Resolution MRF based systems.

## 6.2 Heterogeneous Stacking and ICA for Classification Driven Watershed Segmentation, [76]

Section 3.4.3 proposed heterogeneous stacking for building an automated object segmentation system, whereby the general system is succinctly described by a set of mappings presented in Figure 3.5. The following experiments evaluated several variants of the aforementioned heterogeneous stacking framework. In contrast to CDWS, which utilized manual feature design coupled, this system experimented with Independent Components Analysis (ICA) to automatically extract features from raw image patches [51] as in Equations 4.11 on p. 47 and 4.12 on p. 49. First, the feature extraction matrix $\boldsymbol{A}^{\dagger}$ was learned using an unlabeled set of images. Next, given a training image/label pair, the algorithm: (i) extracts features, $\boldsymbol{f}^{\{0\}}$, using $\boldsymbol{A}^{\dagger}$, and (ii) produces $\boldsymbol{L}_{eroded}, \boldsymbol{L}_{dilated}, \boldsymbol{L}_{e'}, \boldsymbol{L}_{d'}$ by applying morphological operations on the ground truth image $\boldsymbol{L}$. Subsequently, five $\mathfrak{L}_0$ classifiers are trained using ICA features as input and label images as targets. The classifiers output probability maps $\boldsymbol{P}^{\{0\}}_{type}$, $type \in \{region, eroded, dilated, e', d'\}$. A second round of feature extraction is then carried out on the newly extracted probability maps, producing second order features, $\boldsymbol{f}_{\{1\}}$, that serve as the input to train two $\mathfrak{L}_1$ classifiers. In turn, the second order classifiers produce two probability maps, $\boldsymbol{P}^{\{1\}}_{region}$ and $\boldsymbol{P}^{\{1\}}_{eroded}$, used for creating the topological landscape and markers. The last step employs the standard watershed algorithm for producing the final output of the system, $\boldsymbol{L}^{\{ws\}}$.

### 6.2.1 Experimental Procedure

To test HS-CDWS, we used the same nine, $236 \times 637$ pixel, images containing oil sand ore (see Figure 3.1). Using a different set of **unlabeled** oil sand ore images, we learned a generative ICA model using the FastICA algorithm [58]. This ICA model was estimated using $100,000$ randomly selected patches, each $16 \times 16$ pixels in order to learn 49 Gabor-like filters. To provide multi-resolution information, two gaussian filters were applied to each ICA filter response, thereby producing 150 features for each pixel (147 multi-resolution ICA features + 3 multi-resolution raw pixel values from the original image). This constituted $\boldsymbol{f}^{\{0\}}$, the input to the $\mathfrak{L}_0$ classifiers. The target outputs, $\boldsymbol{L}^{\{0\}}$, included the original ground truth as well as the derived targets depicted in Figure 3.6. For all experiments a leave-one-out cross validation (LOOCV) testing strategy was used, whereby each system was trained on eight of the nine images with the remaining image used for testing. The procedure was repeated with every image being a test image once.

To reduce computational complexity, for each target output we trained a set of classifiers, one for each training image. Hence for each cross-validation fold we trained $8 \times 5 = 40$ classifiers corresponding to eight training images and five target

outputs. This strategy effectively reduces the memory overhead needed for training, since the number of training examples is reduced by a factor of eight. Formally, for test image $I_i$ :

$$\boldsymbol{P}^{\{0\}}_{type} = \frac{1}{n-1}\sum_{j\neq i}^{n} h^{\{0\}}_{type,j}$$

where $type \in \{region, eroded, dilated, e', d'\}$. To take advantage of the rich information contained in the probability maps $\boldsymbol{P}^{\{0\}}$, a second round of feature extraction was carried out, where a bank of gaussian filters was used to extract multi-resolution features $\boldsymbol{f}^{\{1\}}$, which is subsequently referred to as the "extended feature set" in the following subsections. To fuse the information into $\mathfrak{L}_1$ probability maps, we trained a set of $\mathfrak{L}_1$ classifiers to produce the mapping: $\boldsymbol{f}^{\{1\}} \mapsto \boldsymbol{P}^{\{1\}}_{type}$, with $type \in \{region, eroded\}$. As in [95], we used an internal procedure analogous to LOOCV in order to maximize generalization accuracy. Both $\mathfrak{L}_0$ level and $\mathfrak{L}_1$ level classification was done using logistic regression as implemented by the PrTools [29] Matlab toolbox.

### 6.2.2 Experiment 1

To examine the efficacy of the proposed algorithm, three sets of systems were tested. First, a standard CDWS system (no stacking) was created using ICA features called **ICA-CDWS**. Next, the second system, named **ICA-HS-CDWS** system, trained $\mathfrak{L}_1$ level classifiers directly on the output of the five $\mathfrak{L}_0$ probability maps produced by classifiers trained on standard ground truth as well as new targets derived from the ground truth. Note that this version of the system did not perform the second round of feature extraction, i.e., $\boldsymbol{f}^{\{1\}} = \boldsymbol{P}^{\{0\}}$. Finally, the third system, **MR-ICA-HS-CDWS**, had the same set-up as the second system, but used the extended set of multi-resolution features extracted from $\boldsymbol{P}^{\{0\}}$. Results, presented in Table 6.5 and Figure 6.6, clearly demonstrate the improvement gained by using heterogeneous stacking together with features extracted from $\boldsymbol{P}^{\{0\}}$. Notice that heterogeneous cascades, with interleaved feature extraction, produce the best results on average and improve upon the scores for essentially every performance metric in every image. The only exception being image 5, where the recall score was slightly degraded by the proposed system. In all other cases the MR-ICA-HS-CDWS system was able to improve performance in comparison to the base (ICA-CDWS) classification. Interestingly, the recall score for image five is one of only two images where the stacking without feature extraction outperformed stacking with interleaved feature extraction. We believe better features can fix this anomaly and further improve performance. The probability that there are no statistically significant differences in performance as calculated by the paired student's t-test for each performance metric is respectively: $0.00004, 0.00001, 0.00000, 0.01942, 0.00049$, (for I/U, Accuracy,

Table 6.5: Performance comparison of base classification (L0) to heterogeneous stacking (L1). For each experimental condition the tables represent leave-one-out cross validation results. Based on the 2-sided paired student's t-test, the probability that there are no statistically significant differences in performance of MR-ICA-HS-CDWS vs ICA-CDWS systems is respectively: $0.00004, 0.00001, 0.00000, 0.01942, 0.00049$, for I/U, Accuracy, Precision, Recall and Label scores.

**ICA-CDWS**

| Image | jacq | acc | prec | recall | label score |
|---|---|---|---|---|---|
| 1 | 0.68 | 0.77 | 0.79 | 0.83 | 0.51 |
| 2 | 0.74 | 0.83 | 0.80 | 0.91 | 0.62 |
| 3 | 0.73 | 0.81 | 0.84 | 0.84 | 0.56 |
| 4 | 0.72 | 0.79 | 0.86 | 0.81 | 0.51 |
| 5 | 0.69 | 0.78 | 0.79 | 0.84 | 0.52 |
| 6 | 0.76 | 0.83 | 0.87 | 0.86 | 0.62 |
| 7 | 0.73 | 0.80 | 0.84 | 0.84 | 0.51 |
| 8 | 0.66 | 0.76 | 0.75 | 0.85 | 0.54 |
| 9 | 0.73 | 0.80 | 0.83 | 0.85 | 0.54 |
| **mean** | **0.71** | **0.80** | **0.82** | **0.85** | **0.55** |
| stdev | 0.03 | 0.02 | 0.04 | 0.03 | 0.04 |

**MR-ICA-HS-CDWS**

| Image | jacq | acc | prec | recall | label score |
|---|---|---|---|---|---|
| 1 | 0.71 | 0.80 | 0.82 | 0.84 | 0.59 |
| 2 | 0.77 | 0.85 | 0.83 | 0.91 | 0.63 |
| 3 | 0.76 | 0.84 | 0.87 | 0.86 | 0.62 |
| 4 | 0.74 | 0.81 | 0.88 | 0.82 | 0.54 |
| 5 | 0.71 | 0.80 | 0.83 | 0.83 | 0.57 |
| 6 | 0.81 | 0.86 | 0.89 | 0.89 | 0.69 |
| 7 | 0.77 | 0.84 | 0.88 | 0.86 | 0.53 |
| 8 | 0.71 | 0.80 | 0.79 | 0.87 | 0.57 |
| 9 | 0.74 | 0.81 | 0.85 | 0.85 | 0.61 |
| **mean** | **0.75** | **0.83** | **0.85** | **0.86** | **0.60** |
| stdev | 0.03 | 0.02 | 0.04 | 0.03 | 0.05 |

**ICA-HS-CDWS**

| Image | jacq | acc | prec | recall | label score |
|---|---|---|---|---|---|
| 1 | 0.70 | 0.79 | 0.81 | 0.83 | 0.56 |
| 2 | 0.77 | 0.86 | 0.83 | 0.91 | 0.61 |
| 3 | 0.75 | 0.83 | 0.86 | 0.85 | 0.61 |
| 4 | 0.74 | 0.81 | 0.88 | 0.82 | 0.54 |
| 5 | 0.70 | 0.79 | 0.81 | 0.84 | 0.55 |
| 6 | 0.79 | 0.85 | 0.88 | 0.89 | 0.65 |
| 7 | 0.75 | 0.82 | 0.86 | 0.86 | 0.55 |
| 8 | 0.68 | 0.79 | 0.77 | 0.86 | 0.56 |
| 9 | 0.73 | 0.81 | 0.84 | 0.86 | 0.56 |
| **mean** | **0.74** | **0.82** | **0.84** | **0.86** | **0.58** |
| stdev | 0.03 | 0.03 | 0.04 | 0.03 | 0.04 |

Precision, Recall and Label scores) indicating that the performance of MR-ICA-HS-CDWS is superior to that of the ICA-CDWS system. In addition, to compare the three aforementioned systems against previous results, Table 6.6 displays data from the original CDWS research ([75]). Several points are immediately apparent. First the ICA features are weaker than the original hand-crafted features used by CDWS. To some extent this is not surprising, as ICA extracted 49 linear features at three resolutions. In contrast, CDWS utilized 30 hand-crafted non-linear extraction procedures (e.g., morphological operators) at four resolutions. We believe non-linear feature extraction methods (e.g., non-linear PCA) can improve performance and expect to pursue this line of research in the future. However, despite the shortcomings of ICA, the MR-ICA-HS-CDWS system, a *fully automated* algorithm was able to achieve results very similar to those of CDWS utilizing hand-crafted features.

Table 6.6: Performance of OSA, WipFrag, and original CDWS systems against CDWS using ICA and Heterogeneous Stacking

| System | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| OSA | 0.68 | 0.78 | 0.84 | 0.79 | 0.55 |
| WipFrag | 0.59 | 0.65 | 0.66 | 0.85 | 0.36 |
| **CDWS** | **0.76** | **0.84** | **0.87** | **0.86** | **0.62** |
| ICA->CDWS | 0.71 | 0.80 | 0.82 | 0.85 | 0.55 |
| HS(ICA)->CDWS | 0.74 | 0.82 | 0.84 | 0.86 | 0.58 |
| **MR-HS(ICA)->CDWS** | **0.75** | **0.83** | **0.85** | **0.86** | **0.60** |

### 6.2.3 Experiment 2

To further test and isolate the benefits of stacking, we applied the same experimental procedure to aerial images of forest plantations as in the previous set of experiments on CDWS (c.f., Section 6.1.5). For each HSV image the previously described **manual** feature extraction scheme was applied to each of the H, S, V channels separately (producing 450 features). These features were used to train the $\mathfrak{L}_0$ classifiers. Subsequently the outputs of $\mathfrak{L}_0$ classifiers were filtered using the same gaussian filters as described in the last section. To evaluate performance, the same LOOCV experimental procedure as described in section 6.1.2 was used. Results are shown in Table 6.7. Comparing the top table to the bottom table demonstrates the improvement in generalization performance as a result of utilizing the heterogeneous stacking procedure.

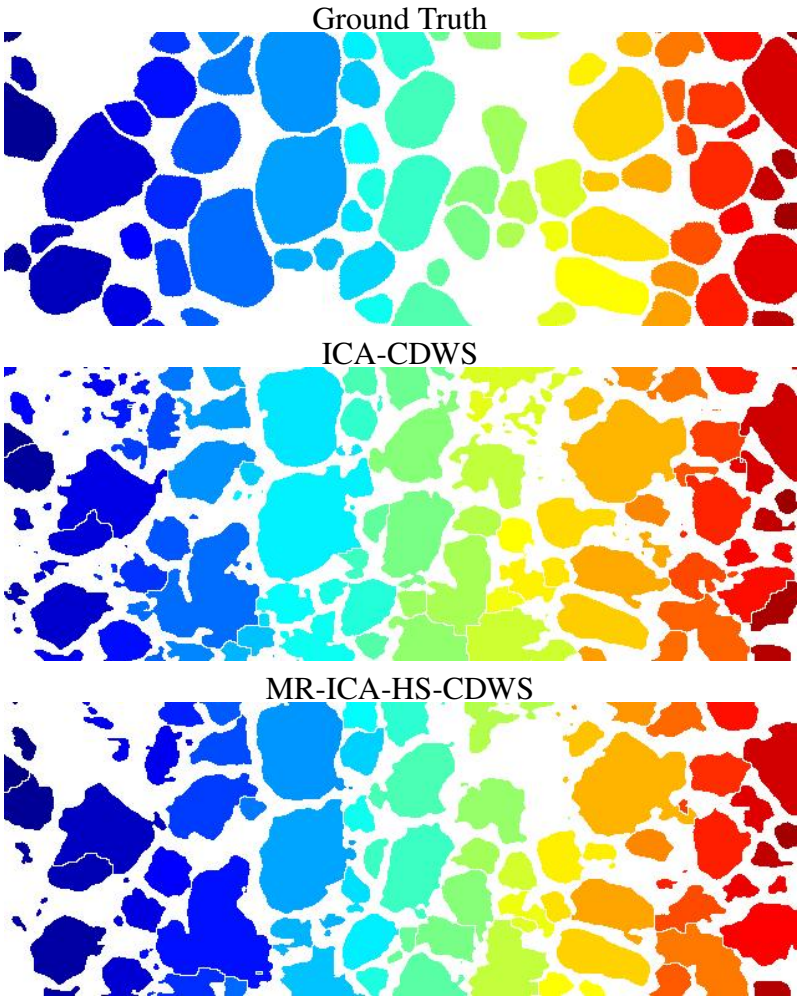



Figure 6.5: Output for $\mathfrak{L}_0$ and $\mathfrak{L}_1$ layers. Notice the significant reduction in noise as well as the improvement in object-object boundary delineation.

Table 6.7: Forestry Domain: Performance comparison of base classification ($\mathfrak{L}_0$) to heterogeneous stacking ($\mathfrak{L}_1$). For each experimental condition the tables represent leave-one-out cross validation results. In this case the $\mathfrak{L}_0$ features were the same as in the original CDWS experiment #3. The associated 2-sided paired t-test produced probabilities of $0.021, 0.060, 0.567, 0.0006, 0.0014$ for I/U, Accuracy, Precision, Recall and Label scores being statistically insignificant. Clearly for this domain, heterogeneous staking is mainly improving the recall score while maintaining precision at a relative constant.

**Hand Crafted Features (HC)**

**HC-CDWS**

| Image | jacq | acc | prec | recall | label score |
|---|---|---|---|---|---|
| 1 | 0.72 | 0.97 | 0.89 | 0.78 | 0.69 |
| 2 | 0.70 | 0.91 | 0.96 | 0.72 | 0.47 |
| 3 | 0.54 | 0.92 | 0.96 | 0.55 | 0.27 |
| 4 | 0.69 | 0.97 | 0.91 | 0.74 | 0.68 |
| 5 | 0.80 | 0.91 | 0.92 | 0.86 | 0.47 |
| 6 | 0.73 | 0.95 | 0.94 | 0.76 | 0.54 |
| 7 | 0.66 | 0.89 | 0.74 | 0.86 | 0.58 |
| 8 | 0.51 | 0.85 | 0.53 | 0.93 | 0.39 |
| 9 | 0.82 | 0.90 | 0.88 | 0.92 | 0.55 |
| 10 | 0.74 | 0.89 | 0.77 | 0.94 | 0.56 |
| 11 | 0.80 | 0.96 | 0.92 | 0.86 | 0.52 |
| 12 | 0.65 | 0.93 | 0.72 | 0.88 | 0.42 |
| 13 | 0.47 | 0.83 | 0.49 | 0.91 | 0.42 |
| 14 | 0.57 | 0.90 | 0.62 | 0.88 | 0.43 |
| 15 | 0.80 | 0.89 | 0.87 | 0.91 | 0.26 |
| 16 | 0.82 | 0.92 | 0.92 | 0.89 | 0.51 |
| 17 | 0.81 | 0.94 | 0.91 | 0.89 | 0.32 |
| 18 | 0.85 | 0.96 | 0.95 | 0.89 | 0.41 |
| **mean** | **0.70** | **0.92** | **0.83** | **0.84** | **0.47** |
| stdev | 0.12 | 0.04 | 0.15 | 0.10 | 0.12 |

**MR-HC-HS-CDWS**

| Image | jacq | acc | prec | recall | label score |
|---|---|---|---|---|---|
| 1 | 0.77 | 0.97 | 0.91 | 0.83 | 0.76 |
| 2 | 0.74 | 0.92 | 0.97 | 0.76 | 0.55 |
| 3 | 0.59 | 0.93 | 0.97 | 0.60 | 0.35 |
| 4 | 0.66 | 0.96 | 0.82 | 0.77 | 0.67 |
| 5 | 0.81 | 0.92 | 0.91 | 0.88 | 0.54 |
| 6 | 0.74 | 0.95 | 0.94 | 0.77 | 0.58 |
| 7 | 0.67 | 0.89 | 0.74 | 0.87 | 0.60 |
| 8 | 0.49 | 0.84 | 0.50 | 0.94 | 0.38 |
| 9 | 0.83 | 0.91 | 0.88 | 0.93 | 0.49 |
| 10 | 0.71 | 0.87 | 0.74 | 0.95 | 0.58 |
| 11 | 0.81 | 0.96 | 0.92 | 0.87 | 0.55 |
| 12 | 0.65 | 0.93 | 0.71 | 0.88 | 0.42 |
| 13 | 0.50 | 0.85 | 0.53 | 0.91 | 0.46 |
| 14 | 0.66 | 0.93 | 0.71 | 0.90 | 0.52 |
| 15 | 0.82 | 0.91 | 0.89 | 0.92 | 0.35 |
| 16 | 0.84 | 0.93 | 0.95 | 0.89 | 0.53 |
| 17 | 0.84 | 0.95 | 0.94 | 0.90 | 0.44 |
| 18 | 0.87 | 0.97 | 0.96 | 0.90 | 0.53 |
| **mean** | **0.72** | **0.92** | **0.83** | **0.86** | **0.52** |
| stdev | 0.12 | 0.04 | 0.15 | 0.09 | 0.11 |

## 6.3 Output Decomposition Mixture of Experts

To test the Output Decomposition Mixture of Experts approach outlined in previous chapter, the same nine, $236 \times 637$ pixel images containing mineral ore were used (see Figures 5.1 and 6.6 for examples). All algorithms, described below, were trained on image 1 and evaluated[3] on images 2-9. Feature extraction, for both the input images and output labels, was performed using a bank of log-Gabor filters as in [63], with 7 scales and 6 orientations for a total of 42 filters. Since the inverse Fourier transform of filter responses contains both real and imaginary parts, we further separated the feature maps into two components corresponding to the even (real) and odd (imaginary) filter responses. As a result, both inputs and outputs were decomposed into 84 feature maps. Each of the 84 experts was focused on a single output feature map and was trained using either: (a) linear regression, or (b) regression trees [45]. The gating function was designed as in the previous section. To test the efficacy of frequency domain filtering we run our system with $\boldsymbol{\alpha} \in \{0, 1\}$, corresponding respectively to **Raw** (i.e., non-filtered) and **Filtered** outputs in Equation 5.6. For further comparison we also implemented the **standard** contextual pixel labeling approach, defined by Equation 5.2, that used the same input features to directly output labels (depicted by a dashed arrow in Figure 5.1). In addition, we also tested several typical ensemble methods based on bagging. The first version was based on the original bootstrap version of bagging from [15] whereby 60% of the training samples were randomly selected for training each ensemble member. The second version was loosely based on [16], whereby we randomly permuted 10% of the labels within the training set for each member of the ensemble. Each version was tested with 40 and 80 ensemble members. **The metrics used for evaluation are presented in Appendix C on p. 138**.

### 6.3.1 Results

Experimental results are presented in Table 6.8 with examples of test output presented in Figure 6.6. All algorithms performed comparably, in terms of pixel based measures. However, significant differences exist in terms of the label score, which was designed specifically to evaluate object level information, namely the number of objects, their location and boundaries. From this perspective, the proposed OD-MoE algorithm is far more suitable to the object delineation task(s) as indicated by a label score, of 0.43 for OD-MoE(Filtered) using regression trees, which is almost three times better than the competition. As mentioned throughout this thesis, the identification of object parts rather than simple pixel based labels, lies at the heart of the output decomposition function. Each log-Gabor filter identifies various frequency components comprising the **target** objects. In turn, these components are

[3] Similar results to those presented in this paper were obtained using different train test splits.

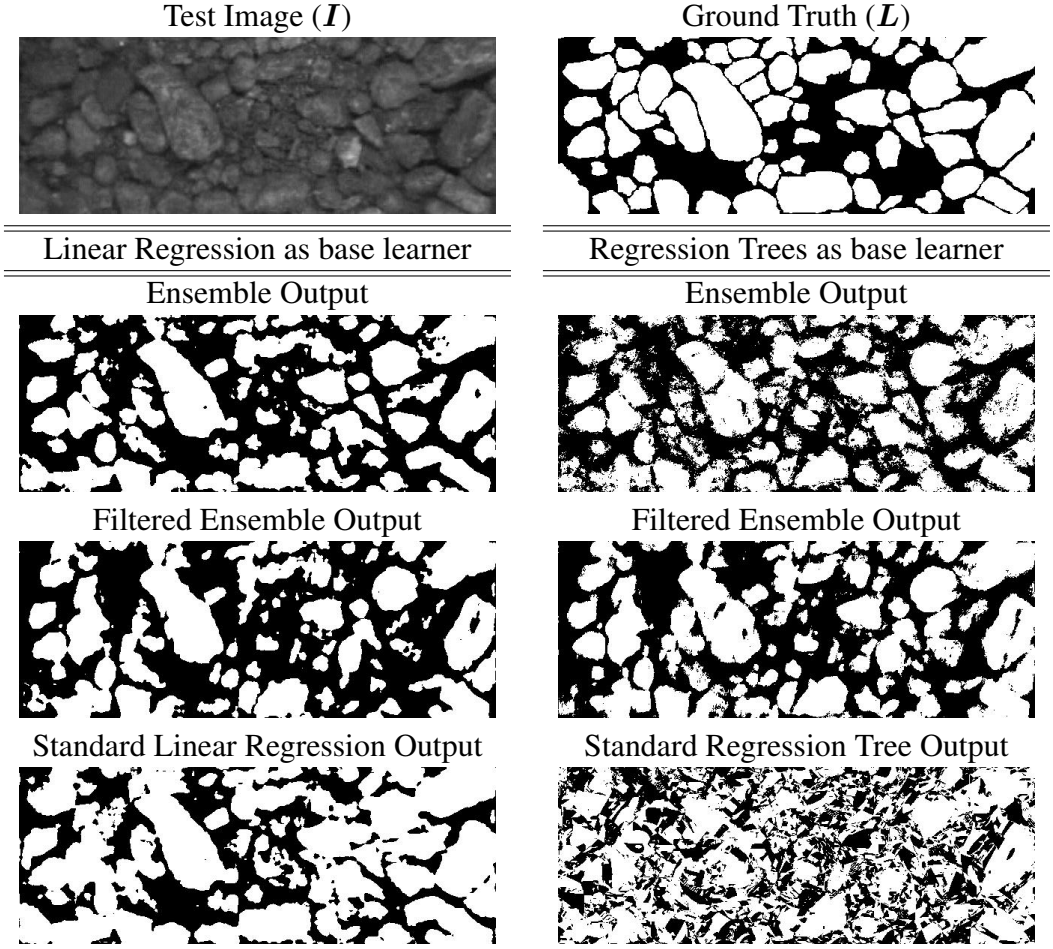



Figure 6.6: **Top Row:** Test input image and corresponding ground truth. Test output using Linear Regression as base learner (**Left**) and Regression Tree as base learner (**Right**).

easier to learn than the unstructured pixel labels. It is thus no surprise that the algorithm improves the labeling at the object level rather than the pixel level. The difference in segmentation quality can be readily observed in Figure 6.6. With respect to bagging, despite an increase in performance at the pixel level when compared to using a single regression tree, the label score clearly remains unaffected. Visually the output looks very similar to the output of a single regression tree[4] which provides a stark contrast to the output of OD-MoE.

Examining the merit of frequency domain filtering, we can see mixed results. When linear regression is used to construct the individual experts, the filtering step

[4]The output from Bag(40) and Bag(80) looked very similar to the output of standard regression tree, while the output from RLP_Bag(40) and RLP_Bag(80) looked very similar to the output of standard linear regression.

Table 6.8: Average performance on test images. **Standard** denotes regression performed directly on the ground truth as is commonly done for pixel labeling. **Raw** denotes OD-MoE without filtering. **Filtered** denotes OD-MoE with filtering prior to output reconstruction. **Bag** denotes a standard bagging procedure with each member of the ensemble using randomly selected $60\%$ of the training samples. **RPL_Bag** denotes bagging where $10\%$ of the labels were randomly permuted for each ensemble member. Ensemble sizes are shown in brackets.

Linear Regression as base learner

| Algorithm | jacq | acc | prec | recall | label score |
|---|---|---|---|---|---|
| Standard | **0.67** | 0.75 | 0.78 | 0.84 | 0.14 |
| OD-MoE(Raw) | 0.64 | **0.76** | 0.86 | 0.72 | 0.40 |
| OD-MoE(Filtered) | 0.59 | 0.73 | **0.87** | 0.64 | 0.38 |

Regression Tree as base learner

| Algorithm | jacq | acc | prec | recall | label score |
|---|---|---|---|---|---|
| Standard | 0.52 | 0.62 | 0.68 | 0.69 | 0.04 |
| OD-MoE(Raw) | 0.61 | 0.74 | 0.85 | 0.69 | 0.41 |
| OD-MoE(Filtered) | 0.62 | 0.75 | 0.86 | 0.68 | **0.43** |

Bagged Regression Trees

| | jacq | acc | prec | recall | label score |
|---|---|---|---|---|---|
| Bag(40) | 0.63 | 0.70 | 0.71 | **0.86** | 0.04 |
| Bag(80) | 0.63 | 0.71 | 0.71 | 0.85 | 0.04 |
| RLP_Bag(40) | 0.59 | 0.69 | 0.74 | 0.74 | 0.14 |
| RPL_Bag(80) | 0.59 | 0.69 | 0.74 | 0.75 | 0.14 |

is detrimental to performance. However, when regression trees are employed at the base level, overall performance significantly improves. Our results consistently demonstrate that regardless of the base regressor used, precision scores improve as a result of the filtering step. In contrast the experimental results in the previous section indicate that Heterogeneous Stacking primarily increases the recall scores. Perhaps future studies can combine these these two promising techniques to improve both precision and recall. Visually, Figure 6.6 shows a significant reduction in noise when frequency domain filtering is used in conjunction with regression trees.

This section presented the Output Decomposition Mixture-of-Experts (OD-MoE) algorithm designed for object delineation task. By using an output decomposition function to identify coherent parts of the objects, each function approximator, comprising the mixture of experts, can be focused on a specific object aspect. The experimental results illustrate the utility of mapping to primitive object structures, which appears to be an easier task than attempting to identify individual pixel labels. This enables OD-MoE to produce results superior to those of standard pixel labeling algorithms. More specifically, OD-MoE is able to separate individual objects, while the basic pixel labeling algorithms, devoid of higher level knowledge of objects, consistently fused several objects together.

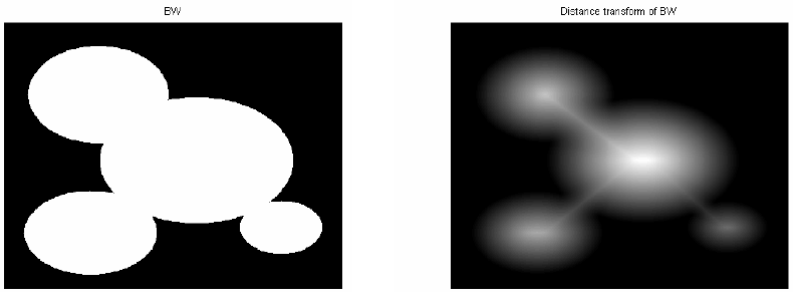



Figure 6.7: Distance Transform of a binary image.

## 6.4 Stacked Convolutional Regression (SCR)

In order to evaluate the efficacy of stacked convolutional regression networks within the context of data driven region growing framework, the same nine images of ore fragments were used. To allow comparison to previous experimental results with CDWS and Heterogenous Stacking systems the same LOOCV approach was utilized as in Section 6.1.2. For all experiments using SCR the target labels $\boldsymbol{L}_{region}$, $\boldsymbol{L}_{boundary}$, $\boldsymbol{L}_{marker}$ were modified using the distance transform and re-scaled to the range $[-1, 1]$. The distance transform, $bwdist(\boldsymbol{L}_{type})$, was applied to all pixel belonging to target objects and measured the Euclidean distance of a given pixel to the object border as depicted in Figure 6.7. The preprocessing is then defined as:

$$\boldsymbol{L}_{type} = \frac{ln(bwdist(\boldsymbol{L}_{type}))}{\max_{(i,j)\in S}\left(ln(bwdist(\boldsymbol{L}_{type}))\right)}$$

where $type \in \{region, boundary, marker\}$. All experiments utilized trained *conventional* neural networks to learn mappings in the form $N_r^{\boldsymbol{I}}(i,j) \mapsto N_r^{\boldsymbol{L}_{type}}(i,j)$ using stochastic gradient descent with mini-batches of 100 samples. The learning parameters, e.g., initial learning rate and the annealing schedule were optimized based on a tenth ground-truthed image not used during the learning and evaluation stages of the following experiments. It should be noted that all cites $(i,j) \in S$ were used in training. When applied to test images, the conventional networks were converted into convolutional networks to avoid utilizing the less efficient sliding window approach.

### 6.4.1 Experiment 1

To test the benefits of producing label patches $N_r(i,j)$ rather than individual pixels, three main network configurations created with $r \in \{0, 5, 10\}$, resulting in output patch sizes of: $1\times1, 11\times11, 21\times21$. For all configurations, the neural network was grown from 1 hidden layer to 7 hidden layers, with all layers containing 100 hidden

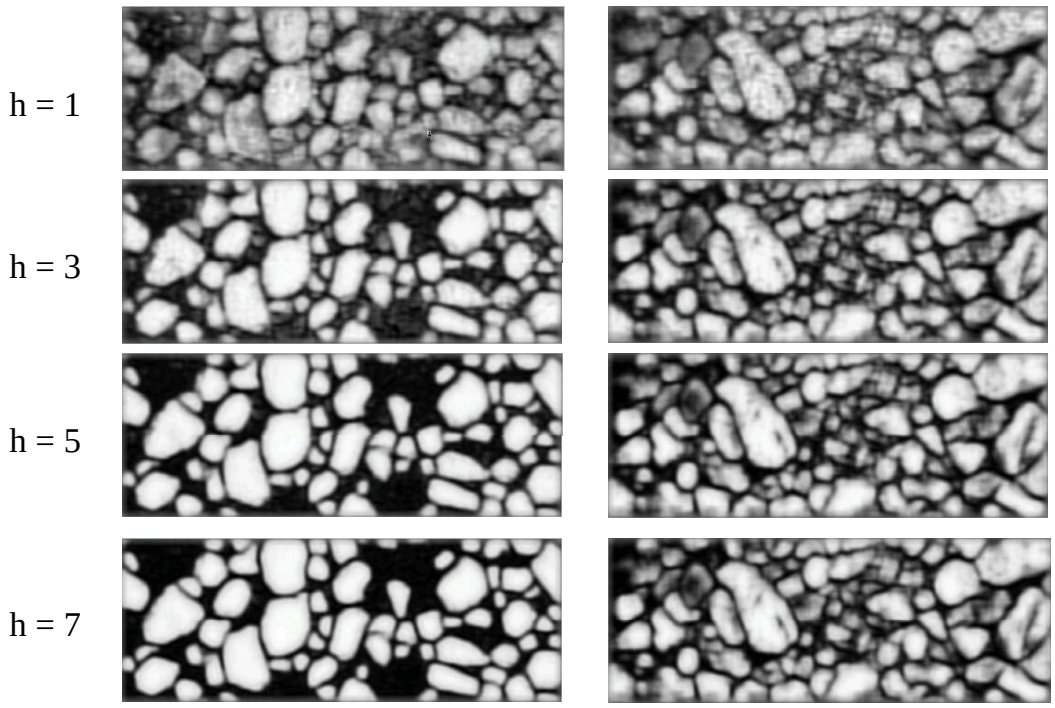



Figure 6.8: Output of the trained stacked convolutional regression neural network. **Left:** Training output as a function of the number of hidden layers, $h$, in a $21 \times 21 \mapsto \{100\}^h \mapsto 21 \times 21$ network. **Right:** Test Image output.

nodes as defined in Equation 5.22. To prevent overfitting, $\ell_1$ and $\ell_2$ regularization was used as in Equations 3.18 and 3.21 with $\lambda = 0.1$ in both cases. The change from non-regularized to regularized networks can be observed in Figure 6.9. Visually, regularization appears to produce sharper filters.

Quantitative results are presented in Tables 6.9 and 6.10. In addition, Figure 6.8 presents the test/train output of a single Stacked Convolutional Network. In addition to testing the different output patch sizes, we examined if narrow networks with only 10 hidden units per layer were as effective as the networks utilizing 100 hidden units per layer. Finally we also looked at the effects of preprocessing the images with histogram equalization [39].

Several points are immediately clear. First and foremost, as conjectured in previous chapters, mapping to output patches rather than individual labels produces significantly better segmentation results as evidenced by the results in Table 6.9. At the pixel level, the single site output labeling networks attained a significantly lower precision. Furthermore regardless of output topology, networks with a larger number of hidden layers produce more accurate object segmentation. Even for networks labeling single cites, $\boldsymbol{L}(i, j)$, deeper networks perform much better. However, the effect of labeling neighborhoods larger than one pixel has a profound effect. Ob-

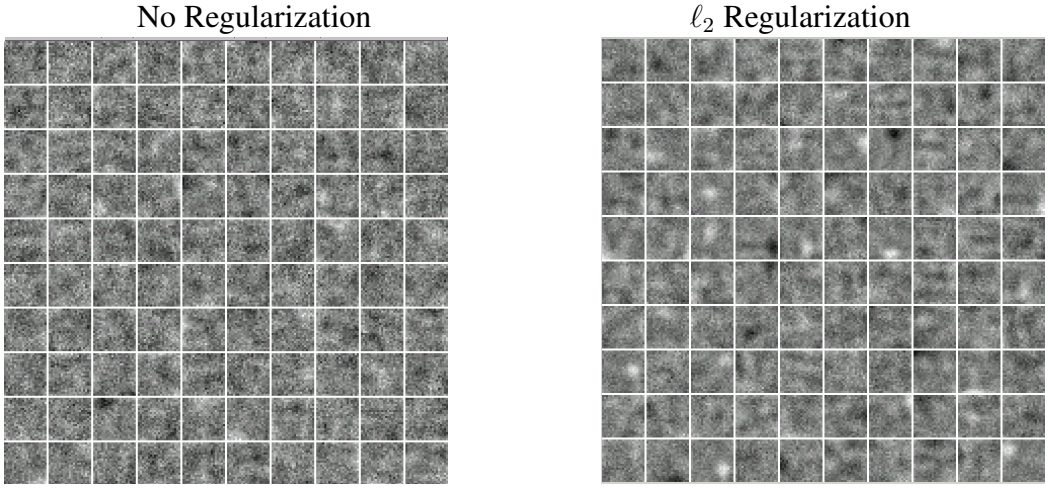



Figure 6.9: Effects of Regularization.

serve that a network with a single hidden layer mapping to $N_5(i,j)$ or $N_{10}(i,j)$ significantly outperforms the system utilizing networks with $N_0(i,j)$ output and seven hidden layers.

In terms of hidden layer size, clearly the narrow networks under-perform. Given $21 \times 21$ input patch, ten components simply do not adequately describe such a large patch. As a result recall appears to be significantly lower for the narrow networks. Comparing effectiveness of $\ell_1$ and $\ell_2$ regularization, there seems to be little difference in terms of performance. In light of minimal performance differences, the pruning capabilities of $\ell_1$ regularization would be preferable to $\ell_2$ regularization. To test the current limits for using patch-based networks, we attempted to train large $31 \times 31 \mapsto \{300\}^7 \mapsto 31 \times 31$ (containing 1,026,600 free parameters) networks using $\ell_1$ regularization. The performance gain was minimal (label score of $0.6587$, which is once again not a surprise given that only $236 \times 637 = 150,332$ training patches were used for training.

Regarding histogram equalization as a preprocessing step we observe mixed results in Table 6.10. For networks with one hidden layer histogram equalization significantly improves performance. On the other hand, these performance gains disappear once the networks are 'grown' to seven hidden layers. In turn, this yields further insights as to the role the additional hidden layers play. In light of the aforementioned results, one is led to believe that the hidden layers learn illumination invariance. Changes in illumination and reflectance however, can be viewed as forms of noise, thereby corroborating the view put forth in previous chapters that the internal pixel-wise layers act as denoising elements.

Table 6.9: Experiments using $\ell_2$ Regularization. These experiments looked at performance with respect to the size of output patches. $N_r^{\boldsymbol{L}}(i,j)$ was varied from $r = \{0, 5, 10\}$ resulting in output patch sizes of: $1 \times 1, 11 \times 11, 21 \times 21$. In all cases, results are presented using $h = \{1, 3, 5, 7\}$ hidden layers, with each layer having a constant 100 nodes.

| 21x21-{100}xh-1x1 | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| **h=1** | 0.6456 | 0.7581 | 0.8463 | 0.7383 | 0.3826 |
| **h=3** | 0.682 | 0.7674 | 0.793 | 0.8341 | 0.4564 |
| **h=5** | 0.6869 | 0.7744 | 0.8044 | 0.8287 | 0.495 |
| **h=7** | 0.6901 | 0.7849 | 0.8325 | 0.8036 | 0.5114 |

| 21x21-{100}xh-11x11 | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| **h=1** | 0.667 | 0.7842 | 0.8962 | 0.7255 | 0.5548 |
| **h=3** | 0.7234 | 0.8214 | 0.9049 | 0.7835 | 0.6301 |
| **h=5** | 0.7222 | 0.8222 | 0.915 | 0.7747 | 0.6475 |
| **h=7** | 0.724 | 0.8231 | 0.9133 | 0.778 | 0.6454 |

| 21x21-{100}xh-21x21 | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| **h=1** | 0.699 | 0.8002 | 0.8737 | 0.7787 | 0.547 |
| **h=3** | 0.7306 | 0.8251 | 0.8998 | 0.7962 | 0.6334 |
| **h=5** | 0.733 | 0.8266 | 0.9001 | 0.7988 | 0.6472 |
| **h=7** | 0.7344 | 0.8276 | 0.9005 | 0.8001 | 0.6487 |

Table 6.10: Experiments with $\ell_1$ Regularization. The experiments examined performance with respect to the size of hidden layer, preprocessing and regularization. The input and output patch sizes were $21 \times 21$ pixels. Results are presented using $h = \{1, 3, 5, 7\}$ hidden layers.

with Histeq and L1=0.1

| 21x21-{100}xh-21x21 | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| **h=1** | 0.7088 | 0.8096 | 0.8904 | 0.7774 | 0.5988 |
| **h=3** | 0.7229 | 0.8226 | 0.9139 | 0.7762 | 0.6386 |
| **h=5** | 0.7243 | 0.8236 | 0.9149 | 0.7771 | 0.6472 |
| **h=7** | 0.7264 | 0.8247 | 0.9134 | 0.7807 | 0.6499 |

L1reg 0.1

| 21x21-{100}xh-21x21 | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| **h=1** | 0.6939 | 0.7965 | 0.8721 | 0.7736 | 0.5587 |
| **h=3** | 0.7208 | 0.8206 | 0.91 | 0.7769 | 0.6364 |
| **h=5** | 0.7227 | 0.8223 | 0.9131 | 0.7766 | 0.6471 |
| **h=7** | 0.7239 | 0.8229 | 0.9123 | 0.7786 | 0.6433 |

L1reg 0.1

| 21x21-{10}xh-21x21 | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| **h=1** | 0.5891 | 0.7359 | 0.8918 | 0.6348 | 0.4385 |
| **h=3** | 0.6625 | 0.7801 | 0.8878 | 0.7237 | 0.5459 |
| **h=5** | 0.6641 | 0.7824 | 0.8943 | 0.7213 | 0.5611 |
| **h=7** | 0.6681 | 0.7849 | 0.895 | 0.7253 | 0.5634 |

### 6.4.2 Experiment 2

To demonstrate the ability of SCR to operate at multiple resolutions and utilize more than two convolutional layers, the second experiment created a set of progressively deeper networks with alternating convolutional-conventional layers. Each convolutional layer utilized three sets of $5 \times 5$ filters, with each set operating at a different resolution. During training we extracted $5 \times 5, 9 \times 9, 17 \times 17$ image patches and resampled the larger patches into $5 \times 5$ patches. The three sets were then concatenated together to form training data for the convolutional layers. This procedure was repeated for each convolutional layer during the training process. In addition, we tested the utility of hybrid training, where the target output was concatenated with the input to create a new output vector used during the training of the network. Based on the discussion in [7] unsupervised and hybrid training performed better than strictly supervised greedy layer-wise training we have adopted. Experimental setup was as in the previous experiment.

The results, presented in Table 6.11, clearly demonstrate the efficacy of multi-resolution, multi-convolutional architecture. Although hybrid training did improve the results, the pair-wise t-test did not detect statistically significant differences. However, both approaches produce statistically significant improvements with respect to the networks trained in the previous experiment. Visually, the segmentation results are presented in Figure 6.10 and indicate a high degree of correspondence between the output of the SCR and ground truth.

Table 6.11: Multi-resolution, Multi-Convolution results. **Top:** Results using Supervised Greedy Layerwise Learning. **Bottom:** Results using Hybrid Greedy Layerwise where the target output is augmented with the input vector. Both training regimes incrementally trained a set of alternating convolutional-conventional hidden layers. The convolutional layers utilized $5 \times 5$ filters analyzing the input at 3 spatial resolutions. All hidden layers contained 20 hidden nodes and utilized $\ell_2$ regularization. Comparing the performance of the best 7 layer network from Table 6.10 to the best network (with 12 layers) in this table using the paired t-test results in the following p-values: 0.0000, 0.000328, 0.439039, 0.000008, 0.013133. Except for recall, the improvement is statistically significant.

| 5x5@3res-{20}xh-5x5 | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| h=2 | 0.4945 | 0.597 | 0.6646 | 0.6637 | 0.3004 |
| h=4 | 0.7431 | 0.8346 | 0.9094 | 0.8033 | 0.6508 |
| h=6 | 0.7542 | 0.8412 | 0.9064 | 0.8188 | 0.6693 |
| h=8 | 0.7523 | 0.8404 | 0.9091 | 0.8143 | 0.6676 |
| h=10 | 0.7532 | 0.8408 | 0.9082 | 0.8161 | 0.6702 |
| h=12 | 0.7521 | 0.8401 | 0.9081 | 0.8149 | 0.6686 |
| h=14 | 0.7524 | 0.8402 | 0.9071 | 0.8162 | 0.6674 |

| 5x5@3res-{20}xh-5x5 | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| h=2 | 0.456 | 0.5463 | 0.6165 | 0.6426 | 0.2182 |
| h=4 | 0.7432 | 0.8331 | 0.8995 | 0.8113 | 0.6427 |
| h=6 | 0.7508 | 0.8394 | 0.9094 | 0.8122 | 0.6658 |
| h=8 | 0.753 | 0.841 | 0.9103 | 0.814 | 0.6676 |
| h=10 | 0.7547 | 0.8421 | 0.9108 | 0.8157 | 0.672 |
| h=12 | 0.7537 | 0.8416 | 0.912 | 0.8136 | 0.6779 |
| h=14 | 0.7535 | 0.8414 | 0.9111 | 0.814 | 0.6773 |

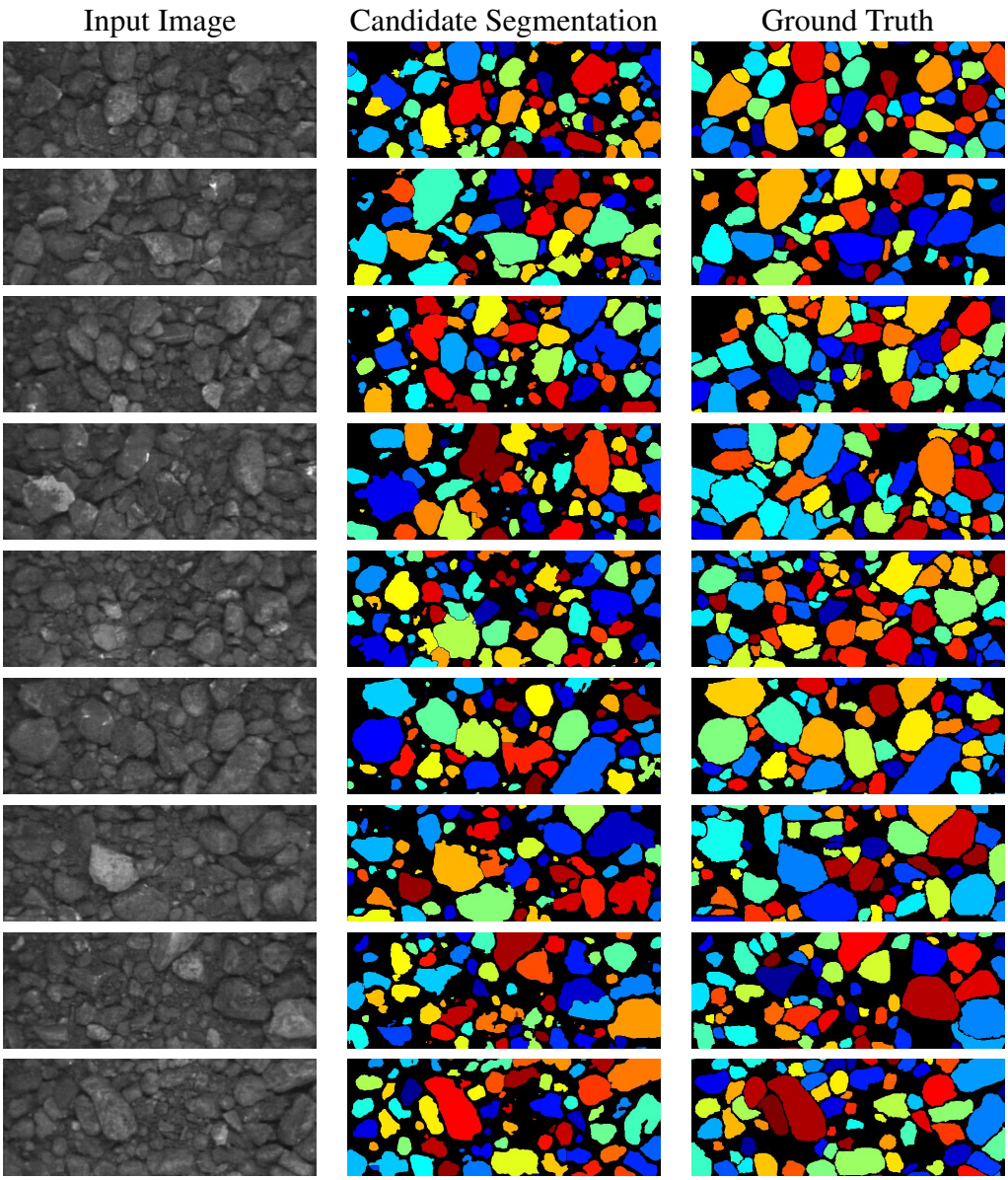



Figure 6.10: Output of the DDRG Framework utilizing a Deep Convolutional Network with 8 conventional and 7 convolutional layers. The first 7 convolutional layers used $5 \times 5$ filters performing analysis at 3 spatial resolutions. All hidden layers had 20 hidden nodes and were trained using the hybrid target function and layer-wise learning. Each connected component in the Ground Truth and Candidate Segmentation has been assigned a unique color for visualization. Corresponding quantitative results are presented at the Bottom of Table 6.11.

Table 6.12: Experiments for evaluating the contributions of individual components within the DDRG framework.

| 21x21-{100}xh-21x21 | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| **DDRG** | 0.7264 | 0.8247 | 0.9134 | 0.7807 | **0.6499** |
| **No Boundaries** | 0.6797 | 0.7975 | 0.9241 | 0.72 | **0.6127** |
| **No Markers and No Boundaries** | 0.7367 | 0.8297 | 0.904 | 0.7999 | **0.5794** |
| **No Watershed** | 0.7365 | 0.8296 | 0.9046 | 0.7991 | **0.579** |

Table 6.13: Comparison of Object Segmentation Systems against DDRG framework utilizing SCR networks. CPL(LogReg)+ DDRG denotes the CDWS system presented at the beginning of the chapter which produced $\boldsymbol{L}_{marker}$, and $\boldsymbol{L}_{region}$.

| System | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| **OSA** | 0.6838 | 0.7825 | 0.8357 | 0.7907 | 0.5522 |
| **WipFrag** | 0.5913 | 0.6493 | 0.6587 | 0.8529 | 0.3574 |
| **CPL(LogReg)+DDRG** | 0.7602 | 0.8385 | 0.8669 | 0.8614 | 0.6189 |
| **7Layer SCR + DDRG** | 0.7264 | 0.8247 | 0.9134 | 0.7807 | 0.6499 |
| **12 layer Multi-Res SCR + DDRG** | 0.7537 | 0.8416 | 0.912 | 0.8136 | 0.6779 |

### 6.4.3 Experiment 3

To ascertain the contribution of each component, $\boldsymbol{L}_{region}$, $\boldsymbol{L}_{boundary}$, $\boldsymbol{L}_{marker}$ on the overall performance of Data Driven Region Growing the system was re-applied to the 9 Oil Ore Images in the now standard LOOCV manner, whereby each component was removed. First we disabled the utilization of $\boldsymbol{L}_{boundary}$, next we removed $\boldsymbol{L}_{region}$ (in addition to removing the $\boldsymbol{L}_{boundary}$ component). Finally we disabled the watershed algorithm, leaving only $\boldsymbol{L}_{region}$. Experimental results are presented in Table 6.12. Clearly each component significantly contributes to the DDRG framework. Interestingly, note that watershed without markers is essentially useless. This in turn implies that the distance transform is not effective at inducing a favorable tolopogy for the watershed algorithm. In contrast, markers and boundary information to indeed provide useful *and* complementary information for improving the region growing process.

The component-by-component breakdown, allows for a fair comparison with the CDWS system, which utilized only $\boldsymbol{L}_{region}$ and $\boldsymbol{L}_{marker}$. In addition, the performance of other systems specific to Oil Ore Segmentation is also compared to DDRG in Table 6.13. While the full DDRG system outperforms all other systems, it attains the high level of performance due to the inclusion of boundary information extracted from $\boldsymbol{L}_{boundary}$. When boundary constraints have been removed, the system performance is comparable to Classification Based Watershed Segmentation

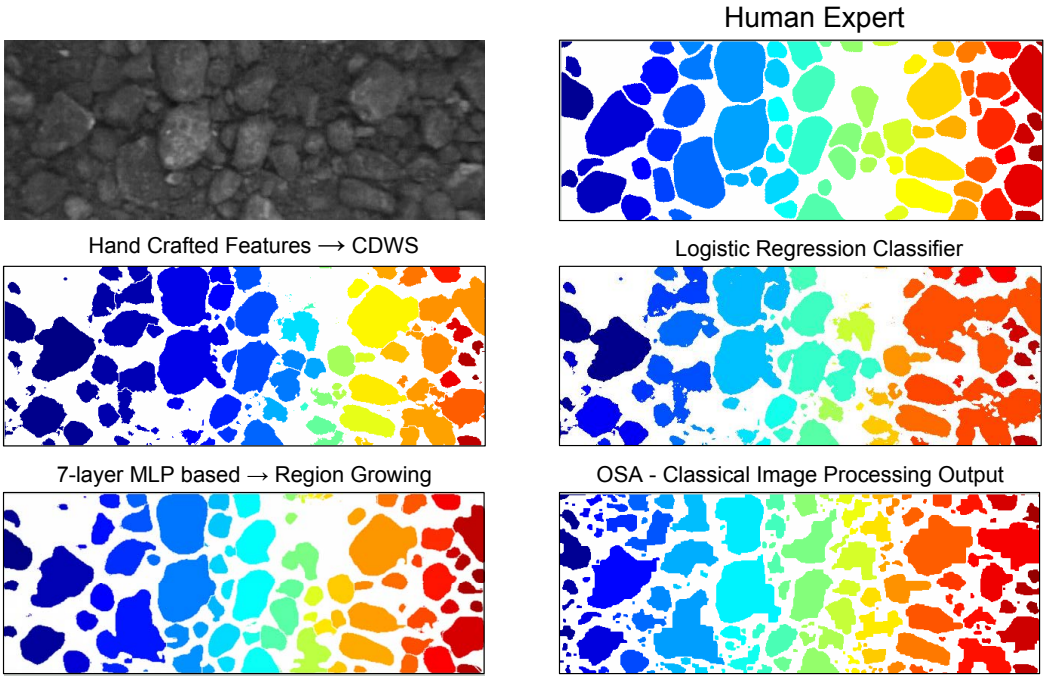



Figure 6.11: Output of DDRG and other algorithms. **Top Left:** Input Image. **Top Right:** Ground Truth **Middle Right:** Output of Logistic Regression utilizing hand crafted features from Figure 6.1. **Middle Left:** Output of CDWS system utilizing Logistic Regression and hand crafted features from Figure 6.1. **Bottom Right:** Output of Ore Segmentation Analyst (OSA) composed of a sequential pipeline of hand-designed image processing operators. **Bottom Left:** Output of DDRG utilizing $21 \times 21 \mapsto \{100\}^7 \mapsto 21 \times 21$ SCR networks.

(CDWS) system. However, the CDWS system utilized a large set of hand-crafted features, while the DDRG system utilized stacked convolutional regression which mapped raw image patches to labels in a fully automated fashion.

An example of object segmentation differences between the various systems is presented in Figure 6.11. In contrast to other systems, Stacked Convolutional Regression Networks produce much smoother object boundaries and, at least visually, closely mimic the ground-truth object boundaries.

Table 6.14: Performance of DDRG utilizing seven layer SCR on datasets $DS1, DS2, DS3$. For comparison, the label score for CDWS is presented.

| DatSet | I/U | Pixel accuracy | Precision | Recall | label score | CDWS label score |
|---|---|---|---|---|---|---|
| **DS1** | 0.6833 | 0.8121 | 0.88 | 0.7534 | 0.6031 | 59.1 |
| **DS2** | 0.7264 | 0.8305 | 0.847 | 0.8371 | 0.6286 | 56.1 |
| **DS3** | 0.736 | 0.8172 | 0.8925 | 0.8089 | 0.6014 | 52.1 |

### 6.4.4 Large Scale Experiments

To further evaluate the DDRG framework and the SCR algorithm the system described in the previous section was run on three large image datasets. Datasets, $DS1$ and $DS2$ each contained 99 images along with the ground truth, while $DS3$ contained 50 labeled images. The image dimensions for each dataset were: $501 \times 161$, $501 \times 201$, and $501 \times 381$ pixels. Example images from each data set are shown in Figure 6.12. Observe the dramatic change in illumination, camera angle and object size for $DS3$. We used the SCR models trained on the nine images described in Section 6.1.2. The models utilized histogram equalization as preprocessing and were trained with $\ell_1$ regularization set to 0.1. The model topology was $21 \times 21 \mapsto \{100\}^7 \mapsto 21 \times 21$. Previous performance of these models, using the LOOCV procedure, was presented in Table 6.10. Model averaging was performed analogous to previous experimental setups, the output of the *nine* models was averaged together to produce the final probability maps, $\boldsymbol{P}_{region}, \boldsymbol{P}_{marker}, \boldsymbol{P}_{boundary}$ and subsequently re-scaled to the range $[0, 1]$. Once again only default thresholds were used, $\tau_1 = \tau_2 = \tau_3 = 0.5$. The results are presented in Table 6.14.

Compared with the LOOCV results (label score of 0.6499) in previous section, performance dropped slightly but is never-the-less still superior to that of CDWS system. In fact performance of all other systems (WipFrag and OSA) also drops (results not shown). Subsequent examination of the datasets revealed a large number of inconsistencies within the ground truth images, which were produced by first running the OSA system and then manually modifying the output to correct gross errors. In contrast, the training set had relatively higher quality of ground truth, which was manually and somewhat meticulously created from scratch. In addition, the aforementioned changes in illumination, camera angle and object size have also negatively impacted the quality of the final object segmentation. Despite these challenges, the DDRG system demonstrated robust performance, with minimal performance degradation.

Image from DS 1

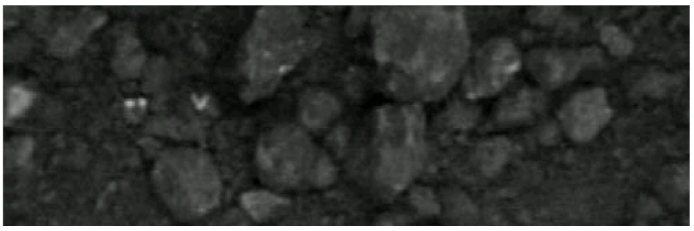


Image from DS 2

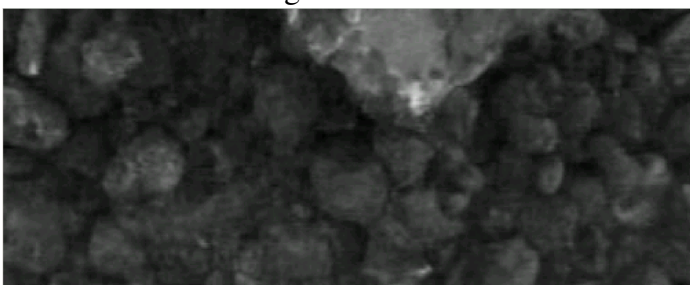


Image from DS 3

Figure 6.12: Example images from the three large data sets.

Table 6.15: Performance of DDRG utilizing SCR on aerial forest images. **Top Row:** Mean **Bottom Row:** Standard Deviation. Compare with performance of CDWS in Table 6.4.

**SCR with 5 hidden layers**

| I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|
| **0.5581** | **0.8731** | **0.6487** | **0.8367** | **0.4833** |
| 0.1738 | 0.0724 | 0.2062 | 0.1417 | 0.1493 |

**SCR with 7 hidden layers**

| I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|
| **0.6115** | **0.9117** | **0.774** | **0.7718** | **0.5616** |
| 0.1558 | 0.0446 | 0.1884 | 0.1511 | 0.1237 |

### 6.4.5 Aerial Forest Images Revisited

To evaluate how suitable the DDRG framework and SCR is for other domains, we once again turned to the aerial forest imagery dataset. The experimental setup used was the same as described in section 6.1.5 with one exception. Due to computational constraints, we down-sampled the images from $512 \times 512$ to $256 \times 256$. In order to properly evaluate the results we up-sampled the output back to its original size. The change in resolutions did not appear to change the performance measures. For each of the 18 HSV images, three SCR networks were trained to produce $\boldsymbol{P}_{region}, \boldsymbol{P}_{marker}, \boldsymbol{P}_{boundary}$. The networks had the following topology - $21 \times 21 \times 3 \mapsto \{100\}^h \mapsto 21 \times 21 \times 3$, with $h \in \{5, 7\}$. Since histogram equalization is not appropriate in HSV space, we used histogram matching instead. For each test image, the histogram of each (H,S,V) channel was transformed to match the histogram of the corresponding histogram of the training image. As in previous experiments model averaging was used within the LOOCV evaluation scheme. Experimental results are presented in Table 6.15 and Figure 6.13. Examining the output of SCR to that of CDWS, in Figure 6.13, reveals that SCR significantly smooths out object boundaries. This is most likely due to running the algorithm in lower data resolution. This is also further evidenced by the lower pixel based statistics in comparison to CDWS (in Table 6.4. Note the CDWS system was trained using features extracted at three resolutions (equivalent to $512 \times 512$, $256 \times 256$, and $128 \times 128$ image sizes). Hence it includes, (at least partially) the data used to train CDWS. On the positive side, we can see a very significant increase in the labeling score in comparison to CDWS. The CDWS system produced a label score of $0.47$, while the SCR networks produced a score of $0.56$. In addition, a very large change in label score can be seen between the networks with 5 hidden layers and 7 hidden layers. Comparing the final output of DDRG using 5 and 7 layer SRC reveals an interesting phenomenon. Output of 7 hidden layer system is very similar to the output of the 5 hidden layer system modified by morphological erosion. Perhaps the smoothed object boundaries are a function of more than just input resolution! The aforementioned observation seems to indicate that the upper hidden layers of the 7-layer $h_{region}$ network learned a function similar to morphological erosion.

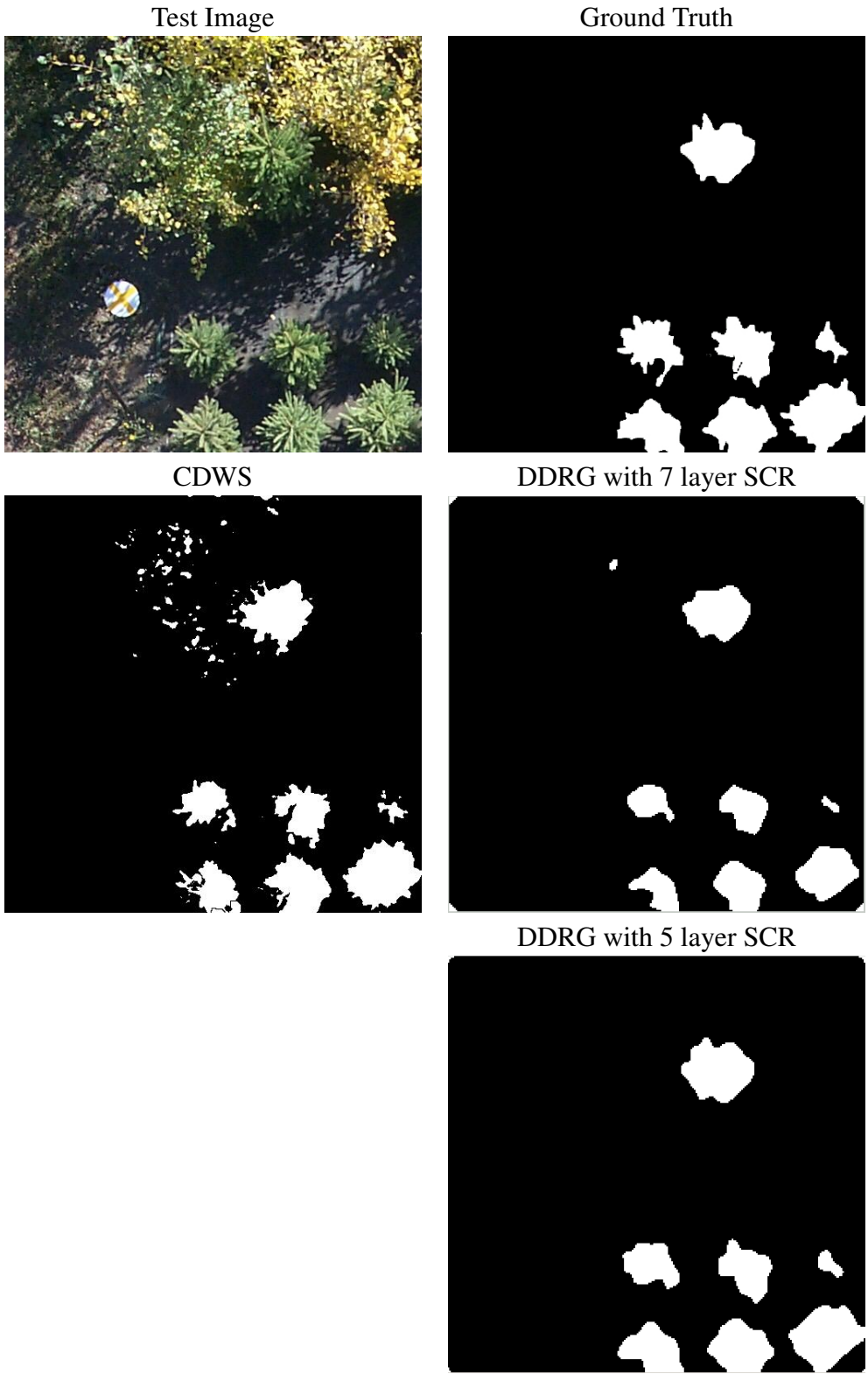



Figure 6.13: Same input image as in Figure 6.4.

Table 6.16: Performance of DDRG utilizing seven layer SCR in conjunction with post processing techniques. Top row show performance using non-default thresholding parameters of: $\tau_1 = 0.45$, $\tau_1 = 0.2$, $\tau_1 = 0.3$. Using this result as the starting point, filtering procedures based on object Size (SF) and Morphological Open (OF) were applied in a manner analogous to those of Table 6.3.

| Post Processing | I/U | Pixel accuracy | Precision | Recall | label score |
|---|---|---|---|---|---|
| tau1=0.5, tau2=0.5, tau3=0.5 | 0.7264 | 0.8247 | 0.9134 | 0.7807 | 0.6499 |
| tau1=0.45, tau2=0.2, tau3=0.3 | 0.7527 | 0.8369 | 0.886 | 0.8343 | 0.6813 |
| SF = 100 | 0.7548 | 0.8392 | 0.8901 | 0.8333 | 0.686 |
| SF = 200 | 0.7531 | 0.8395 | 0.8944 | 0.8274 | 0.6923 |
| SF = 400 | 0.7422 | 0.8397 | 0.8969 | 0.8126 | 0.6952 |
| SF = 800 | 0.7278 | 0.8502 | 0.8914 | 0.7991 | 0.6966 |
| OF = 3 | 0.7519 | 0.8371 | 0.8902 | 0.8297 | 0.6849 |
| OF = 5 | 0.7526 | 0.8388 | 0.8978 | 0.824 | 0.6927 |
| OF = 7 | 0.7388 | 0.8323 | 0.9122 | 0.7961 | 0.6919 |
| OF = 9 | 0.7144 | 0.8194 | 0.926 | 0.7582 | 0.6744 |

### 6.4.6 Post Processing Revisited

Similar to experiments in Table 6.2 we examined performance with respect to several post processing techniques as well as to different threshold settings for $\tau 1, \tau 2, \tau 3$. Results are presented in Table 6.16. The most significant performance boost is produced as a result of changing the threshold parameters. In turn, this result motivates the use of an on-line decision module(s) to adaptively determine threshold parameters for a given image or even for a given connected component. To push the idea further we experimented with filtering out objects based on solidity. Solidity is defined as $\psi = \frac{|G_k|}{|G_k^{'}|}$, where $|G_k|$ is the number of pixels in object $k$ and $|G_k^{'}|$ is the number of pixels in the *convex hull* of object $k$. Thus $\psi = 1$ would imply the object is perfectly convex (see Figure 6.14 for more details). In [87] we discovered that this was one of the most critical shape descriptors, for Ore Fragments. Figure 6.14 presents a histogram of object solidity for ground truth images used in the previous experiments. Clearly most of the objects have a solidity greater that 0.8. Using this insight, we can filter out object that have solidity lower than a predefined threshold $\tau_\psi$. Results of this filtering operation are presented in Figure 6.15 and demonstrate that solidity based filtering can indeed select objects that more closely match the ground truth. One drawback of this basic technique is that a large number of objects are excluded from further analysis. As mentioned previously, rather than discard objects a potential future research direction would

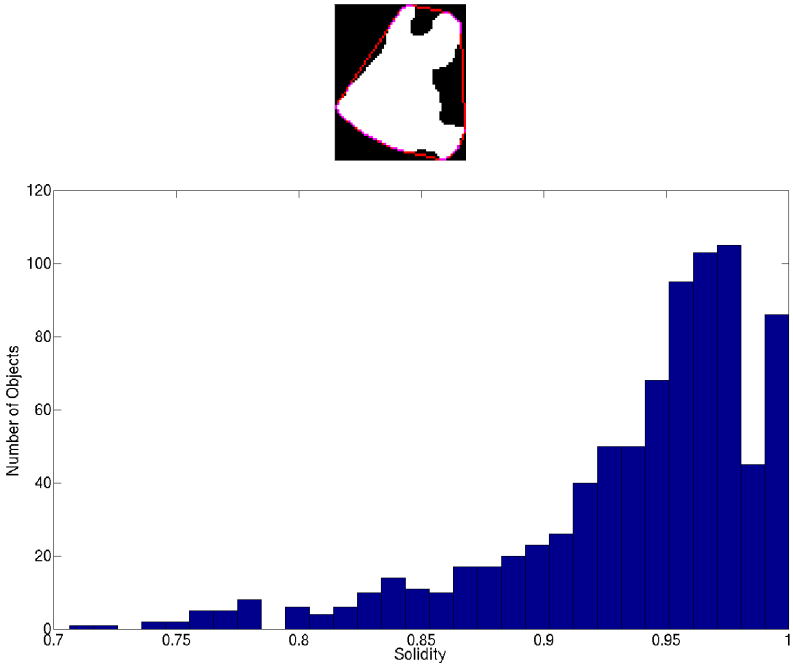



Figure 6.14: Solidity Object Filtering. **Top:** Object (white pixels), and its convex hull (red pixels). Solidity is the proportion of the object pixels to those in the convex hull. In this case solidity is 0.78. **Bottom:** Histogram of solidity for objects within the Ground Truth.

be to create an adaptive module that can maximize solidity by adaptively choosing a threshold value for a given connected component.

## 6.5 Discussion

Previous sections presented extensive experimental results evaluating (a) Data Driven Region Growing (DDRG) and its predecessor CDWS, (b) Heterogeneous Stacking and ICA, (c) Output Decomposition Mixture of Experts, and (d) Stacked Convolutional Regression (SCR).

We have demonstrated that each component of the DDRG framework significantly contributes to the overall performance of the system. Evaluation of the Output Decomposition Mixture of Experts (OD-MoE) system demonstrated that extracting features from both input and output can improve performance at the object level. Since both the DDRG framework and Output Decomposition, put additional pressure on the domain experts by requiring additional feature extraction routines to be coded we turned our attention to automated feature extraction methods. To further automate the system, we examined the performance of ICA to SCR. Exper-

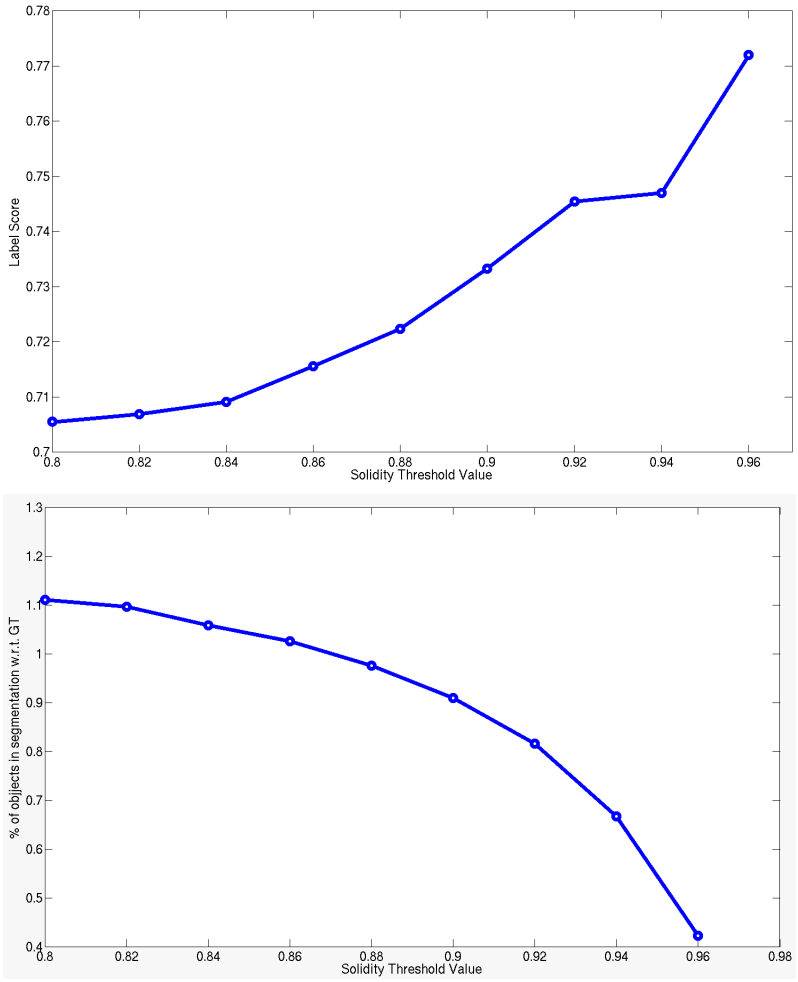



Figure 6.15: Solidity Object Filtering. **Top:** Label Score as a function of Solidity. Objects are filtered if their solidity is less then threshold (x-axis). The remaining objects are scored against ground truth objects that they overlap. **Bottom:** Proportion of objects within DDRG segmentation with solidity grater than threshold value (x-axis).

imental results indicate that ICA, even at three resolutions, performs considerably worse than the proposed SCR algorithm. Furthermore, the SCR algorithms can be viewed as an extension of the (OD-MoE) system by simultaneously extracting features from input and output. Empirical evidence further corroborated the improved performance as a result of analyzing output patches. In addition, we demonstrated that superior performance can be achieved by 'growing' deep networks. Used in conjunction with input/output patch analysis, state-of-the-art results were achieved for the domain of oil ore fragments and aerial forestry.

# Chapter 7

# Conclusion

## 7.1 Summary

This dissertation proposed the Data Driven Region Growing Framework (DDRG) as a solution to the object segmentation problem. In order to separate target objects from the background and each other, the DDRG framework utilizes three machine learned mappings as input into the watershed algorithm. Respectively, the mappings identify foreground-background regions, object markers, and object contours, that are combined together into a topology used by the watershed algorithm to produce the final output. In order to produce the three mappings, $h_{region}$, $h_{marker}$, and $h_{boundary}$, relevant image feature are needed.

On a different line of research, the concept of output decomposition proposed explicitly extracting features from the ground truth and learning a mixture-of-specialists that map to individual output features. The output features are subsequently fused to produce the final output. The argument, corroborated by experimental results and not unlike the philosophy underlying random field methods, is that the analysis of label interactions can lead to improved object segmentation.

The use of output decomposition within the data driven region growing framework would potentially require six unique feature extraction functions to be defined, two for each of the three mappings, $h_{region},\ h_{marker},\ h_{boundary}$. This further exacerbates the problems with manually defining feature extraction functions and motivates the use of automated feature extraction methods. As a result, we focused our attention on neural network based feature extraction techniques which encompass PCA, ICA, Autoencoders and their numerous variants. When viewed as patch based neural networks, it can be shown that all the aforementioned techniques correspond to a sub type of convolutional neural networks that can simultaneously extract relevant features and regress on them. In fact, Output Decomposition Mixture of Experts itself can be viewed as a convolutional neural network where the filters are manually selected rather than learned from data.

The approach, adopted by this research, is to learn a specific set of features using automated feature extraction techniques. Neural networks in general, and convolutional neural networks specifically, are able to simultaneously create features and use them for producing the desired output. Motivated by promising results of Heterogeneous Stacking, Output Decomposition Mixtures of Experts and greedy layer-wise learning research, the Stacked Convolutional Regression (SCR) algorithm was proposed. This approach aims to directly map image patches, $N_r^{\boldsymbol{I}}(i,j)$ to label patches $N_r^{\boldsymbol{L}}(i,j)$ and utilizes incremental layer-wise training strategy, which enables the creation of progressively deeper networks, that in turn are able to extract progressively more useful features for the task at hand. Experimental results indicate that state-of-the-art performance can be achieved by employing Stacked Convolutional Regression Networks to produce the probability maps utilized within the Data Driven Region Growing Framework. Furthermore, due to the automated

feature extraction ability of SCR the mappings can be produced in a highly automated manner, with very little input need from the user. In turn, the high degree of automation enables the proposed system to be easily ported from domain to domain as demonstrated by re-training the system for the domain of forestry, where state-of-the-art results have also been demonstrated.

## 7.2 Contributions

This dissertation presented several contributions, namely: (i) Data Driven Region Growing, (ii) Heterogeneous Stacking, (iii) Output Decomposition Mixture of Experts, and (iv) Stacked Convolutional Regression, all aimed at improving object segmentation. Empirical evidence presented in the previous chapter suggests that each of the algorithms has the potential of improving the object segmentation output. The unifying theme of this research is the investigation of output decomposition and manipulation techniques. To the best of our knowledge, no one has explicitly focused on extracting features from ground truth images. Hence, the presented research direction(s) are unique.

## 7.3 Related and Future Research Directions

Over the past two decades, machine learning has slowly but surely proliferated into image processing research and many other areas of science. However, the research into improving region growing methods has largely ignored this invaluable source of ideas and paradigms. The basic idea of using the output of a machine learned module as input into a region growing algorithm has received relatively little attention. Within the levelsets research, Cobzas et al. [19] have recently used the output of logistic regression as input to a level set function for improving the segmentation of tumors within 3D MRI volumes. In [21] the researchers implemented a semi-automated system that used user defined seeds (a.k.a. markers) to initialize a probabilistic level set method that employed an on-line trained naive bays classifier to form the input to the level set function. The automation of marker construction has received even less attention, save for the previously mentioned work of Lezoray and Cardot in [77, 78]. To the best of our knowledge, there has not been any work on incorporating machine learned boundary constraints into the region growing methods.

Although the proposal to explicitly extract features from the ground truth images is unique, there are several algorithms that can be viewed as being similar. Markov Networks [34], (Hierarchical) Markov Random Fields [18], and Conditional Random Fields [65] to a limited extent attempt to address the problem of output correlations. In [34] researchers used PCA to decompose the input (low

resolution image patches) and output (high resolution patches) and used a random field to map from one to the other. Similar idea was reimplemented using convolutional networks in [60]. Hierarchical Markov Random Fields [18] create a multi-resolution pyramid of the ground truth during the learning phase and propagate information from layer to layer (both up and down the resolution hierarchy) to attain a consistent pixel labeling at all resolutions. Conditional Random Fields [65] explicitly calculate the probability of a given output configuration given input features and attempt to maximize the configuration likelihood over the whole image. In addition, joint kernel maps [124] (also called kernel dependency estimation [123]) attempt to link hand crafted input kernels to hand crafted output kernels using SVM's. To some extent their work is related to the Output Decomposition Mixture of Experts approach. In [89] an energy based model [71] was used in a manner analogous to heterogeneous stacking and was shown to be effective at 'cleaning up' label noise.

### 7.3.1 Convolutional Networks

Clearly, the results from Chapters 4 and 5 indicate a profound connection between patch-based conventional networks and convolutional networks. The fact that conventional networks mapping:

$$N_r^{\boldsymbol{I}}(i,j) \mapsto N_r^{\boldsymbol{L}}(i,j) \tag{7.1}$$

can be converted into convolutional networks presents a number of new research questions. For example:

- Can deep convolutional networks be pre-trained by first training a conventional ANN network (on a subset of the image patches for instance) as in Equation 7.1, then after converting the ANN to a CNN with two convolutional layers further training or even growing the network?

- Is it possible to grow filters? Suppose after creating an ANN as in Equation 7.1, the output patch is expanded to $N_{r+r'}^{\boldsymbol{L}}(i,j)$ and the inner portion of the filter is seeded with previously learned parameters.

- The last chapter remarked that Output Decomposition using multi-resolution Gabor filters at multiple resolutions can be implemented as Convolutional Network operating at multiple resolutions using a fixed set of (Gabor) filters. Similar to the previous point, one could envision replacing the pre-training procedure with seeding the initial weights with 'hand-selected' filters as a starting point, and then letting backpropagation modify them. In fact there

is no reason to think that applying autoencoders to first learn input representation and separately an output representation cannot produce good starting points for learning a discriminative convolutional neural network.

### 7.3.2 Adaptive Processing

Experimental results on adaptive processing suggest the use of adaptive on-line decision making. In turn, previous research on Adaptive Object Recognition (ADORE) [26, 74, 72] has demonstrated some success in this area. Unfortunately, progress has essentially stalled due to the need for automated feature extraction algorithms. To give a concrete example, the previously mentioned Solidity based thresholding can be used to implement an adaptive system that iteratively searches over thresholds $\tau$ to find one that maximizes solidity. Thus solidity is an example of a feature describing object properties, which is needed for adaptive decision making. Recently, in [66], a deep neural network was trained to recognize convex vs non-convex shapes. This indicates that the defining shape characteristics of objects may be automatically learned. In light of these new developments and the experimental results presented in the previous chapter demonstrating the efficacy of automated feature extraction methods, perhaps it is time to revive research focusing on further automating ADORE and other online decision making systems.

## 7.4 Final Thoughts

Automated object segmentation is a difficult task. This dissertation took a step towards solving this problem by employing machine learning methods in conjunction with region growing in order to separate and delineate objects of interest from the background and each other. More specifically, we presented algorithms that modify and decompose the ground truth in order to take advantage on the structural regularities embedded within it. Overall the algorithms presented in this research further automate the system creation process and have the potential to one day produce fully automated object segmentation systems.

# Bibliography


[1] web.cecs.pdx.edu/ mperkows/CAPSTONES/2005/L005.Neura_Networks.ppt, 2005.

[2] R. Adams and L. Bischof. Seeded region growing. *PAMI*, 16(6):641–647, 1994.

[3] W. Au and B. Roberts. Adaptive configuration and control in an ATR system. In *Proceedings of the DARPA Image Understanding Workshop*, pages 667–676, Palm Springs, CA, 1996.

[4] Shumeet Baluja and S. Fahlman. Reducing network depth in the cascade-correlation learning architecture. Technical Report CMU-CS-94-209, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, October 1994.

[5] D. Barash and D. Comaniciu. A common framework for nonlinear diffusion, adaptive smoothing, bilateral filtering and mean shift. 22(1):73–81, January 2004.

[6] Yoshua Bengio. On the challenge of learning complex functions. In Paul Cisek, John Kalaska, and Trevor Drew, editors, *Computational Neuroscience: Theoretical Insights into Brain Function*, Progress in Brain Research. Elsevier, 2007.

[7] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks.

[8] J. Besag. Spatial interaction and the statistical analysis of lattice systems. B-36(2):192–236, 1974.

[9] J. Besag. On the statistical analysis of dirty pictures. B-48(3):259–302, 1986.

[10] S. Beucher and F. Meyer. The morphological approach to segmentation: the watershed transformation. In E. Dougherty, editor, *Mathematical Morphology in Image Processing*. Marcel Dekker, New York, 1992.

[11] J. Bins and B. Draper. Feature selection from huge feature sets. In *Proceedings of International Conference on Computer Vision*, volume 2, pages 159–165, 2001.

[12] A. Bleau and L. J. Leon. Watershed-based segmentation and region merging. *CVIU*, 77(3):317–370, March 2000.

[13] Jeremy S. De Bonet and Paul A. Viola. A non-parametric multi-scale statistical model for natural images. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[14] Charles A. Bouman and Michael Shapiro. A multiscale random field model for bayesian image segmentation. *IEEE Transactions on Image Processing*, 3:162–177, 1994.

[15] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[16] Leo Breiman. Randomizing outputs to increase prediction accuracy. *Machine Learning*, 40(3):229–242, 2000.

[17] H. Cheng and C. A. Bouman. Multiscale bayesian segmentation using a trainable context model. *IEEE Transactions on Image Processing*, 10(4):511–525, 2001.

[18] Li Cheng, Terry Caelli, and Victor Ochoa. A trainable hierarchical hidden markov tree model for color image segmentation and labeling. In *ICPR 2002*, Quebec city, Canada, 2002.

[19] D. Cobzas, N. Birkbeck, M. Schmidt, M. Jagersand, and A. Murtha. A 3d variational brain tumor segmentation using a high dimensional feature set. In *Mathematical Methods in Biomedical Image Analysis (MMBIA 2007) n conjunction with ICCV*, 2007.

[20] F. S. Cohen, Z. Fan, and M. A. Patel. Classification of rotated and scaled textured images using gaussian markov random field models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(2):192–202, 1991.

[21] D. Cremers, O. Fluck, M. Rousson, and S. Aharon. A probabilistic level set formulation for interactive organ segmentation. In *Proc. of the SPIE Medical Imaging*, San Diego, USA, February 2007.

[22] D.S. Culvenor. Tida: an algorithm for the delineation of tree crowns in high spatial resolution remotely sensed imagery. *Computers & Geosciences*, 28(1):33–44, 2002.

[23] F. Dornaika and H. Zhang. Granulometry using mathematical morphology and motion. In *Proceedings of IAPR Workshop on Machine Vision Applications*, pages 51–54, Tokyo, Japan, November 2000.

[24] B. Draper. Modelling object recognition as a markov decision process. *International Conference on Pattern Recognition*, pages D95–99, 1996.

[25] B. Draper, U. Ahlrichs, and D. Paulus. Adapting object recognition across domains: A demonstration. In *Proceedings of International Conference on Vision Systems*, pages 256–267, Vancouver, B.C., 2001.

[26] B. Draper, J. Bins, and K. Baek. ADORE: adaptive object recognition. *Videre*, 1(4):86–99, 2000.

[27] B. Draper, R. Collins, J. Brolio, A. Hanson, and E. Riseman. Issues in the development of a blackboard-based schema system for image understanding. In R. Englemore and T. Morgan, editors, *Blackboard Systems*, pages 189–218. Addison-Wesley, London, 1988.

[28] B. Draper, R. Collins, J. Brolio, A. Hanson, and E. Riseman. The schema system. *International Journal of Computer Vision*, 2:209–250, 1989.

[29] R.P.W. Duin, P. Juszczak, P. Paclik, E. Pekalska, D. de Ridder, and D.M.J. Tax. PRTools4, A Matlab Toolbox for Pattern Recognition. Delft University of Technology, 2004.

[30] M. Egmont-Petersen, D. de Ridder, and H. Handels. Image processing with neural networks- a review. *Pattern Recognition*, 35:2–8, 2002.

[31] J. Fan, G. Zeng, M. Body, and M.S. Hacid. Seeded region growing: an extensive and comparative study. *PRL*, 26(8):1139–1156, 2005.

[32] Beat Fasel. Multiscale facial expression recognition using convolutional neural networks. In *ICVGIP*, 2002.

[33] C. Fox and G. Nicholls. Exact map states and expectations from perfect sampling: Greig, porteous and seheult revisited. In *Twentieth International Workshop on Baysian Inference and Maximum Entropy Methods in Science and Engineering*, 2000.

[34] William T. Freeman, Egon C. Pasztor, and Owen T. Carmichael. Learning low-level vision. *International Journal of Computer Vision*, 40(1):25–47, 2000.

[35] Y. Freund and R. Schapire. A decision-theoretical generalization of on-line learning and an application to boosting. *Computer System Science*, 55:119–139, 1997.

[36] K. Fukushima, S. Miyake, and T. Ito. Neocognitron: A neural model for a mechanism of visual pattern recognition. *T-SMC*, 13:826–834, 1983.

[37] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, January 1979.

[38] Karen Glocer, Damian Eads, and James Theiler. Online feature selection for pixel classification. In *ICML*, pages 249–256, 2005.

[39] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice Hall, 2 edition, 2002.

[40] F.A. Gougeon and D.G. Leckie. Forest information extraction from high spatial resolution images using an individual tree crown approach. Technical report, Pacific Forestry Centre, 2003.

[41] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning*, 3:1157–1182, 2003.

[42] A.R. Hanson and E.M. Riseman. Visions: A computer system for interpreting scenes. In *CVS78*, pages 303–333, 1978.

[43] R. M. Haralick. Statistical and Structural Approaches to Texture. *Proceedings of the IEEE*, 67:786–804, 1979.

[44] R. M. Haralick, K. Shanmugam, and I. Dinstein. Textural Features for Image Classification. *IEEE Transactions on Systems, Man, and Cybernetics*, 3(6):610–621, November 1973.

[45] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer Verlag, New York, 2001.

[46] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillian College Pub. Co., 1994.

[47] Michael T. Heath. *Scientific Computing*. McGraw-Hill Higher Education, 2001.

[48] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.

[49] G.E. Hinton. Training products of experts by minimizing contrastive divergence. 14(8):1771–1800, 2002.

[50] Georey E. Hinton. Products of experts. In *Ninth International Conference on Artificial Neural Networks*, pages 1–6, 1999.

[51] P. Hoyer and A. Hyvärinen. Independent component analysis applied to feature extraction from colour and stereo images. *Network: Computation in Neural Systems*, 11(3):191–210, 2000.

[52] Peter J. Huber. Projection pursuit. *The Annals of Statistics*, 13(2):435–475, 1985.

[53] A. Hyvärinen. Regression using independent component analysis. In *Proc. Int. Conf. on Articial Neural Networks*, pages 491–496, Edinburgh, UK, 1999.

[54] A. Hyvärinen and E. Oja. Independent component analysis: algorithms and applications. *Neural Netw.*, 13(4-5):411–430, 2000.

[55] Aapo Hyvärinen. Sparse code shrinkage: denoising of nongaussian data by maximum likelihood estimation. *Neural Comput.*, 11(7):1739–1768, 1999.

[56] Aapo Hyvärinen and Ella Bingham. Connection between multilayer perceptrons and regression using independent component analysis. *Neurocomputing*, 50:211–222, 2003.

[57] Aapo Hyvarinen, Juha Karhunen, and Erkki Oja. *Independent Component Analysis*. Wiley-Interscience, May 2001.

[58] A. Hyvrinen. Fast and robust fixed-point algorithms for independent component analysis. *IEEE Transactions on Neural Networks*, 10(3):626–634, 1999.

[59] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computing*, 3:79–87, 1991.

[60] V. Jain, J. F. Murray, F. Roth, S. Turaga, V. Zhigulin, K. L. Briggman, M. N. Helmstaedter, W. Denk, and H. S. Seung. Supervised learning of image restoration with convolutional networks. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, 2007.

[61] J. Kemeny, A. Devgan, R. Hagaman, and X. Wu. Analysis of rock fragmentation using digital image processing. *Geotechnical Engineering*, 119(7):1144–1160, 1993.

[62] Michael Kirby. *Geometric Data Analysis: An Empirical Approach to Dimensionality Reduction and the Study of Patterns*. John Wiley & Sons, New York, 2001.

[63] Peter Kovesi. Image features from phase congruency. *Videre: A Journal of Computer Vision Research*, 1(2), 1999.

[64] Sanjiv Kumar and Martial Hebert. Discriminative fields for modeling spatial dependencies in natural images. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, December 2003.

[65] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[66] Hugo Larochelle, Dumitru Erhan, Aaron Courville, James Bergstra, and Yoshua Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 473–480, New York, NY, USA, 2007. ACM.

[67] M. Larsen and M. Rudemo. Estimation of tree positions from aerial photos. In *Proceedings of the 1997 Swedish Symposium on Image Analysis*, 1997.

[68] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and Muller K., editors, *Neural Networks: Tricks of the trade*. Springer, 1998.

[69] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. Gradient-based learning for object detection, segmentation and recognition. Technical report, AT&T Labs, 1999.

[70] Yann LeCun, Sumit Chopra, Raia Hadsell, Ranzato Marc'Aurelio, and Fu-Jie Huang. A tutorial on energy-based learning. In G. Bakir, T. Hofman, B. Schölkopf, A. Smola, and B. Taskar, editors, *Predicting Structured Data*. MIT Press, 2006.

[71] Yann LeCun and Fu Jie Huang. Loss functions for discriminative training of energy-based models. In *Proc. of the 10-th International Workshop on Artificial Intelligence and Statistics (AIStats'05)*, 2005.

[72] I. Levner and V. Bulitko. Machine learning for adaptive image interpretation. In *Proceedings of the 16th Annual Innovative Applications of Artificial Intelligence Conference*, San Diego, CA, 2004.

[73] I. Levner and V. Bulitko. Comparison of machine learned image interpretation systems in the domain of forestry. In *WACV05*, pages I: 421–426, 2005.

[74] I. Levner, V. Bulitko, G. Lee, L. Li, and R. Greiner. Automated feature extraction for object recognition. In *Proceedings of the Image and Vision Computing New Zealand conference*, Palmerson North, NZ, 2003.

[75] I. Levner and H. Zhang. Classification-driven watershed segmentation. *IEEE Transactions on Image Processing*, 16(5):1437–1445, May 2007.

[76] Ilya Levner, Hong Zhang, and Russ Greiner. Heterogeneous stacking for classification driven watershed segmentation. *EURASIP Journal on Advances in Signal Processing*, 2008(Article ID 485821 (9 pages)), 2008.

[77] O. Lezoray and H. Cardot. Bayesian marker extraction for color watershed in segmenting microscopic images. In *Proceedings of the 16th International Conference on Pattern Recognition*, pages 739–742, 2002.

[78] O. Lezoray and H. Cardot. Cooperation of color pixel classification schemes and color watershed: a study for microscopic images. *IEEE Transactions on Image Processing*, pages 783–789, July 2002.

[79] N. H. Maerz, T. C. Palangio, and J. A. Franklin. Wipfrag image based granulometry system. In *Measurement of Blast Fragmentation*, pages 91–99. Franklin Katsabanis (eds), 1996.

[80] M.A. Maloof, P. Langley, S. Sage, and T.O. Binford. Learning to detect rooftops in aerial images. In *Proceedings of the Image Understanding Workshop*, pages 835–845, San Francisco, CA, 1997. Morgan Kaufmann.

[81] W. Mann and T. Binford. Successor: Interpretation overview and constraint system. *IUW*, pages 1505–1518, 1996.

[82] David R. Martin, Charless C. Fowlkes, and Jitendra Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(5):530–549, 2004.

[83] Gonzalo Martínez-Muñoz and Alberto Suárez. Switching class labels to generate classification ensembles. *Pattern Recognition*, 38(10):1483–1494, 2005.

[84] A. R. McIntosh, F. L. Bookstein, J. V. Haxby, and C. L. Grady. Spatial pattern analysis of functional brain images using partial least squares. *Neuroimage*, 3(3 Pt 1):143–157, June 1996.

[85] D. McKeown, W. Harvey, and J. McDermott. Rule-based interpretation of aerial imagery. *PAMI*, 7(5):570–585, 1985.

[86] Fernand Meyer. Topographic distance and watershed lines. *Signal Process.*, 38(1):113–125, 1994.

[87] D. Mukherjee, Y. Potapovich, I. Levner, and H. Zhang. Oil sand ore size analysis through learning image and shape features. *Submitted to Pattern Recognition*, 2008.

[88] D. Murgu. Individual tree detection and localization in aerial imagery. Master's thesis, Department of Computer Science, University of British Columbia, 1996.

[89] F. Ning, D. Delhomme, Yann LeCun, F. Piano, Léon Bottou, and Paolo Emilio Barbano. Toward automatic phenotyping of developing embryos from videos. *IEEE Transactions on Image Processing*, 14(9):1360–1371, 2005.

[90] P. P. Ohanian and R. C. Dubes. Performance Evaluation for Four Classes of Textural Features. *Pattern Recognition*, 25:819–833, 1992.

[91] T. Ojala and M. Pietikinen. Unsupervised texture segmentation using feature distributions. *Pattern Recognition*, 32(3):477–486, 1999.

[92] Bruno A. Olshausen and David J. Field. Natural image statistics and efficient coding. *Network*, (7):333–339, 1996.

[93] Pekka Orponen. Computational complexity of neural networks: a survey. *Nordic J. of Computing*, 1(1):94–110, 1994.

[94] Stanley Osher and James A. Sethian. Fronts propagating with curvature-dependent speed: algorithms based on hamilton-jacobi formulations. *J. Comput. Phys.*, 79(1):12–49, 1988.

[95] P. Paclik, T.C.W. Landgrebe, D.M.J. Tax, and R.P.W. Duin. On deriving the second-stage training set for trainable combiners. In *Multiple Classifier Systems Conference (MCS)*, 2005.

[96] J. Peng and B. Bhanu. Closed-loop object recognition using reinforcement learning. *PAMI*, 20(2):139–154, 1998.

[97] A. Pentland, B. Moghaddam, and T. Starner. View-based and modular eigenspaces for face recognition. In *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition (CVPR'94)*, Seattle, WA, June 1994.

[98] D. S. Phatak and I. Koren. Connectivity and performance tradeoffs in the cascade correlation learning architecture. *IEEE Transactions on Neural Networks*, 5:930–935, 1994.

[99] A.J. Pinz. A computer vision system for the recognition of trees in aerial photographs. In *J.C.Tilton (ed.), Multisource Data Integration in Remote Sensing*, pages 111–124, Maryland: NASA, 1991.

[100] M. Polak, H. Zhang, and M. Pi. An evaluation metric for image segmentation of multiple objects. *submitted to Image and Vision Computing*.

[101] R.J. Pollock. A model-based approach to automatically locating tree crowns in high spatial resolution images. In J. Desachy, editor, *Image and Signal Processing for Remote Sensing*, 1994.

[102] Trygve Randen and John Hakon Husoy. Filtering for texture classification: A comparative study. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4):291–310, 1999.

[103] M. Ranzato, C.S. Poultney, S. Chopra, and Y. LeCun. Efficient learning of sparse overcomplete representations with an energy-based model. In Scholkopf et al., editor, *Advances in Neural Information Processing Systems 19*, Cambridge, MA, 2006. MIT Press.

[104] Y. Raviv and N. Intrator. Bootstrapping with noise: An effective regularization technique. *Connection Science, Special issue on Combining Estimators*, 8:356–372, 1996.

[105] R. Reed, R. Marks II, and S. Oh. Similarities of error regularization, sigmoid gain scaling, target smoothing and training with jitter. *IEEE Transactions on Neural Networks*, 6(3):529–538, 1995.

[106] R. Rimey and C. Brown. Control of selective perception using bayes nets and decision theory. *International Journal of Computer Vision*, 12:173–207, 1994.

[107] J.L. Rodgers, Nicewander W.A, and L. Toothaker. Linearly independent, orthogonal, and uncorrelated variables. *The American Statistician*, 38(2):133–134, 1984.

[108] Stefan Roth. *High-Order Markov Random Fields for Low-Level Vision*. PhD thesis, Brown University, 2007.

[109] Stefan Roth and Michael J. Black. Fields of experts: A framework for learning image priors. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*, pages 860–867, Washington, DC, USA, 2005. IEEE Computer Society.

[110] D. E. Rumelhart, G. E. Hinton, and J. L. McClelland. A general framework for parallel distributed processing. In D. E. Rumelhart, J. L. McClelland, et al., editors, *Parallel Distributed Processing: Volume 1: Foundations*, pages 45–76. MIT Press, Cambridge, 1987.

[111] J. Schleifer and B. Tessier. FRAGSCAN: A tool to measure fragmentation of blasted rock. In *Measurement of Blast Fragmentation*, pages 73–78. Franklin & Katsabanis (eds), 1996.

[112] Matthias Scholz, Fatma Kaplan, Charles L. Guy, Joachim Kopka, and Joachim Selbig. Non-linear pca: a missing data approach. *Bioinformatics*, 21(20):3887–3895, 2005.

[113] Holger Schwenk. The diabolo classifier. *Neural Comput.*, 10(8):2175–2200, 1998.

[114] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, 2000.

[115] E P Simoncelli and E H Adelson. Noise removal via Bayesian wavelet coring. In *Third Int'l Conf on Image Proc*, volume I, pages 379–382, Lausanne, 1996. IEEE Sig Proc Society.

[116] L. Sirovich and M. Kirby. Low dimensional procedure for the characterization of human faces. *Journal of Optical Society of America*, 4(3):519–524, 1987.

[117] Yee Whye Teh, Max Welling, Simon Osindero, and Geoffrey E. Hinton. Energy-based models for sparse overcomplete representations. *J. Mach. Learn. Res.*, 4(7-8):1235–1260, 2004.

[118] K. Tieu and P. Viola. Boosting image retrieval. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 228–235, 2000.

[119] Kai Ming Ting and Ian H. Witten. Issues in stacked generalization. *Journal of Artificial Intelligence Research*, 10:271–289, 1999.

[120] van der Schaaff A van Hateren JH. Independent component filters of natural images compared with simple cells in primary visual cortex, 1997.

[121] Xiaoli Wang, Mark Polak, Vadim Bulitko, and Hong Zhang. Machine learning for adaptive parameter selection in ore image segmentation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI), Workshop on Learning in Computer Vision*, Pittsburgh, Pennsylvania, 2005.

[122] A. Webb. *Statistical Pattern Recognition*. John Wiley & Sons, 2002.

[123] J. Weston, O. Chapelle, A. Elisseeff, B. Schlkopf, and V. Vapnik. Kernel dependency estimation. In S. Thrun Becker, S. and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15, pages 873–880, Cambridge, MA, USA, 2003. MIT Press.

[124] J. Weston, B. Schlkopf, and O. Bousquet. Joint kernel maps. In A. Prieto F. Sandoval Cabestany, J., editor, *Proceedings of the 8th International Work-Conference on Artificial Neural Networks (Computational Intelligence and Bioinspired System)*, volume LNCS 3512, pages 176–191, Berlin Heidelberg, Germany, 2005. Springer-Verlag.

[125] Peter M Williams. Bayesian regularisation and pruning using a laplace prior. *Neural Computation*, 7:117–143, 1995.

[126] D. H. Wolpert. Stacked generalisation. *Neural Networks*, 5:241–259, 1992.

[127] M. Wulder, K.O. Niemann, and D. Goodenough. Error reduction methods for local maximum filtering. In *The Proceedings of the 22nd Symposium of the Canadian Remote Sensing Society*, Victoria, British Columbia, 2000.

[128] F.X. Yan, H. Zhang, and C.R. Kube. A multistage adaptive thresholding method. 26(8):1183–1191, June 2005.

[129] T.S. Yu and K.S. Fu. Recursive contextual classification using a spatial stochastic model. *Pattern Recognition*, 16(1):89–108, 1983.

# Appendix A

# Oil Sand Ore Granulometry

The field of granulometry aims to determine particle sizes of a given granular substance. For mining applications, it has been carried out using tedious and laborious procedures such as sieving, centrifugation, and sedimentation. The advent of inexpensive fast computing power, along with the availability of inexpensive portable video cameras, enables image-based granulometry to become feasible. The image-based approach offers numerous advantages including the fact that it is: non-evasive allowing measurements on weak rock and ore which tend to break up when screened; non-disruptive to production; relatively fast and therefore high throughput; physically unaffected by size or volume of rock.

One key reason we are interested in this domain, is that the many difficulties present within the domain on image-based granulometry are also present within numerous other image interpretation domains. Consider the input image shown at the top of Figure 1.1. Each fragment, while belonging to the same class, varies in size, shape, texture and reflectance. In addition, separate fragments (i.e., objects) can touch one another, requiring the use of pixel grouping techniques in order to properly measure the size of individual fragments. To complicate things further, some fragments are covered by fine particles and hence have a similar appearance to the background. Such difficulties are found in many other domains as well, although usually not all at once. Hence, we expect an algorithm performing well on this image-based granularity domain, in principle, to be useful for numerous other applications.

Unfortunately, to date, no algorithm is capable of performing adequately in this domain. In the last 20 years, many special purpose systems have been developed (see [23] and references within) with the aim of automating the image-based granulometry task. The Wipfrag system, developed by Wipware, Inc [79], attempts to build an edge net in the image, using an edge detector. Based on this edge net, the particles are delineated. Similarly, the Split system [61], uses a combination of edge detection and edge-following algorithms. Shadowed areas, such as those

between particles, are thresholded as a step toward producing binary images indicating particle and non-particle areas. The particles are delineated by searching for large gradient paths ahead of sharp shadow convexities to separate clusters of touching particles. A watershed algorithm further divides the touching particles. The Fragscan system [111], appears to be the only commercial system not using edge detection. Instead a series of opening operations are used to simulate sieving. A threshold is obtained by successive filters, as well as automatic threshold selection, and the image is converted to binary, before a series of opening operations is completed. A more recent system, called the Ore Size Analyst (OSA) [23], uses a sequence of processes to delineate particles. In order, the processing steps are: noise removal, contrast enhancement via local histogram equalization, adaptive thresholding, and postprocessing consisting of hole filling and morphological opening operations. One drawback of the aforementioned systems is that the hand-tuned parameters are sensitive to changes in illumination, particle size, non-uniform texture and a host of other variabilities typically present within images. As a result, finding globally optimal paraments is difficult. To overcome this problem an improved version of OSA uses the Adaptive Object Recognition framework (ADORE) [26, 73, 121], which learns to select parameter settings that optimize performance on an image by image basis (i.e., ADORE performs adaptive parameter selection).

# Appendix B

# Forest Inventory Building (from [73])

Forest maps and inventories have become a critical tool for wood resource management (planting and cutting), ecosystem management and wildlife research. Unfortunately, forest mapping at the level of individual trees is a continuous and costly undertaking. Canada alone has an estimated 344 million hectares of forests to inventory on a 10-20 year cycle [101].

At these scales, ground-based surveys and inventories are not feasible. Researchers have therefore turned to developing automated systems to produce forest maps from airborne images, LIDAR and other 2D/3D data sources. The final goal is to measure the type (species), position, height, crown diameter, wood volume and age class for every tree in the survey area.

The task of large-scale forest mapping from aerial images presents formidable challenges, including: (i) massive amounts of high-resolution data, in order to recognize and measure individual tree crowns, (ii) construction and maintenance of (and providing access to) very large databases; Canada alone has an estimated $10^{11}$ trees, (iii) geo-referencing of airborne images for validation purposes, (iv) orthorectification of aerial images, particularly given that elevation maps are often unavailable at the required accuracy . Of a particular interest are the challenges created by the image content, including variations in sun and camera angles and the resulting overlap between shadows and tree crowns. These challenges are known to have an adverse effect on special purpose algorithms for individual tree identification [22]. In fact, the task is substantially challenging even to expert human interpreters resulting in up to 40% error in comparison to ground-based surveys [40].

## B.1 Special Purpose Forestry Systems

A number of approaches have been proposed for creating forest inventories from aerial images. Image-based (model-free) approaches use simplifying assumptions about forest images. For example, [40, 99] use a token-based recognition approach

which assumes a high level of contrast between the tree crown and the surrounding area. They deal with canopy feature extraction almost exclusively in terms of finding image features which evidence different types of tree canopies. A current example of this approach is the ITC system [40] where tree canopies are detected and classified by a mixture of "valley-finding" (low intensity iso-contours), peak intensity detection [127] as well as texture, structure and contextual image features. This falls within traditional image segmentation and region labeling strategies where there is no explicit need to model features in terms of known tree attributes or specific 3D geometric models. Consequently, the approach is designed to apply where there is sufficient spatial separation between trees. Unfortunately, the performance can degrade significantly as such methods are applied to naturally occurring dense forests with overlapping tree crowns.

Another approach uses example-based image models. The underlying idea is to compare pre-specified tree crown image(s) with the image at hand. Typically such methods, e.g., [88], have a collection of example tree crowns (i.e., templates) which they match to the image. Drawbacks of such approaches include the need to collect a very large database of templates to account for differences in tree species, size, the slant of the terrain and illumination.

Model-based approaches take advantage of explicit tree crown models that are matched to image regions. While minimizing the amount of image feature processing, elementary image features are used to hypothesize large numbers of regions for matching with 3D CAD tree models via computer graphics methods. For example, the STCI system [101] uses a template matching approach, however, unlike the example-based approaches discussed above, the crown templates are synthesized from a tree crown model. The upper part of a tree crown (known as "sun crown") is modelled as a generalized ellipsoid of revolution and ray-tracing techniques are used to generate templates [67]. Model-based approaches typically rely on detecting image features such as crown peaks and normally use pre-generated templates to match projected models with image data. The latter technique can require generation of many templates for different slant angles, tree types, etc. Additionally, model-based methods use simple shadow models and simplified 3D CAD models typically representing only the canopy envelope. As a result, the model-based approaches are often unable to deal with natural variations in foliage, branches, and the resulting irregular canopy boundaries.

## B.2 Machine Learning Approaches

All of the approaches reviewed in the previous section are promising, at least in a laboratory setting, but share some common drawbacks. First, they were carefully crafted in a development process that required both time and expertise. More im-

portantly, this development process exploited domain properties, such as whether the tree canopies are separated or overlapping, whether the ground is flat or mountainous, or whether the forest has a homogeneous or heterogeneous species composition. Similarly, assumptions about the properties and position of the sensor(s) are integrated in the system design. As a result, these systems work within a narrow set of operating conditions, and cannot be applied under different conditions without re-engineering.

To put the control of object recognition on a stronger theoretical foundation, researchers have attempted using Bayes nets (e.g., TEA1 [106] and SUCCESSOR [81]). Unfortunately, the design of Bayes nets can itself become an *ad hoc* knowledge engineering process. Other researchers tried to eliminate the knowledge acquisition bottleneck by machine-learning control policies from expert annotated examples. For instance, in [3] researchers used genetic algorithms to learn target recognition strategies, while reinforcement learning has been used in [24] to learn control strategies or in [96] to find parameters for vision procedures. Maloof et al. trained classifiers to accept or reject data instances between steps of a static sequence of procedures [80]. More recently, Baysian approaches have regained popularity and evolved into sophisticated systems. One such system, is based on the Hierarchical Hidden Markov Random Field (HHMRF) model presented in [18]. The system, based on ideas from [17], first labels individual image pixels at multiple resolutions using an MDL based K-means algorithm, which uses a mixture of Gaussians to model the multi-spectral pixel distributions seen during training. Then the HHRMF algorithm merges the results produced independently at each resolution into a final hypothesis which is consistent with all layers of the multi-resolution hierarchy. The HHMRF system, in combination with ADORE, presented in the previous appendix, is currently one of the best segmentation systems for forest image annotation.

# Appendix C

# Evaluation Criteria

To evaluate the performance of the algorithm(s) several criteria are used. Respectively, $TP$, $TN$, $FP$, $FN$, stand for the number of samples (i.e., pixels) being labeled as true positive, true negative, false positive, false negative.

**Intersection-over-union,** $\frac{|\boldsymbol{A}\cap\boldsymbol{B}|}{|\boldsymbol{A}\cup\boldsymbol{B}|}$ abbreviated as (I/U) for binary labeling $A$ and $B$, and defined as $\frac{TP}{TP+FP+FN}$ and is also known as the Jaccard measure.

**Pixel Accuracy** defined as $\frac{TP+TN}{TP+TN+FP+FN}$.

**Precision** defined as $\frac{TP}{TP+FP}$ and is also known as positive predictive value.

**Recall** defined as $\frac{TP}{TP+FN}$ and is also known as sensitivity.

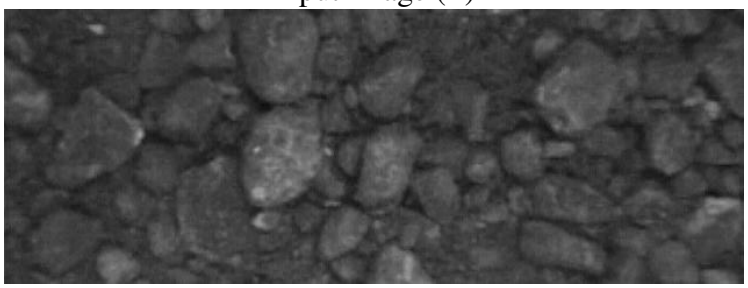
**Labeling score** defined as $\mathcal{L} = min(\mathcal{S}(A,B), \mathcal{S}(B,A))$, with

$$\mathcal{S}(A,B) = \sum_{j}^{m} \left[ \sum_{i}^{n} \left( \frac{|A_j \cap B_i|}{|A_j \cup B_i|} \frac{B_i}{\bigcup\limits_{\substack{|A_j \cap B_i| \\ \neq 0}} B_i} \right) \frac{A_j}{\bigcup\limits_{j} \mathrm{A_j}} \right] \tag{C.1}$$

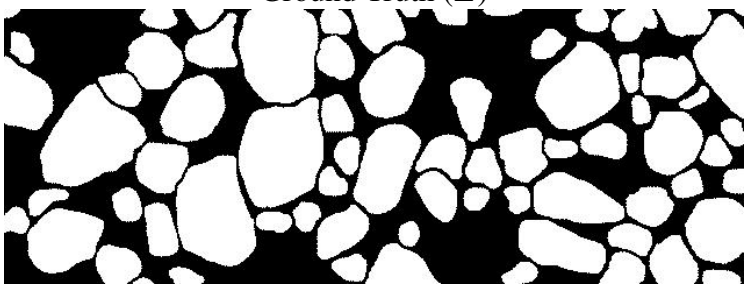
where $A_j$ is a connected component in image $A$ and $B_i$ is a connected component in image $B$. This labeling score, described in [100], is a form of local intersection-over-union (I/U) whereby both errors at the pixel level and object level are penalized.

As an example, consider the segmentation in Figure C.1 on p. 139. At the pixel level it is nearly perfect. However, since the objective is delineate individual objects a high pixel level accuracy does not necessarily imply a good object level segmentation.

Input Image ($\boldsymbol{I}$)



Ground Truth ($\boldsymbol{L}$)



Candidate Object Segmentation ($\boldsymbol{L}_{region}$)

| | |
|---|---|
| I/U | 0.9213 |
| Precision | 0.9350 |
| Recall | 1 |
| Pixel Accuracy | 0.9592 |
| Labeling Score | **0.0598** |

Figure C.1: An example of a poor object segmentation. While the foreground-background separation is nearly perfect, the single connected blob corresponds to many underlying objects. Hence while the pixel level scores are high the object level score is extremely low.