SCENE LABELING WITH SUPERVISED CONTEXTUAL MODELS

by

Mojtaba Seyedhosseini

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Electrical and Computer Engineering

The University of Utah

August 2014

The University of Utah Graduate School

STATEMENT OF DISSERTATION APPROVAL

The dissertation of **Mojtaba Seyedhosseini**

has been approved by the following supervisory committee members:

Tolga Tasdizen	, Chair	**06/03/2014** Date Approved
Behrouz Farhang	, Member	**06/23/2014** Date Approved
Rong-Rong Chen	, Member	**06/23/2014** Date Approved
Sarang Joshi	, Member	Date Approved
Guido Gerig	, Member	Date Approved

and by **Gianluca Lazzi**, Chair of

the Department of **Electrical and Computer Engineering**

and by David B. Kieda, Dean of The Graduate School.

ABSTRACT

Scene labeling is the problem of assigning an object label to each pixel of a given image. It is the primary step towards image understanding and unifies object recognition and image segmentation in a single framework. A perfect scene labeling framework detects and densely labels every region and every object that exists in an image. This task is of substantial importance in a wide range of applications in computer vision. Contextual information plays an important role in scene labeling frameworks. A contextual model utilizes the relationships among the objects in a scene to facilitate object detection and image segmentation. Using contextual information in an effective way is one of the main questions that should be answered in any scene labeling framework.

In this dissertation, we develop two scene labeling frameworks that rely heavily on contextual information to improve the performance over state-of-the-art methods. The first model, called the *multiclass multiscale contextual model (MCMS)*, uses contextual information from multiple objects and at different scales for learning discriminative models in a supervised setting. The MCMS model incorporates cross-object and interobject information into one probabilistic framework, and thus is able to capture geometrical relationships and dependencies among multiple objects in addition to local information from each single object present in an image. The second model, called the *contextual hierarchical model (CHM)*, learns contextual information in a hierarchy for scene labeling. At each level of the hierarchy, a classifier is trained based on downsampled input images and outputs of previous levels. The CHM then incorporates the resulting multiresolution contextual information into a classifier to segment the input image at original resolution. This training strategy allows for optimization of a joint posterior probability at multiple resolutions through the hierarchy. We demonstrate the performance of CHM on different challenging tasks

such as outdoor scene labeling and edge detection in natural images and membrane detection in electron microscopy images.

We also introduce two novel classification methods. *WNS-AdaBoost* speeds up the training of AdaBoost by providing a compact representation of a training set. *Disjunctive normal random forest (DNRF)* is an ensemble method that is able to learn complex decision boundaries and achieves low generalization error by optimizing a single objective function for each weak classifier in the ensemble.

Finally, a segmentation framework is introduced that exploits both shape information and regional statistics to segment irregularly shaped intracellular structures such as mitochondria in electron microscopy images.

CONTENTS

ACKNOWLEDGMENTS

I would like to express my gratitude to my adviser, Tolga Tasdizen, for his invaluable support and encouragement. His immense knowledge and inspiration helped me to overcome many difficulties during my research. He taught me how I should keep my eyes on the big picture while I am working on the details of a problem. I could not have asked for a better adviser, both personally and scientifically. I also thank my other committee members, Behrouz Farhang, Guido Gerig, Sarang Joshi, and Rong-Rong Chen, for their helpful comments on my proposal that shaped the structure of this dissertation.

Many thanks go to the Scientific Computing and Imaging Institute for providing a comfortable research environment and powerful computing resources. Specifically, I thank the support team members for their generous help.

I am also grateful to other members in our research team with whom I have interacted during my PhD. In particular, I would like to acknowledge Antonio Paiva, Elizabeth Jurrus, Luke Hogrebe, Ting Liu, Cory Jones, Mehdi Sajjadi, and Paul Rosen, who helped me at different points with helpful comments and scientific discussions.

I want to thank our collaborators at the NCMIR Institute who helped me with the biological part of this research. Particularly, I thank Mark Ellisman, Rick Giuly, Alex Perez, Jeffrey Bush, and David Lee for their feedback and discussions about the methods developed in this dissertation.

For my friends, thank you all for the fun and memorable moments that you created for me during these years. I'll surely miss the bbqs and movie nights that we had together.

To my parents who are an endless source of love and encouragement. Their support was the most important thing that kept me on track. It is to them that I dedicate this

work. My deepest appreciation goes to my sister and my brothers for their spiritual support. None of this would have been possible without their encouragement.

Finally, I would like to acknowledge my funding support. This research was supported by *NIH 1R01NS075314-01(TT,MHE)* and *NSF IIS-1149299 (TT)*.

CHAPTER 1

INTRODUCTION

The main focus of this dissertation is the development of frameworks that use contextual information in a supervised setting for scene labeling. Scene labeling, *i.e.,* assigning an object label to each image pixel, is a fundamental problem in computer vision and is commonly used as the primary step for a wide range of applications [1]. It integrates the problems of detection and segmentation in a single framework [2]. For instance, in a dataset of horse images, scene labeling can be thought of as the task of labeling each pixel as part of a horse or nonhorse, *i.e.,* background. In more complicated cases such as outdoor scene images, it might require multiple labels, *e.g.,* buildings, cars, roads, sky, etc. This general definition can also be extended to the edge detection problem where each pixel is classified as edge or nonedge in a binary-decision framework.

Pixels cannot be labeled based only on a small region around them. For example, it is almost impossible to distinguish a pixel belonging to the sky from a pixel belonging to the sea by looking only at a small patch around them. Therefore, a scene labeling framework needs to take into account short-range and long-range contextual information. Contextual information has been widely used to resolve ambiguities in high-level problems in computer vision such as image segmentation [3], object detection [4], and scene understanding [5,6]. Contextual information can refer to either interobject configuration, *e.g.,* a segmented horse's body may suggest the position of its legs [3], or intraobject dependencies, *e.g.,* the existence of a keyboard in an image implies that there is very likely a mouse near it [4]. From the Bayesian point of view, contextual information can be interpreted as the probability image map of an object, which carries prior information in the maximum a posteriori (MAP) pixel

classification problem.

An important question about any scene labeling method is how it takes contextual information into account. The main challenge is to pool contextual information from a large neighborhood while keeping the complexity tractable [2]. A common approach is to use a series of cascaded classifiers [3, 5–7]. In this architecture, each classifier is sequentially trained using the outputs of the previous classifiers as inputs. This gradually increases the area of influence and allows later classifiers in the series to obtain contextual information from larger neighborhood areas. However, the performance of series architecture can be improved by using more informative contextual information. In this dissertation, we introduce two contextual models, which are able to learn cross-object and interobject contextual information in an effective manner. They exploit contextual information at multiple resolutions/scales in a probabilistic framework. The proposed methods outperform state-of-the-art methods on different applications and can be used as the first step towards scene understanding. Some results of our contextual models are shown in Fig. 1.1.

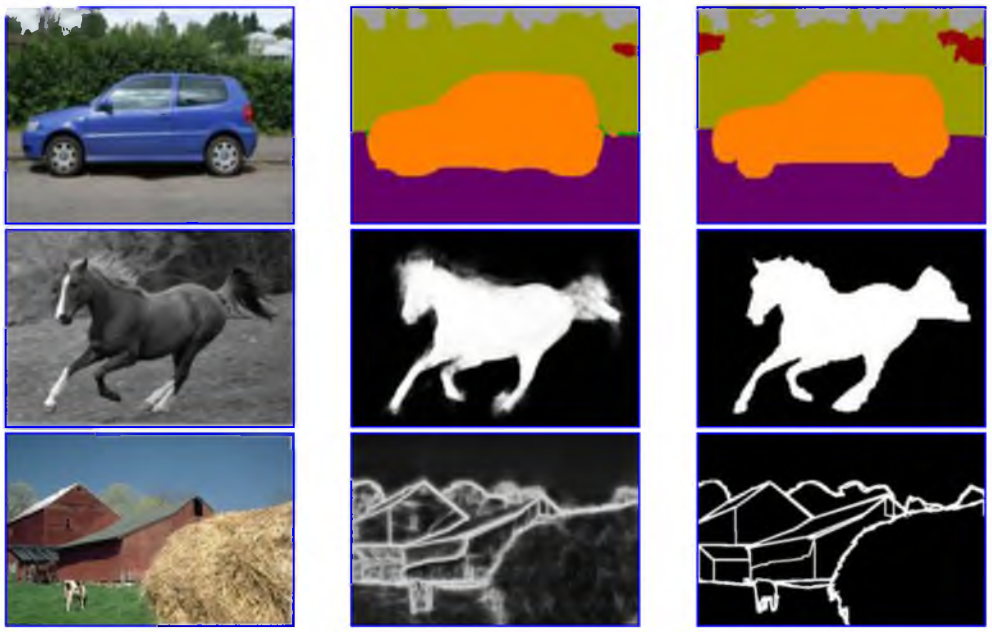


Figure 1.1: Results of proposed contextual models on different applications. **First row:** Scene labeling (Stanford background dataset [8]). **Second row:** Horse segmentation (Weizmann dataset [9]). **Third row:** Edge detection (Berkeley dataset [10]). See Chapters 2, 3 for details.

1.1 Neural Circuit Reconstruction

Our models are motivated by the problem of reconstruction of the connectome, *i.e.,* the map of connectivity of all neurons in the mammalian nervous system [11], which is a challenge facing neuroscientists [7]. Electron microscopy (EM) is an image acquisition technique that can generate high resolution images with enough details for this problem [12]. The sheer size of a typical EM dataset, often approaching tens of terabytes [13], makes manual analysis infeasible [14]. The only complete reconstruction of a nervous system has been performed for the nematode C. elegans, which contains 302 neurons and about 6000 synapses [15, 16]. The manual labeling of this small organism is reported to take more than a decade [14]. EM acquisition techniques can be used to obtain much larger datasets containing several orders of magnitude more neurons than the C. elegans. These datasets might contain thousands of neurons with millions of synapses [14,17]. Hence, automated segmentation methods are required to process and segment these images.

Fully automatic reconstruction of the connectome remains a challenging problem because of the noisy texture, irregular shapes, complex structures, and the large variations in the physical topologies of cells [18, 19]. Moreover, different structures have similar local appearances, which makes it difficult for the automatic method to detect and segment them consistently. A robust segmentation method must overcome these issues.

The connectome problem can be formulated in a scene labeling framework where each pixel has to be labeled as an object of interest. For example, for membrane detection each pixel in the image is labelled as membrane or nonmembrane. Our proposed models make a wide use of contextual information to overcome the aforementioned challenges. The proposed methods surpass both the accuracy and computational efficiency of state-of-the-art methods and facilitate the analyzing and interpretation of the EM images data, paving the way for understanding neurodegenerative diseases at the microscopic level.

1.2 Classifiers

We also develop two novel supervised learning/classification methods that can be applied to general machine learning problems as well as in our contextual models. In the first method, a new AdaBoost learning framework, called *WNS-AdaBoost*, is proposed that significantly speeds up the learning process of AdaBoost. For this purpose, we introduce a novel sampling strategy, weighted novelty selection (WNS), and combine it with AdaBoost to obtain the WNS-AdaBoost framework. WNS is a sampling method that reduces the number of data points by selecting representative points from the dataset. It also determines a corresponding weight for each of these selected points that shows the importance of that point and aims at preserving the distribution of the original data. By reducing the number of training samples, the proposed framework significantly reduces the training time. The output of the WNS algorithm is then used by AdaBoost, or any of its variants, to learn a discriminative model. This is achieved by training AdaBoost on the representative set of data points and initializing the weight distribution with the weights obtained from WNS after normalization.

In the second method, we propose a new classifier, called the *disjunctive normal decision tree*, which allows the linear discriminants at each node of a decision tree to be at any arbitrary orientation. The main advantage of our approach is that it learns all the weak learners of the decision tree in a unified framework. To be clear, unlike conventional decision trees and their variants that learn the splitting function at each node independently, our approach allows weak learners of different nodes to interact with each other during the training because it minimizes a single global objective function. We will employ this new decision tree to build a new random forest, called *disjunctive normal random forests (DNRF)*, which outperforms conventional random forests.

1.3 Intracellular Component Segmentation

In addition to the neural circuit map, neuroscientists are interested in the morphology and distribution of intracellular components. For example, abnormal mitochondria morphology can be seen in Parkinson's disease-related genes [20], or

geometrical properties of mitochondria can be used to distinguish cancer cells from normal cells [21]. Moreover, an accurate mitochondria segmentation would improve cell segmentation results by distinguishing mitochondria membranes from other cell membranes [22]. The texture and physical topologies of intracellular components are highly variable [18] (Fig. 1.2). Even though our contextual model, *i.e.,* multiclass multiscale contextual model (MCMS), is able to segment mitochondria, its computational complexity is dominated by the complexity of membrane detection, which is higher than the complexity of mitochondria segmentation. Moreover, manual labeling of membranes is more expensive compared to manual labeling of mitochondria and is unnecessary when the main target is mitochondria. Finally, our contextual models do not take shape information into account, but the shape information is an important clue to mitochondria identity. To address these problems, we propose a specific segmentation framework for mitochondria in EM images. We take advantage of the power of algebraic curves in finding ambiguous edges in cluttered backgrounds to estimate the boundary of mitochondria and extract informative shape and textural features from images. The regional features, *i.e.,* textural features from image regions, are more robust and informative compared to pixel features.

1.4 Contributions

The following list outlines the specific contributions of this dissertation:

1. *Develop a scene labeling framework that is able to use contextual information*

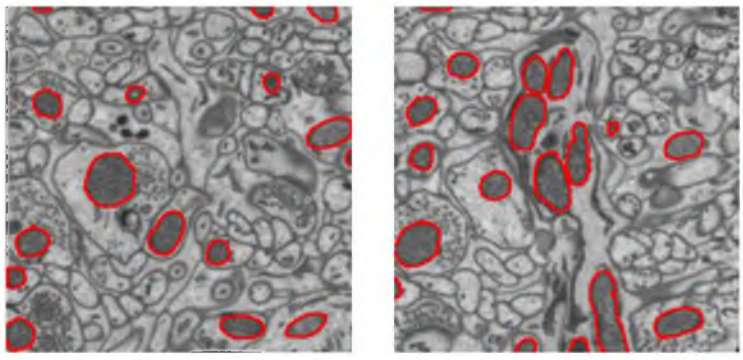


Figure 1.2: Mitochondria (red outlines) appear in different shapes and intensities in EM images. This variety and the existence of other similar structures make segmentation a difficult task.

at multiple scale and from multiple objects. The main advantage of the MCMS model is its ability to pool contextual information from a large neighborhood area in a series architecture without increasing the computational complexity. Moreover, this model is able to leverage contextual information from multiple objects in a single framework. This is discussed in more detail in Chapter 2 and [23, 24].

2. *Develop a scene labeling framework that is able to learn contextual information at multiple resolutions by minimizing a joint posterior distribution.* The contextual hierarchical model (CHM) improves the performance of MCMS by learning contextual information at multiple resolutions in a supervised framework. The multiresolution processing of context enables CHM to reach the state-of-the-art performance on different applications. More detail on this method is covered in Chapter 3 and [25].

3. *Develop a classification method that speeds up the training of AdaBoost.* The goal of this work is to provide a compact representation of training data for the AdaBoost classifier to speed up the training process. This approach can be used as a preprocessing step for any AdaBoost based classifier. See Chapter 4 and [26] for more details.

4. *Develop a classification method that improves the performance of the conventional random forest by minimizing a single objective function for each tree in the forest.* Conventional random forests are prone to overfitting in the presence of noise. Moreover, they only learn axis-aligned discriminants at each node. Our DNRF increases the generalization performance of random forests by allowing communication between different nodes in each tree. The unified optimization in DNRF allows the discriminants to be at arbitrary orientations. The details of this formulation and optimization can be found in Chapter 5.

5. *Develop a mitochondria segmentation method that combines both textural and shape information in a single framework.* Even though MCMS and CHM can be used for mitochondria segmentation, they do not take shape information

into account. We propose a segmentation method that uses algebraic curves to extract shape information. This method is more robust to shape variance in the presence of noise. See Chapter 6 and [27] for more details and segmentation results.

1.5 Software

Following the reproducible research instructions [28], we have made the codes for MCMS and CHM public to make it easier for other researchers to understand our methods. The MCMS method is implemented in *C++* and uses OpenCV [29] and Boost libraries [30]. It is a memory efficient code and can be trained on large datasets using a machine with 4 GB of memory. The code is available at `http://www.sci.utah.edu/~mseyed/Mojtaba_Seyedhosseini/MS.html`. The CHM code is mainly implemented in *MATLAB*, and the time consuming parts are implemented in *C* with a *MATLAB* wrapper. The code is available at `http://www.sci.utah.edu/~mseyed/Mojtaba_Seyedhosseini/CHM.html`.

1.6 Overview

Chapter 2 gives an overview of the series classifier idea and its probabilistic interpretation. Then, the importance of context sampling is discussed, and the multiscale contextual model is introduced. Next, the idea of a multiscale contextual model is expanded to the multiclass case, and the multiclass multiscale contextual model (MCMS) is proposed. The effectiveness of proposed models is shown on real datasets.

Chapter 3 discusses the general problem of scene labeling and gives an overview of existing methods for this problem. Then, the CHM is introduced, and its probabilistic formulation is derived. Next, we show that CHM optimizes a joint posterior function at multiple resolutions in a greedy way. We illustrate that CHM outperforms state-of-the-art methods on different datasets and for different applications.

In Chapter 4, we introduce the weighted novelty selection (WNS) as a clustering method. Then, we show how it can be used as a preprocessing step to give a compact representation of a training set. Next, we show how the combination of WNS and AdaBoost speeds up the training process with a minimal loss of accuracy. The

performance of WNS-AdaBoost is shown for two applications.

Chapter 5 gives an overview of conventional random forests and their limitations. Then, disjunctive normal decision trees (DNDTs) are introduced as building blocks of disjunctive normal random forests (DNRFs). We show that unlike conventional random forests, DNRFs are able to learn non-axis-aligned discriminants by optimizing a single objective function. The superior performance of DNRFs is shown on several binary and multiclass classification datasets.

Finally, Chapter 6 describes the specific segmentation method designed for mitochondria segmentation. First, we discuss the algebraic curves and their robustness to the noise. Then, we show how they can be used to extract meaningful features for segmenting mitochondria. The performance of the proposed method is illustrated on two EM datasets.

CHAPTER 2

MULTICLASS MULTISCALE SERIES CONTEXTUAL MODEL

Contextual information has been widely used as a rich source of information to segment multiple objects in an image. A contextual model utilizes the relationships between the objects in a scene to facilitate object detection and segmentation. However, using contextual information from different objects in an effective way for object segmentation remains a difficult problem. In this chapter, we introduce a novel framework, called the *multiclass multiscale (MCMS)* series contextual model, which uses contextual information from multiple objects and at different scales for learning discriminative models in a supervised setting. The MCMS model incorporates cross-object and interobject information into one probabilistic framework and thus is able to capture geometrical relationships and dependencies among multiple objects in addition to local information from each single object present in an image. We demonstrate that our MCMS model improves object segmentation performance in electron microscopy images and provides a coherent segmentation of multiple objects. By speeding up the segmentation process, the proposed method can allow neurobiologists to move beyond individual specimens and analyze populations paving the way for understanding neurodegenerative diseases at the microscopic level.

2.1 Introduction

Shape contexts are extremely rich descriptors [31] that have been used widely for solving high-level vision problems. Contextual information is interpreted as intraobject configurations and interobject relationships [3]. These attributes play an important role in scene understanding [32–34]. For example, the existence of a keyboard in an image suggests that there is very likely a mouse near it [4]. To be

precise, by contextual information we refer to the probability map of the target object that can be used as prior information together with the original image information to solve the maximum aposteriori (MAP) scene labeling problem.

There have been many methods that employ context for solving vision problems such as image segmentation or image classification. Markov random fields (MRF) [35] are one of the earliest and most widespread approaches. Lafferty *et al.* [36] showed that better results for discrimination problems can be obtained by modeling the conditional probability of labels given an observation sequence directly. This nongenerative approach is called the conditional random field (CRF). He *et al.* [37] generalized the CRF approach for the pixel classification problem by learning features at different scales of the image. Jain *et al.* [18] showed MRF and CRF algorithms perform about the same as simple thresholding in pixel classification for binary-like images. They proposed a new single-scale version of the convolutional neural network [38] strategy for restoring membranes in electron microscopic (EM) images. Compared to other methods, convolutional networks take advantage of context information from larger regions, but need many hidden layers. In their model the back propagation has to go over multiple hidden layers for the training, which makes the training step computationally expensive. Tu and Bai [3] proposed the auto-context algorithm, which integrates the original image features together with the contextual information by learning a series of classifiers. Similar to CRF, auto-context targets the posterior distribution directly without splitting it to likelihood and prior distributions. The advantage of auto-context over convolutional networks is its easier training due to treating each classifier in the series one at a time in sequential order. Although they used probabilistic boosting tree as classifier (PBT), auto-context is not restricted to any particular classifier, and different type of classifiers can be used. Jurrus *et al.* [7] employed artificial neural networks (ANN) in a series classifier structure, which learns a set of convolutional filters from the data instead of applying large filter banks to the input image.

Even though all the aforementioned approaches use contextual information together with the input image information to improve the accuracy of the achieved segmentation, they do not take contextual information from multiple objects into

account and thus are not able to capture dependencies between the objects. Torralba *et al.* [4] introduced boosted random field (BRF), which uses boosting to learn the graph structure of CRFs for multiclass object detection and region labeling. Desai *et al.* [39] proposed a discriminative model for multiclass object recognition that can learn intraclass relationships between different categories. The cascaded classification model [5] is a scene understanding framework that combines object detection, multiclass segmentation, and 3D reconstruction. Choi *et al.* [40] introduced a tree-based context model, which exploits dependencies among objects together with local features to improve the object detection accuracy.

While contextual models have been shown to be successful in several computer vision tasks, we propose a more effective way of extracting information from the context image, *i.e.,* the classifier output. We develop a novel framework that exploits contextual information from different scales and different objects to learn a discriminative model for object segmentation. To our knowledge, multiclass and multiscale contextual information have not been previously used in a unified framework for object segmentation. The combination of multiclass and multiscale schemes enables our method to make extensive use of contextual information and thus improves the segmentation accuracy.

We employ the series architecture in [7] and modify it in two important ways to provide more informative contextual information to the classifiers:

- multiscale contextual model: We apply a series of simple linear filters to the context image consecutively to generate a scale-space representation of the context and give the classifier access to samples of the scale space. The samples of the coarser scales are more informative and robust against noise due to the averaging. Therefore, this framework provides more information from the context for the classifier in a similar number of features.

- multiclass contextual model: We also introduce the multiclass series architecture by allowing the classifier for each object type access to the contextual information from each object type of the previous stage. This flow of cross-object information is achieved by feeding neighborhoods from the output of each classifier in the current stage to each classifier in the next stage. The proposed

multiclass framework is able to capture geometric relationships of objects and their dependencies which can be an important clue to their identity. For instance, the existence of mitochondria, *i.e.,* the objects with green boundary in Fig. 2.1, at a certain position in an electron microscopy image is a strong evidence that the existence of synapses, *i.e.,* the objects with yellow boundary in Fig. 2.1, is unlikely. Synapses are more likely in certain configurations and distances to cell membranes, *i.e.,* the red objects in Fig. 2.1.

We introduce a novel and powerful segmentation framework by employing multiscale and multiclass contextual model in a series classifier architecture. The multiclass multiscale (MCMS) series contextual model is able to leverage both the cross-object and the interobject contextual information at multiple scales to give a coherent segmentation of multiple objects present in an image. The rich contextual information that the MCMS model extracts from the image helps the later classifiers to correct the mistakes of the early stages and thus improves the overall performance.

Our model is motivated by the problem of reconstruction of the connectome, *i.e.,* the map of connectivity of all neurons in the mammalian nervous system [11], which is a challenge facing neuroscientists [7]. Electron microscopy (EM) is an image acquisition technique that can generate high resolution images with enough details for this problem [12]. However, the reconstruction of the connectome remains a challenging problem because of the noisy texture, irregular shapes, complex structures, and the large variations in the physical topologies of cells [18,19]. Moreover, the sheer size of a typical EM dataset, often approaching tens of terabytes [13], makes manual analysis infeasible [14]. Hence, automated segmentation methods are required.

General segmentation methods that have been proposed for natural image datasets yield poor results when applied to EM images [19]. Jain et al. [41] showed that multiscale normalized cut [42], boosted edge learning [43] and global probability boundary [44], which result in outstanding segmentation performance on natural images, perform poorly on EM datasets. Therefore, a powerful method for segmenting specific structures in EM images is required.

Many unsupervised techniques have been proposed to address this problem. Vu and Manjunath [45] proposed a graph-cut method that minimizes an energy function

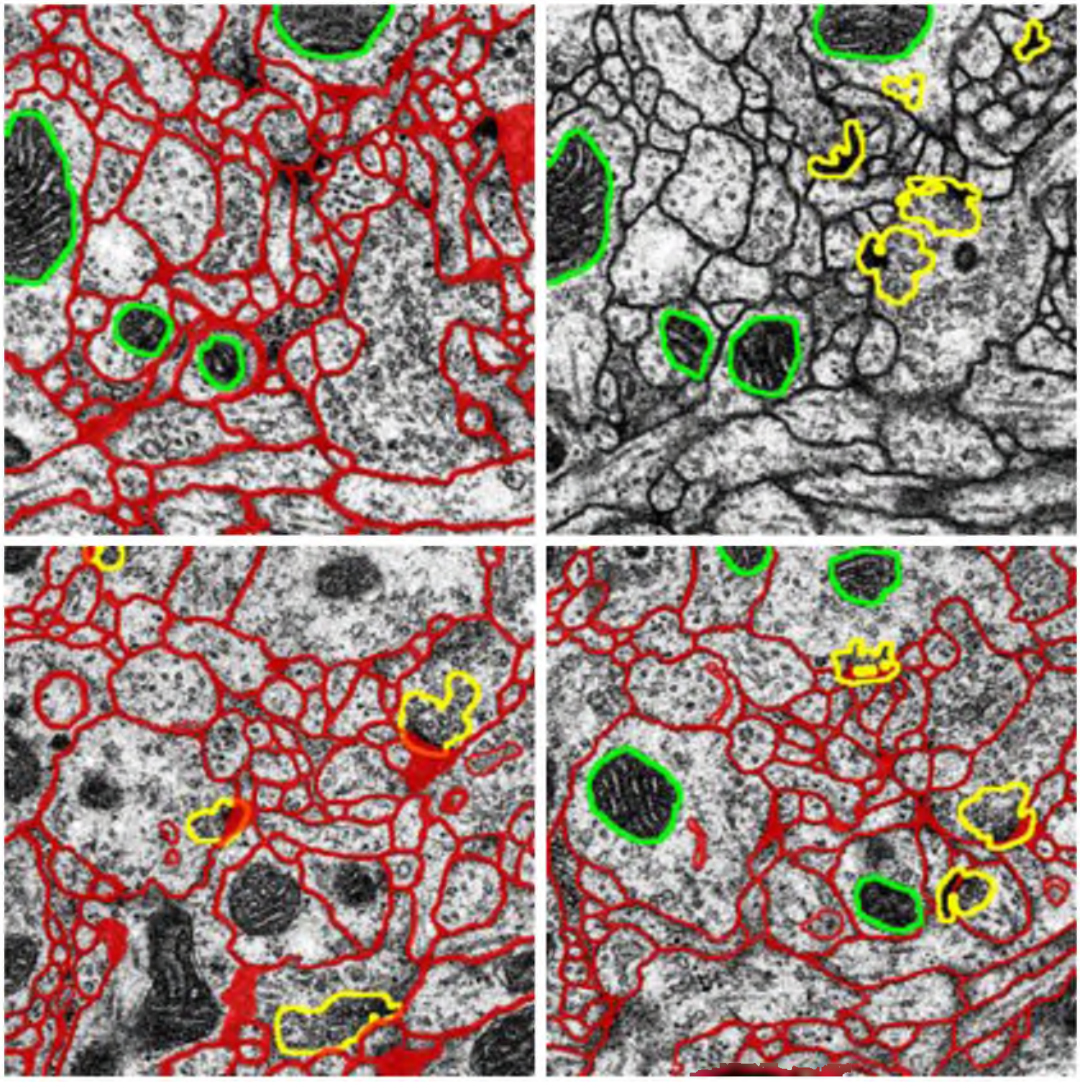


Figure 2.1: Different objects appear in certain configurations to each other. For example synapses, *i.e.,* objects with the yellow boundary, are close to the membrane, *i.e.,* red objects, and usually overlap with them. Mitochondria, *i.e.,* objects with the green boundary, are far from membranes and never overlap with synapses. Using this information can improve the segmentation results for each of these objects.

over the pixel intensity and flux of the gradient field for cell segmentation. However, their model might be confused by the complex intracellular structures and requires user interaction to correct segmentation errors. The contour propagation model [46] that minimizes an energy function for contour tracing of cell membranes can also get stuck in local minima due to complex intracellular structures. Kumar *et al.* [47] introduced a set of so-called Radon-like features (RLF), which take into account both texture and geometric information and overcome the problem of complex intracellular

structures but only achieve modest accuracy levels due to the lack of a supervised classification scheme.

Several supervised methods also have been proposed for object segmentation in EM images such as convolutional neural networks [18] and series of artificial neural networks (ANN) [7] for membrane detection or [19,22] for mitochondria segmentation or [48,49] for synapse segmentation. However, these frameworks target only one object of interest and to our knowledge, they do not use intraclass information to give a coherent segmentation of multiple objects. One of the advantages of our proposed model is that it can segment multiple objects simultaneously. We show that the coherent segmentation improves the segmentation accuracy.

2.2 Multiscale Contextual Model

Let $X = (x(i,j))$ be the input image that comes with a ground truth $Y = (y(i,j))$ where $y(i,j) \in \{-1, 1\}$ is the class label for pixel (i,j). The training set is $T = \{(X_k, Y_k); k = 1, \ldots, M\}$ where M denotes the number of training images. Given an input image X, the MAP estimation of Y for each pixel is given by

$$\hat{y}_{MAP}(i,j) = \arg\max_{y(i,j)} P(y(i,j)|X) \tag{2.1}$$

The local Markovianity assumption can be used to obtain a typical approximation of (2.1)

$$\hat{y}_{MAP}(i,j) = \arg\max_{y(i,j)} P(y(i,j)|X_{N(i,j)}) \tag{2.2}$$

where $N(i,j)$ denotes all the pixels in the neighborhood of pixel (i,j). $N(i,j)$ can be any arbitrary neighborhood lattice like 4-connected or 8-connected or sparse stencil [7] neighbors. This approximation decreases the computational complexity by giving the classifier access to a limited number of neighborhood pixels instead of the entire input image.

In auto-context [3] and series-ANN [7], a classifier is trained based on the neighborhood features at each pixel. We call the output image of this classifier the context image, *i.e.*, $C = (c(i,j))$. The next classifier is trained not only on the neighborhood

features of X but also on the neighborhood features of C. The MAP estimation formula for this classifier can be written as

$$\hat{y}_{MAP}(i,j) = \arg\max_{y(i,j)} P(y(i,j)|X_{N(i,j)}, C_{N'(i,j)}) \tag{2.3}$$

where $N'(i,j)$ is the set of all neighborhood pixels of pixel (i,j) in the context image. Note that N and N' can be different neighborhood systems. The same procedure is repeated through several stages of the series classifier until convergence. It is worth mentioning that (2.3) is closely related to the CRF model; however, multiple models in series are learned, which is an important difference from standard CRF approaches. It has been previously shown that this approach outperforms iterations with the same model [3].

According to (2.3), context provides prior information to solve the MAP problem. Even though the local Markovianity assumption is reasonable and makes the problem tractable, it still results in a significant loss of information from global context. However, it is not practical to sample every pixel in a very large neighborhood area of the context due to computational complexity problem and overfitting. Previous approaches [3, 7] have used a sparse sampling approach to cover large context areas. However, single pixel contextual information in the finest scale conveys only partial information about its neighborhood pixels in a sparse sampling strategy, while each pixel in the coarser scales contains more information about its surrounding area due to averaging filters used. In other words, while it is reasonable to sample context at the finest level a few pixels away, sampling context at the finest scale tens to hundreds of pixels away is error prone and presents a nonoptimal summary of its local area. Conceptually, sampling from scale space representation increases the effective size of the neighborhood while keeping the number of samples small.

Fig. 2.2 illustrates the multiscale contextual model. In this model, a scale-space representation of the context image is created by applying a series of Gaussian filters. This results in a series feature maps with lower resolutions that are robust against the small variations in the location of features as well as noise. Unlike the auto-context structure that uses a sparse sampling approach to take samples from the context image, the multiscale contextual model uses the samples of the scale space

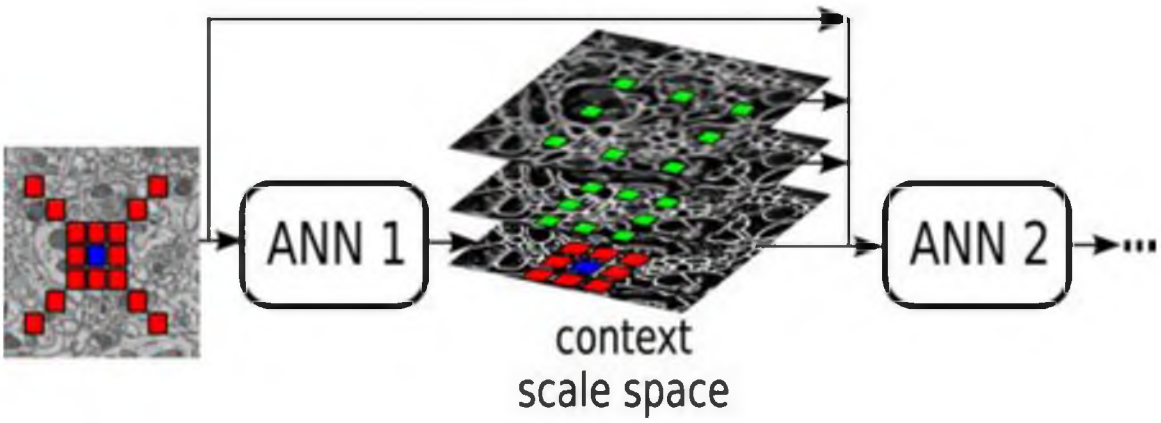


Figure 2.2: Illustration of the multiscale contextual model. Each context image is sampled at different scales (green squares). The blue squares represent the center pixel, and the red squares show the selected locations at original scale.

representation of context. Fig. 2.3 shows the single-scale sampling strategy (Fig. 2.3a) versus the multiscale sampling strategy (Fig. 2.3b). In Fig. 2.3b the classifier can have as an input the center 3×3 patch at the original scale and a summary of eight surrounding 3×3 patches at a coarser scale (The green circles denote the summaries of dashed squares). The green circles in Fig. 2.3b are more informative and less noisy compared to their equivalent red circles in Fig. 2.3a. The summaries become

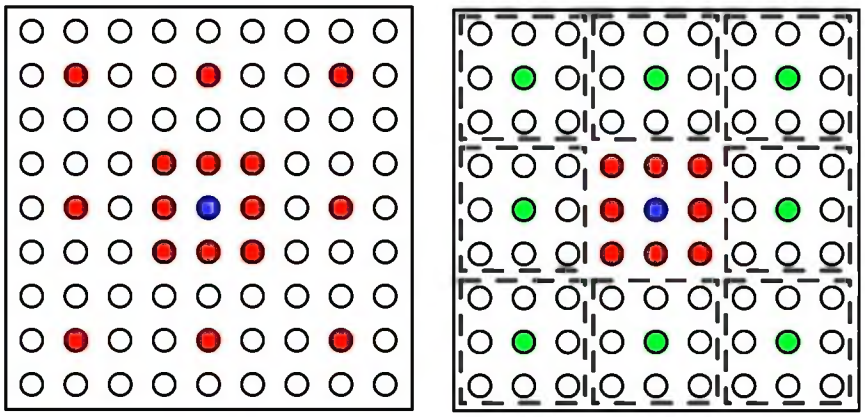


Figure 2.3: Sampling strategy of context: (a) Sampling at a single scale, (b) sampling at multiple scales. Green circles belong to a coarser scale and illustrate the summary of pixels in dashed squares. Green samples at the coarser scale are more informative than corresponding red samples at the original scale.

more informative as the number of scales increases. For example, in the second scale the summary is computed over 3×3 neighborhood of the first scale image, which is equivalent to 5×5 neighborhood of the original image. In practice, we use Gaussian averaging filters to create the summary (green circles). Other methods like maximum pooling can be used instead of Gaussian averaging [50]. The number of scales and Gaussian filter size are set according to the characteristics of the particular application. The size of the filter and number of scales should increase for larger objects.

From a mathematical point of view, (2.3) can be rewritten as

$$\hat{y}_{MAP}(i,j) = \arg\max_{y(i,j)} P(y(i,j)|X_{N(i,j)}, C_{N_0'(i,j)}(0), \\ C_{N_1'(i,j)}(1), \ldots, C_{N_l'(i,j)}(l)) \tag{2.4}$$

where $C(0), C(1), \ldots, C(l)$ denote the scale space representation of the context and $N_0'(i,j)$, $N_1'(i,j), \ldots, N_l'(i,j)$ are corresponding neighborhood structures. Unlike (2.3) that uses the context in a single scale, (2.4) takes the advantage of multiscale contextual information. Even though in (2.4) we still use the Markov assumption, the size of the neighborhood is larger and thus we lose less information compared to (2.3).

The series multiscale contextual model updates the (2.4) iteratively:

$$\hat{y}_{MAP}^{k+1}(i,j) = \arg\max_{y(i,j)} P(y(i,j)|X_{N(i,j)}, C^k_{N_0'(i,j)}(0), \\ C^k_{N_1'(i,j)}(1), \ldots, C^k_{N_l'(i,j)}(l)) \tag{2.5}$$

where $C^k(0), C^k(1), \ldots, C^k(l)$ are the scale space representation of the output of classifier stage k, $k = 1, \ldots, K-1$ and $\hat{y}_{MAP}^{k+1}(i,j)$ denotes the output of the stage $k+1$. In turn, the $k+1$'st classifier output as defined in (2.5) creates the context for the $k+2$'nd classifier. For $k = 0$, no prior information is used, and the model only uses the input image for training. The model repeats (2.5) until the performance improvement between two consecutive stages becomes small. It must be emphasized that despite the iterative form of 2.5, multiple models are learned in the series separately and in sequential order, which is an important difference from standard CRF models.

2.3 Multiclass Multiscale Contextual Model

While our multiscale contextual model extracts a set of rich features from the context image of each object, it is unable to take into account the contextual information from multiple objects. We propose the multiclass multiscale (MCMS) contextual model as a remedy to this problem as it is designed to leverage both the multiscale and the multiclass contextual information. The proposed method can successfully capture long distance dependencies between objects and across different categories.

The multiclass contextual model is illustrated in Fig. 2.4. In this figure, each classifier is a binary classifier, which is trained to segment only one object of interest. In other words, each classifier treats the pixels belonging to the object of interest as positive samples and all the other pixels including the background pixels as negative samples. The multiclass architecture allows the classifier of each object type access to the contextual information from each object type of the previous stage. This flow of information is achieved by feeding neighborhoods from the output of each classifier, *i.e.,* the context image, in stage k to each classifier in stage $k+1$. The multiclass feature pooling scheme is shown in Fig. 2.5. It extracts samples from the neighborhood of center pixel in all the context images from the previous stage. The extracted samples are used together with input image samples as the input to classifier. The same feature vectors are used for all the classifiers. Nonetheless, each classifier is trained to segment a specific object. In other words, although the input feature vectors are the same, the target labels are different for each classifier. The propagation of contextual information among different categories enables the model to learn the geometrical relationships and object dependencies implicitly.

We describe the effectiveness of the multiclass model with a synthetic example. Consider the input image and the corresponding groundtruth images in Fig. 2.6a. Two pixel classifiers are trained for the square and the disk classes separately. The outputs of these classifiers are shown in Fig. 2.6b. The results are not perfect, and each classifier misclassifies some pixels of the other object as positive samples due to the noise and similarity between the textures. The single-class model that uses only the contextual information from the same object is not able to correct the wrong classified pixels completely (Fig. 2.6c). By using the contextual information from both

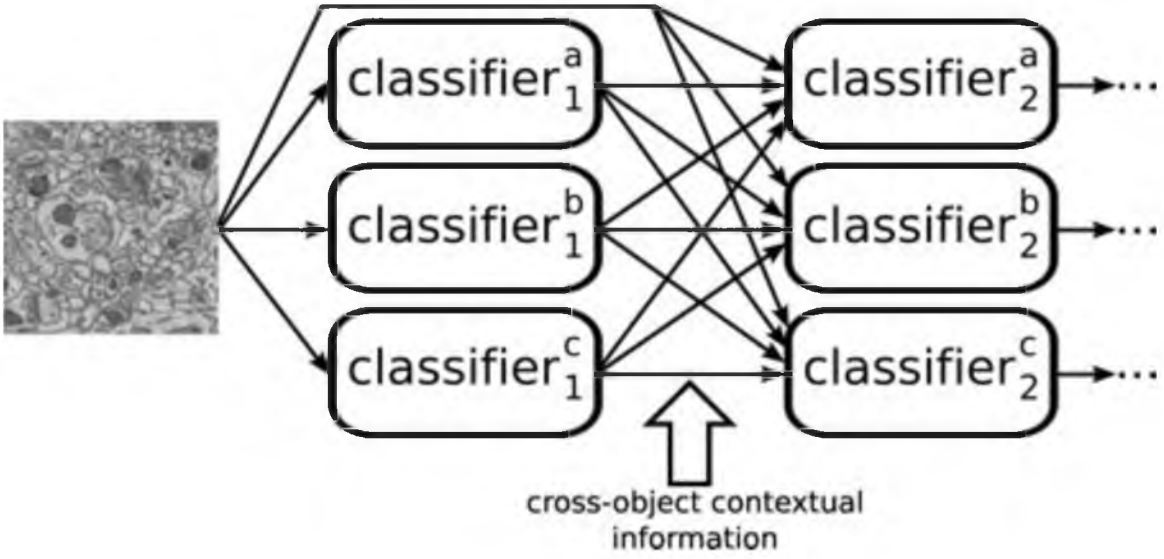


Figure 2.4: Illustration of the multiclass contextual model. Each classifier is a binary classifier that is trained for a specific object (a, b, and c are objects). Each classifier takes advantage of the context images of all objects from the previous stage. Superscripts show object type, and subscripts show the classifier number in the series. Generalization to cases with more classes is straightforward.

of the objects, the multiclass model will classify most of the previously misclassified pixels correctly as shown in Fig. 2.6d. For example, the second stage square classifier exploits the information that those misclassified pixels from the previous stage are classified as disk by the first disk classifier and thus is able to correct them in the second stage. In this example we have two objects but this can be extended to any arbitrary number of objects.

The mathematical formulation of the multiclass contextual model for each classifier is obtained by incorporating the cross-contextual information in (2.3):

$$\hat{y}_{MAP}(i,j) = \arg\max_{y(i,j)} P(y(i,j)|X_{N(i,j)}, C^a_{N'(i,j)}, C^b_{N'(i,j)}, C^c_{N'(i,j)}) \tag{2.6}$$

where C^a, C^b, C^c denote the context images of different objects. We assume three objects in (2.6) for the sake of simplicity, but the extension to more objects is straightforward.

By combining multiclass and multiscale contextual models, the powerful MCMS model is obtained, which is able to extract contextual information from large area and

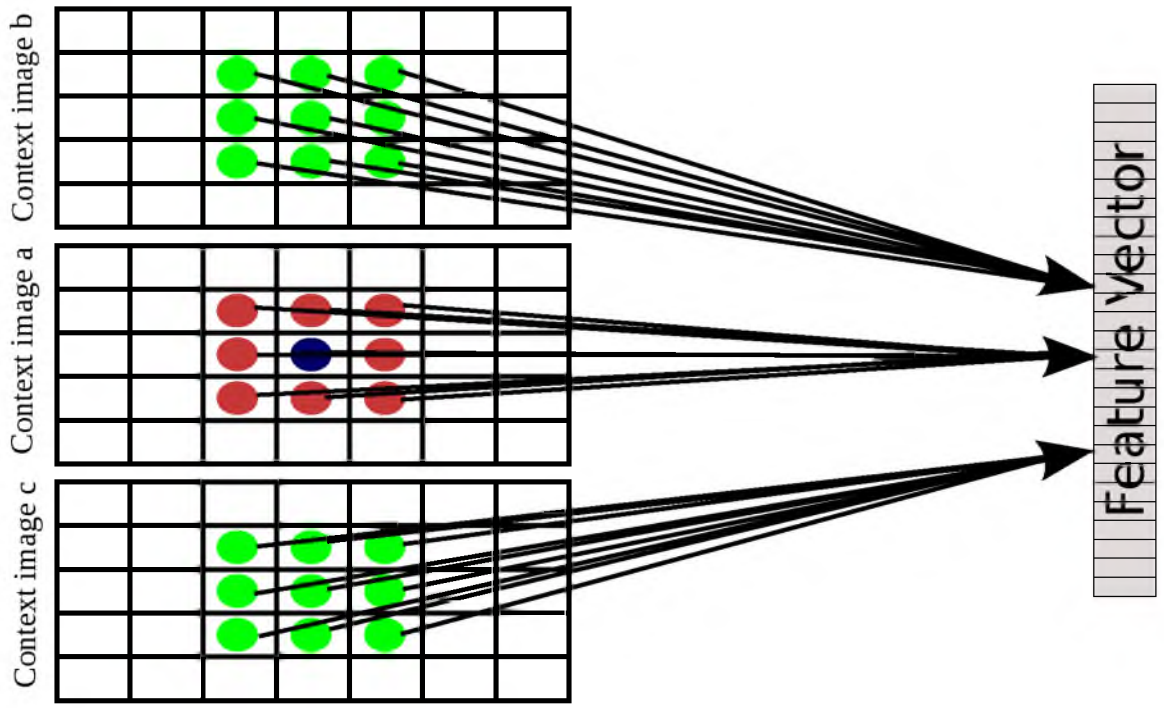


Figure 2.5: The multiclass feature pooling scheme. The neighborhood samples of the center pixel (blue circle) in the context image "a," *i.e.,* red circles, are used together with the neighborhood samples in the context images "b" and "c," *i.e.,* green circles, to form the feature vector. The same feature vector together with the features of input image is used for all the classifiers. In the MCMS model the samples are pooled at multiple scales as well. The multiscale sampling is not shown in this figure for the sake of clarity.

through different objects. The MCMS model is designed to make an extensive use of contextual information. This architecture allows the classifiers in the series to correct the errors of the previous stages by using the information from other classes and thus improves the segmentation performance. The update equation of the MCMS model can be derived by combining (2.3) and (2.4)

$$\begin{aligned}\hat{y}^{a,k+1}_{MAP}(i,j) = \arg\max_{y(i,j)} P(y(i,j)|X_{N(i,j)},\\ C^{a,k}_{N'_0(i,j)}(0), C^{b,k}_{N'_0(i,j)}(0), C^{c,k}_{N'_0(i,j)}(0),\\ C^{a,k}_{N'_1(i,j)}(1), C^{b,k}_{N'_1(i,j)}(1), C^{c,k}_{N'_1(i,j)}(1),\\ \ldots, C^{a,k}_{N'_l(i,j)}(l), C^{b,k}_{N'_l(i,j)}(l), , C^{c,k}_{N'_l(i,j)}(l))\end{aligned} \tag{2.7}$$

where $C^{a,k}(0), C^{a,k}(1), \ldots, C^{a,k}(l)$ are the scale space representation of the output of

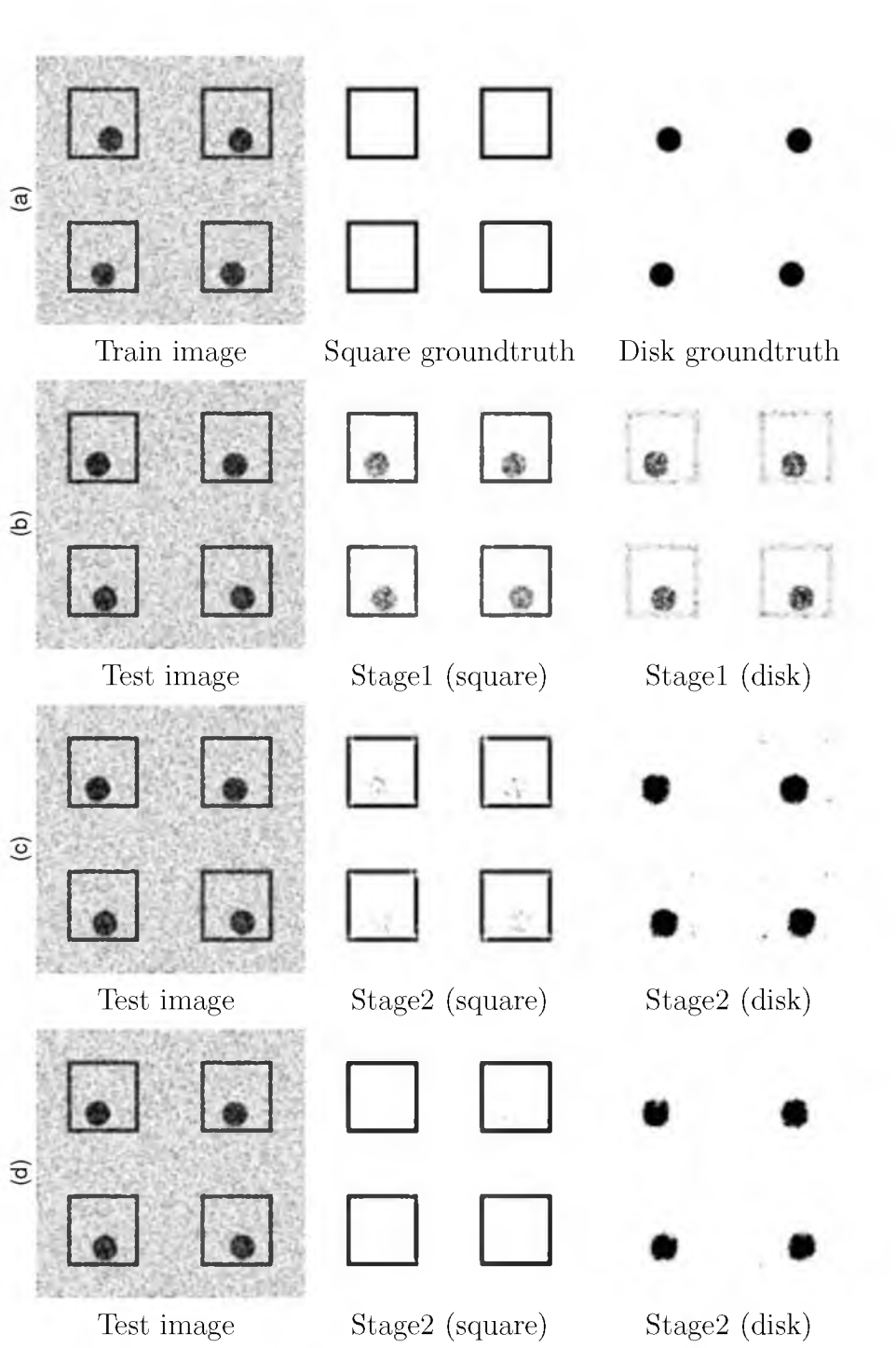


Figure 2.6: A synthetic example that shows the effectiveness of the multiclass contextual model. (a) The input image and corresponding groundtruth images, (b) the outputs of the first stage classifiers, (c) the outputs of the second stage classifiers in the single-class model, and (d) the outputs of the second stage classifiers in the multiclass model. The multiclass model is more successful in removing the parts of the other object compared to the single-class model.

classifier stage k for object "a," $k = 1, \ldots, K-1$ and $\tilde{y}_{MAP}^{a,k+1}(i,j)$ denotes the output of the stage $k+1$ for object "a." Similar equations are updated for objects "b" and "c." Each of these update equations are related to a row of classifiers in Fig. 2.4. The main difference between (2.5) and (2.7) is that the former only pools contextual information from a single object, while the latter takes advantage of contextual information from multiple objects. The overall training algorithm for the MCMS contextual model is described in Algorithm 1.

Algorithm 1 Training algorithm for the MCMS model

Require: A set of training images together with their binary groundtruth images for different objects, $T = \{(X_i, Y_i^s), i = 1, \ldots, M, s = 1, \ldots, N_{obj}\}$.

- For each input image X_i, generate non-informative probability maps, $C_i^{s,0}, s = 1, \ldots, N_{obj}$, with uniform distribution.
- $k = 0$

repeat

for $j = 1 : N_{obj}$ **do**

- Construct a new training set $T_j = \{((X_i, C_i^{s,k}), Y_i^j), i = 1, \ldots, M, s = 1, \ldots, N_{obj}\}$.
- Train a classifier, f_k^j, on features extracted from the input images and scale space representation of the context images (maximize equation (2.7) to obtain classifier parameters).

end for

for $j = 1 : N_{obj}$ **do**

- Use the trained classifier f_k^j to generate new context images $C_i^{j,k+1}$ (equation (2.7)).

end for

- $k = k + 1$

until convergence (improvement is negligible between two consecutive stages)

The time complexity of the MCMS model is almost the same as the multiscale since the classifiers of each stage can be trained in parallel. Although this model has many parameters, the training is not complicated because the classifiers are trained separately through the stages and among the objects.

2.4 Experimental Results

We perform experimental studies to evaluate the performance of both multiscale and MCMS contextual models. We show the effectiveness of the multiscale contextual model for membrane detection in EM images and horse segmentation in a general computer vision dataset. We then show how membrane detection results can be used in the MCMS model to improve mitochondria and synapse segmentation results.

2.4.1 Datasets

We used three different datasets in our experiments:

2.4.1.1 Weizmann horse dataset

The Weizmann dataset [9] contains 328 gray scale horse images with corresponding foreground/ background truth maps. Similar to Tu *et al.* [3], we used half of the images for training, and the remaining images were used for testing. There is only one object category, *i.e.,* horse, in this dataset and thus we could only use it to test the multiscale contextual model.

2.4.1.2 Mouse neuropil dataset

This dataset is a stack of 400 images from the mouse neuropil acquired using serial block face scanning electron microscopy (SBFSEM [12]). Each image is 4096 by 4096 pixels, and the resolution is $10 \times 10 \times 50$ nm/pixel. To evaluate the segmentation performance, a subset of 70 images of size 700 by 700 pixels were selected. An expert anatomist annotated membranes and mitochondria in this subset with different labels. From those 70 images, 14 images were randomly selected and used for training, and the 56 remaining images were used for testing.

2.4.1.3 Drosophila VNC dataset

This dataset contains 30 images from Drosophila first instar larva ventral nerve cord (VNC) [51, 52] acquired using serial-section transmission electron microscopy (ssTEM [17, 53]). It has a resolution of $4 \times 4 \times 50$ nm/pixel, and each $2D$ section is 512 by 512 pixels. For this dataset, an expert annotated membranes, mitochondria, and synapses with different labels. We used 15 images for training and 15 images for testing.

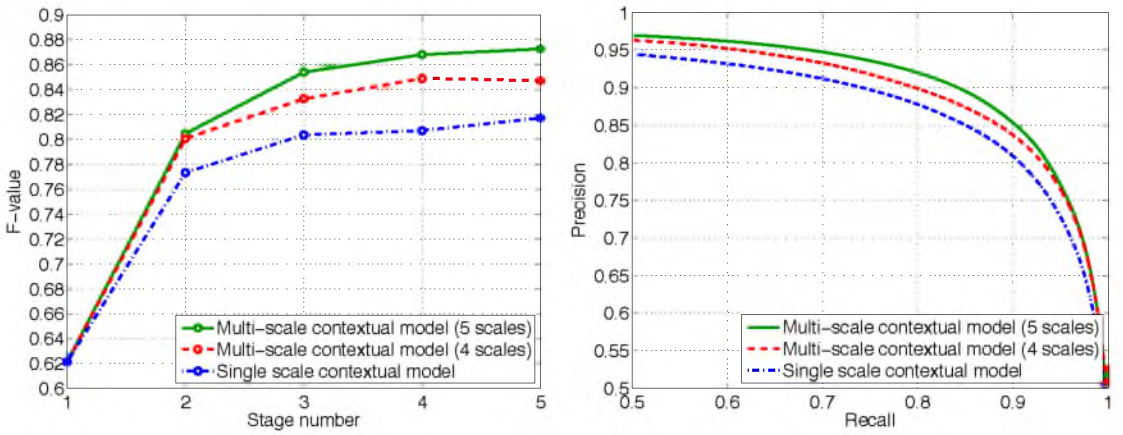
The results presented in this chapter were generated using a $HPDL980$ server containing 160 2.40 GHz Intel CPUs and 750G of memory. The horse dataset requires 19G of memory during training, while the mouse neuropil and Drosophila VNC datasets require 13G and 14G of memory, respectively. It took about 6, 5, and 3 days per stage to train the multiscale contextual model on the horse, mouse neuropil, and Drosophila VNC datasets, respectively. As mentioned before, the training time of the MCMS model is almost the same as the multiscale contextual model. Unlike the training, our model is relatively fast at the test time. Applying the classifiers weights on each input image takes less than 1 minute. Details regarding the parameters for each experiment are described in detail in the following sections.

2.4.2 Multiscale contextual model (horse segmentation)

In this experiment, we test the multiscale contextual model for horse segmentation. We used MLP-ANNs [54,55] as the classifier in the series architecture, as in [7]. Each classifier in the series has one hidden layer with 30 nodes. Back-propagation was used to learn the weight vector and biases [54, 55].

Input image feature vectors were computed on a 31×31 sparse stencil [7] centered on each pixel. The size of the feature vector is 57. The context features were computed using 5×5 patches at five scales (one at original resolution and four at coarser scales). We used a Gaussian filter of size 7×7 to generate the scale space.

The average $F - value = \frac{2 \times Precision \times Recall}{Precision + Recall}$ at threshold 0.5 for different methods is shown in Fig. 2.7(a). As we expected, the performance increases with the number of scales. The test F-value at stage 5 for multiscale contextual model with 5 scales is 87.3%. This result outperforms the auto-context result which is 84% [3]. It must be emphasized that the improvement from the first stage to the last stage in our method is 25.2%, while the improvement in the auto-context method is almost 5%. It is worth noting that we use a simple stencil to generate the input image feature vector instead of applying large filter banks to the input image as in [3], and our first stage F-value (62.1%) is less than auto-context first stage F-value (79%), but our last stage result F-value is higher. This shows that multiscale contextual model can compensate for the bad result of the first stage and improves the performance in later stages by using context in an effective manner. The precision-recall curves of the last stage results



(a) (b)

Figure 2.7: Horse segmentation experiment on the Weizmann horse dataset. (a) The test F-value at different stages of the series for different methods with different number of scales. (b) The precision-recall curves for test images and for different methods (the last stage of the series). Using more scales improves the results.

for the test set are shown in Fig. 2.7(b).

Fig. 2.8 shows some examples of our test images and their segmentation results using different methods with different number of scales. As we can see, the multiscale contextual model outperforms the single-scale contextual model in removing the side effects of the cluttered background and filling the body of horses. For example, in the middle column, the rider is removed by the multiscale contextual model with 5 scales. Fig. 2.9 shows two examples of test images and the corresponding segmentation results at different stages of the multiscale contextual model. The converges of the model can be seen qualitatively in the results.

2.4.3 Multiscale contextual model (membrane detection)

In this experiment, we show the performance of the multiscale contextual model for membrane detection on the mouse neuropil dataset. We used the same architecture as the previous experiment except that each MLP-ANN in the series had one hidden layer with 10 nodes.

This dataset is very imbalanced since the number of positive samples, *i.e.,* membrane pixels, is much less than the negative samples, *i.e.,* nonmembrane pixels. To provide a relatively balanced dataset and optimize the MLP-ANN performance, 5.5

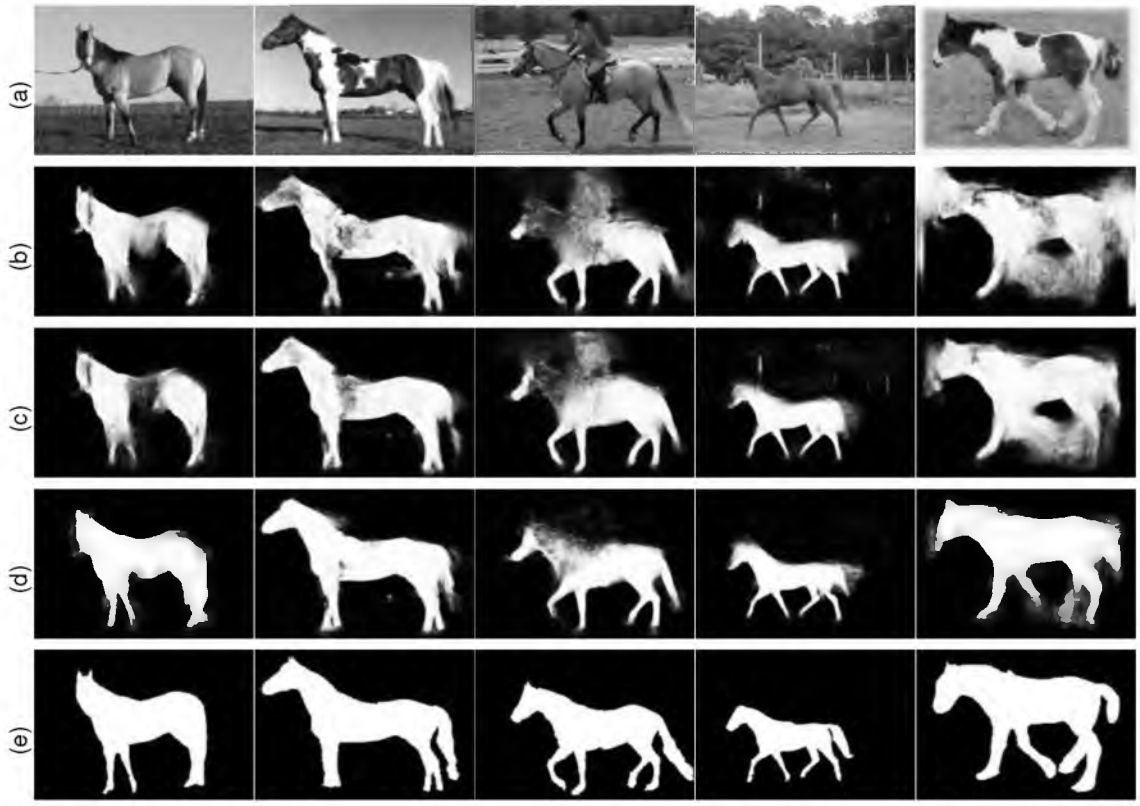


Figure 2.8: Test results for the horse segmentation experiment. (a) Input images, (b) single-scale contextual model [7], (c) multiscale contextual model with 4 scales, (d) multiscale contextual model with 5 scales, (e) groundtruth images. The multiscale contextual model is successful in removing the side effects of the cluttered background and filling the body of horses.

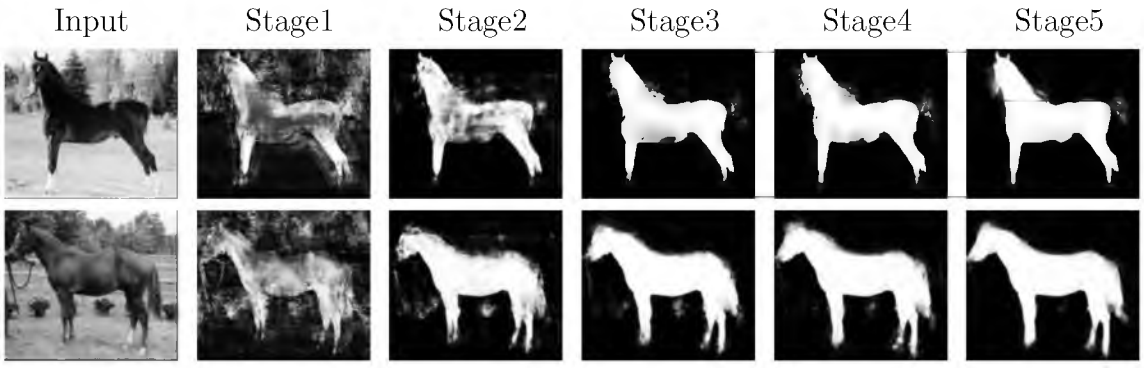
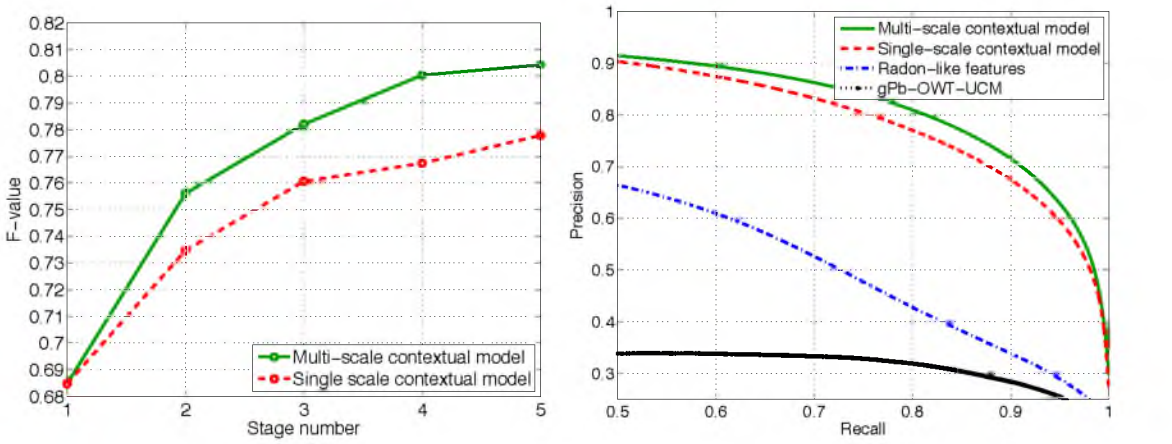


Figure 2.9: Test results for the horse segmentation experiment. The first column shows the input image and the remaining columns show the output at different stages of multiscale contextual model.

million samples were randomly selected from the training set to contain $\frac{1}{3}$ positive and $\frac{2}{3}$ negative examples, as in [7]. Input image feature vectors were computed on a 11×11 stencil. Context features were computed on 5×5 patches at four scales (one at original resolution and three at coarser scales). The classifier then gets as input the 5×5 patch at the original resolution ($C_{N'_0(i,j)}(0)$ in (2.4)) and 5×5 patches at three coarser scales ($C_{N'_l(i,j)}(l)$ in (2.4)). We used a Gaussian filter of size 5×5 to generate the scale space.

We compared the performance of our methods with the RLF [47] and gPb-OWT-UCM (global probability of boundary followed by the oriented watershed transform and ultrametric contour maps) [44]. The average F-value for different stages of multi-scale contextual and MCMS models is shown in Fig. 2.10(a). The performance of the multiscale contextual model is 2.65% better than using a single-scale context [7]. The precision-recall curves for pixel-wise membrane detection are shown in Fig. 2.10(b).

Fig. 2.11 shows five examples of our test images and corresponding membrane detection results for different methods. As shown in our results, the multiscale contextual model outperforms the methods in [7, 44, 47], and it is more successful in removing undesired parts from inside cells.



(a) (b)

Figure 2.10: Membrane detection experiment on the mouse neuropil dataset. (a) The test F-value at different stages of the series for different methods. The F-value for the RLF and gPb-OWT-UCM methods are 0.59 and 0.46, respectively. (b) The precision-recall curves for test images and for different methods (the last stage of the series).

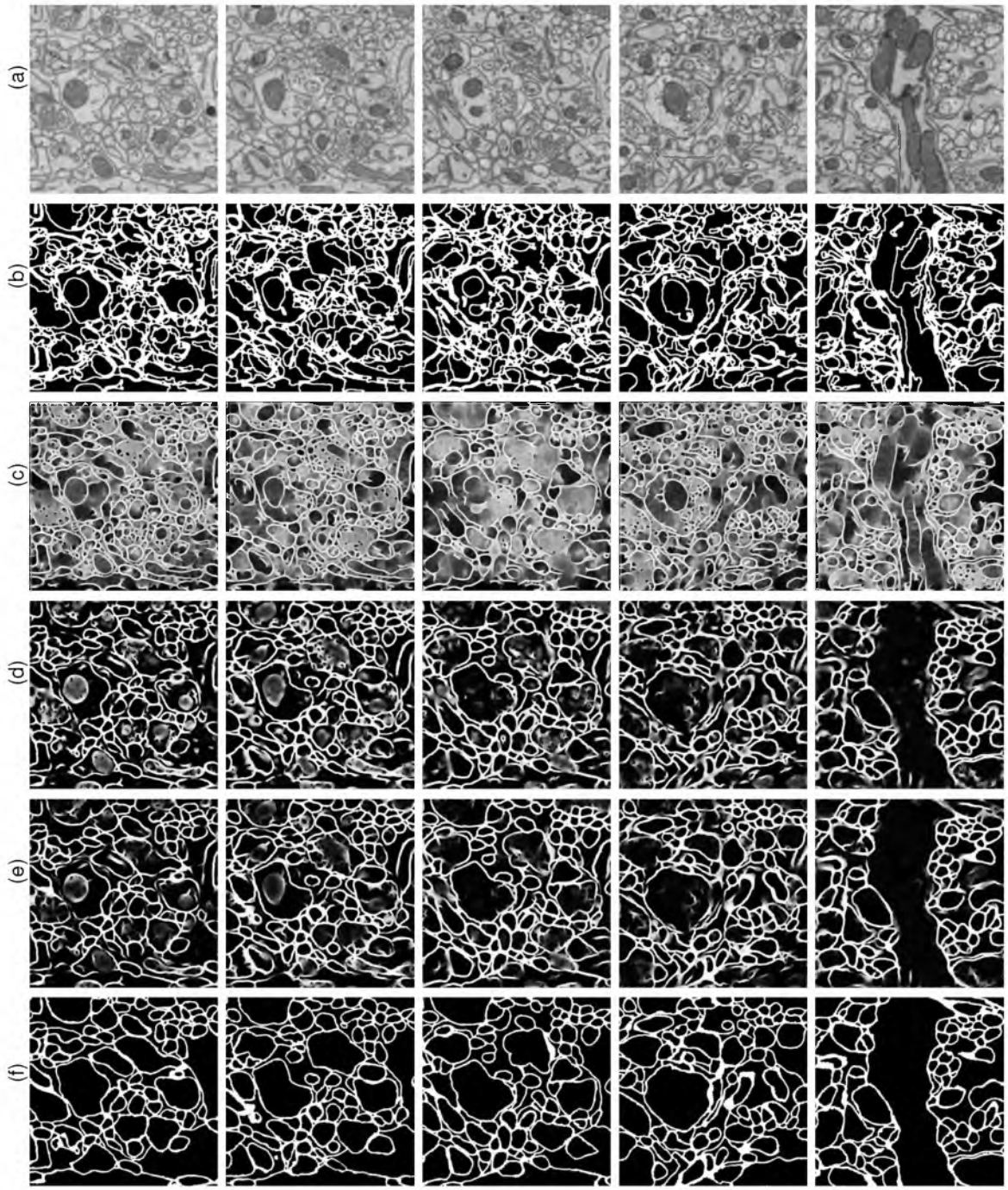
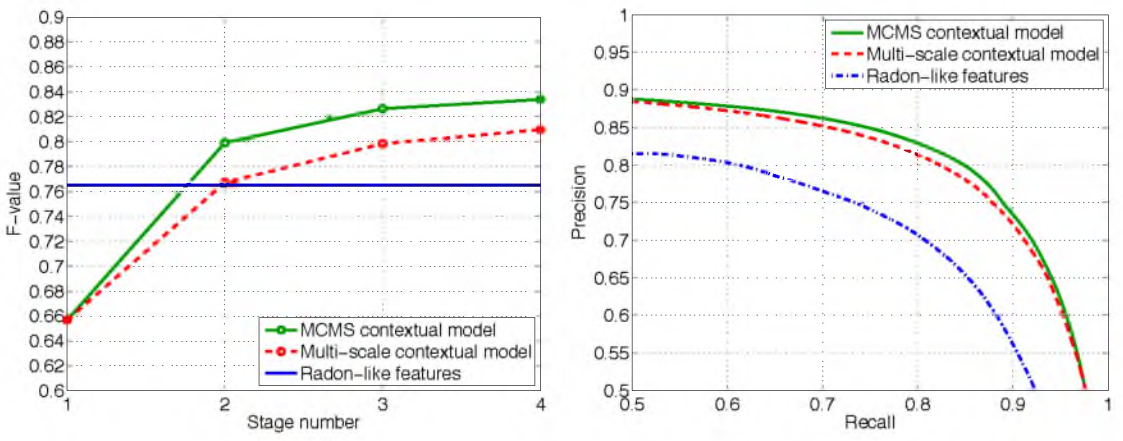


Figure 2.11: Test results for the membrane detection experiment (mouse neuropil dataset). (a) Input images, (b) gPb-OWT-UCM method [44], (c) RLF method [47], (d) single-scale contextual model [7], (e) multiscale contextual model, (f) groundtruth images. The multiscale contextual model is more successful in removing undesired parts from inside cells than the algorithms proposed in [7, 44, 47]. For gPb-OWT-UCM method, the best threshold was picked, and the edges were dilated to the true membrane thickness.

2.4.4 MCMS contextual model (mitochondria segmentation)

In this section, we show that MCMS model outperforms the multiscale contextual model in mitochondria segmentation for the mouse neuropil dataset. For this dataset, the labels are only available for membrane and mitochondria, so $N_{obj} = 2$ in Algorithm 1. We used MLP-ANNs with 10 hidden nodes for both membrane and mitochondria classifiers.

Input image feature vectors were computed on 11×11 and 15×15 stencils for membrane and mitochondria classifiers, respectively. For both of the categories, the context features were computed on 5×5 patches at four scales. To compare the performance, we used the same mitochondria classifiers with the same parameter settings in the multiscale contextual model. The average F-value at different stages and for different methods is shown in Fig. 2.12(a). The performance of the MCMS model is 2.42% better than the multiscale contextual model. The precision-recall curves for pixel-wise mitochondria segmentation are shown in Fig. 2.12(b). Fig. 2.13 shows five test examples and corresponding mitochondria segmentation results for different methods. The MCMS model is more successful in correcting both false positive and false negative errors compared to the multiscale contextual and RLF models.



(a) (b)

Figure 2.12: Mitochondria segmentation experiment on the mouse neuropil dataset. (a) The test F-value at different stages of the series for different methods. (b) The precision-recall curves for test images and for different methods (the last stage of the series).

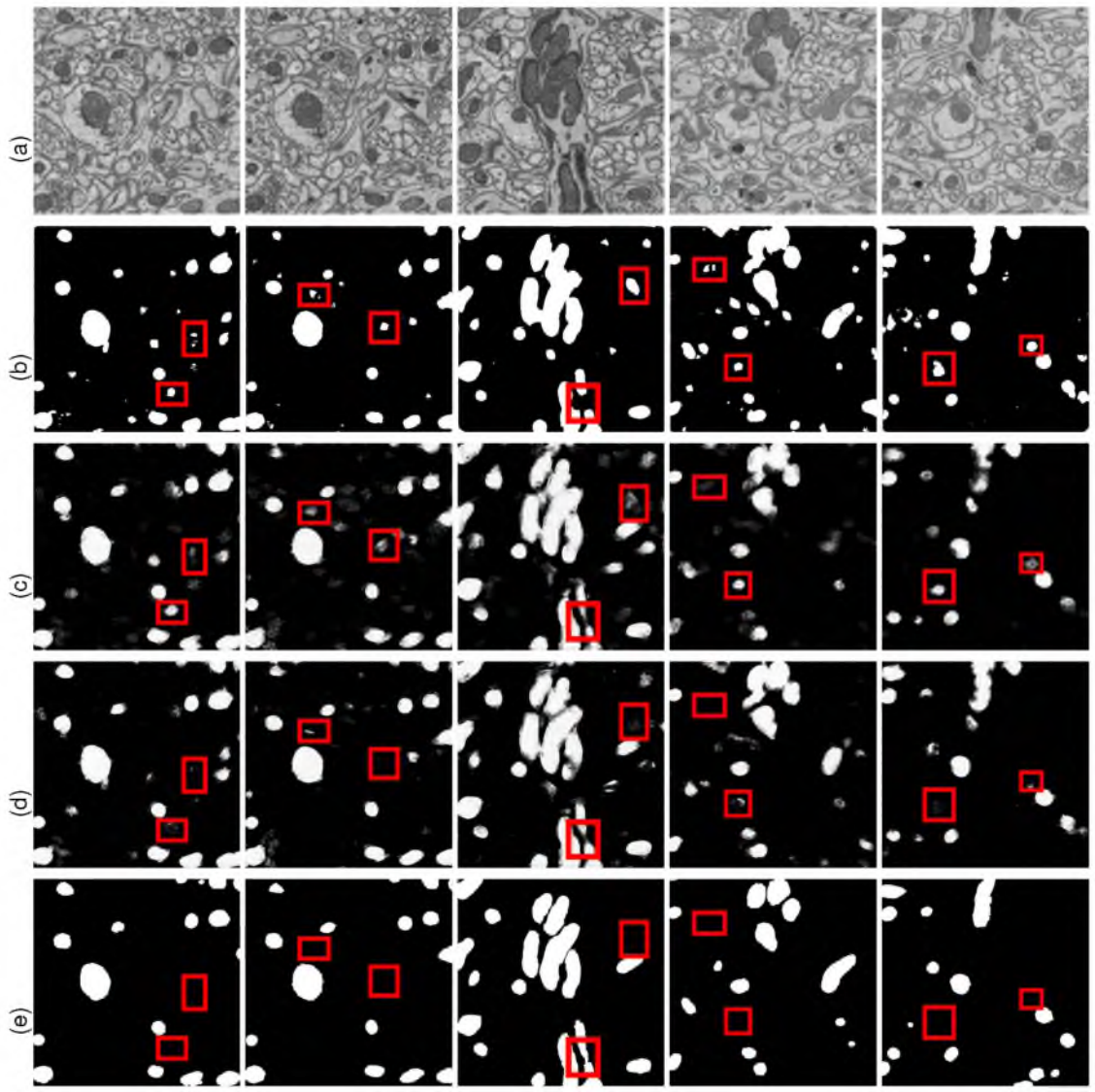


Figure 2.13: Test results for the mitochondria segmentation experiment (mouse neuropil dataset). (a) Input images, (b) RLF method [47], (c) multiscale contextual model, (d) MCMS contextual model, (e) groundtruth images. The MCMS contextual model is more successful in correcting both false positive and false negative errors compared to other methods. Some of the improvements are marked with red rectangles.

2.4.5 MCMS contextual model (mitochondria and synapse segmentation)

In this experiment, we test the MCMS model performance on the Drosophila VNC dataset with three object categories: membrane, mitochondria, and synapse. We used MLP-ANNs with 10 hidden nodes as classifier in the series.

Input image features were computed on 11×11, 15×15, and 15×15 for membrane, mitochondria, and synapse classifiers, respectively. Similar to previous experiments, context features were computed on 5×5 patches at four scales. To compare with the multiscale contextual model, we used classifiers with the same parameter settings for mitochondria and synapse segmentation. Fig. 2.14 shows five test samples and corresponding mitochondria segmentation results for different methods. The MCMS model gives cleaner results compared to other methods. Fig. 2.15 shows synapse segmentation results for five test samples. The MCMS model is more successful in correcting false positive errors compared to the multiscale contextual model. It must be emphasized that in this experiment we target four elements of synapses, *i.e.,* synapstic cleft, postsynaptic density, T-band, and vesicles, simultaneously, which is a challenging task even for expert anatomists. That explains why the results are not as good as the membrane and mitochondria segmentation results.

The average F-value for the test set at different stages is shown in Fig. 2.16. The MCMS model outperforms the multiscale contextual model with 2.9% and 2.92% in mitochondria and synapse segmentation, respectively. The F-value of RLF method for mitochondria segmentation is 60%, which is about 7% worse than the MCMS model.

2.4.6 Results discussion

In all of the above experiments, our goal was to study the effect of using rich contextual information in segmentation performance. We only used the samples of input images on a stencil structure as input image features. The overall performance can be improved by applying filter banks to input images and extract more informative features like what Tu *et al.* [3] did for horse segmentation. We previously showed [23] extracting Radon-like features from input images can improve the membrane detection results.

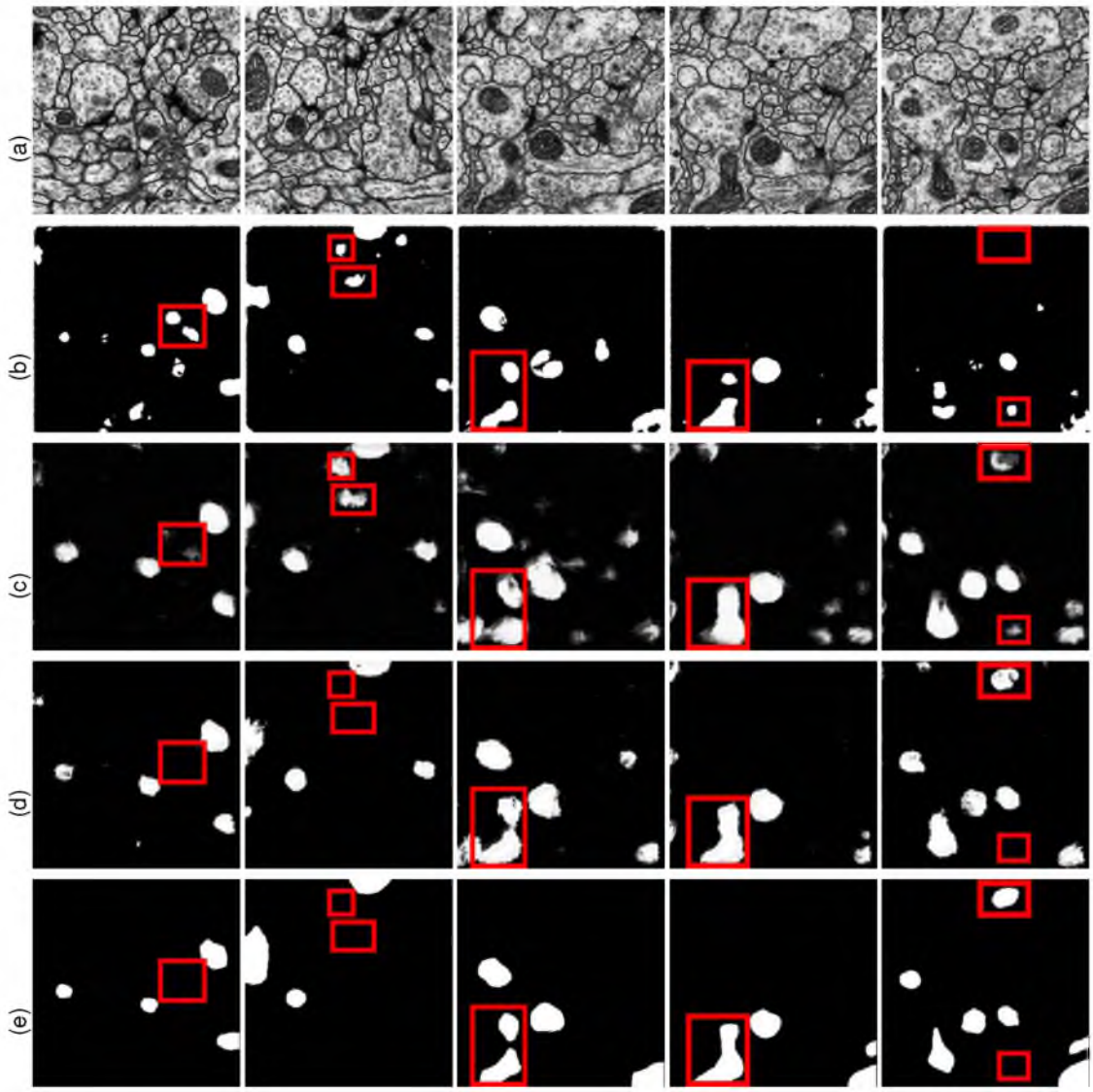


Figure 2.14: Test results for the mitochondria segmentation experiment (Drosophila VNC dataset). (a) Input images, (b) RLF method [47], (c) multiscale contextual model, (d) MCMS contextual model, (e) groundtruth images. The MCMS contextual model gives cleaner results compared to other methods. Some of the improvements are marked with red rectangles.

We noticed that in the MCMS model if a dataset is highly imbalanced, then the effect of small classes on big classes is negligible. For example, the mitochondria contextual information in section 2.4.4 and the synapse and mitochondria contextual information in section 2.4.5 did not improve the membrane detection results. Nonetheless, big classes or same-size classes can improve the segmentation results of small classes as we showed in the experiments. In the mouse neuropil dataset

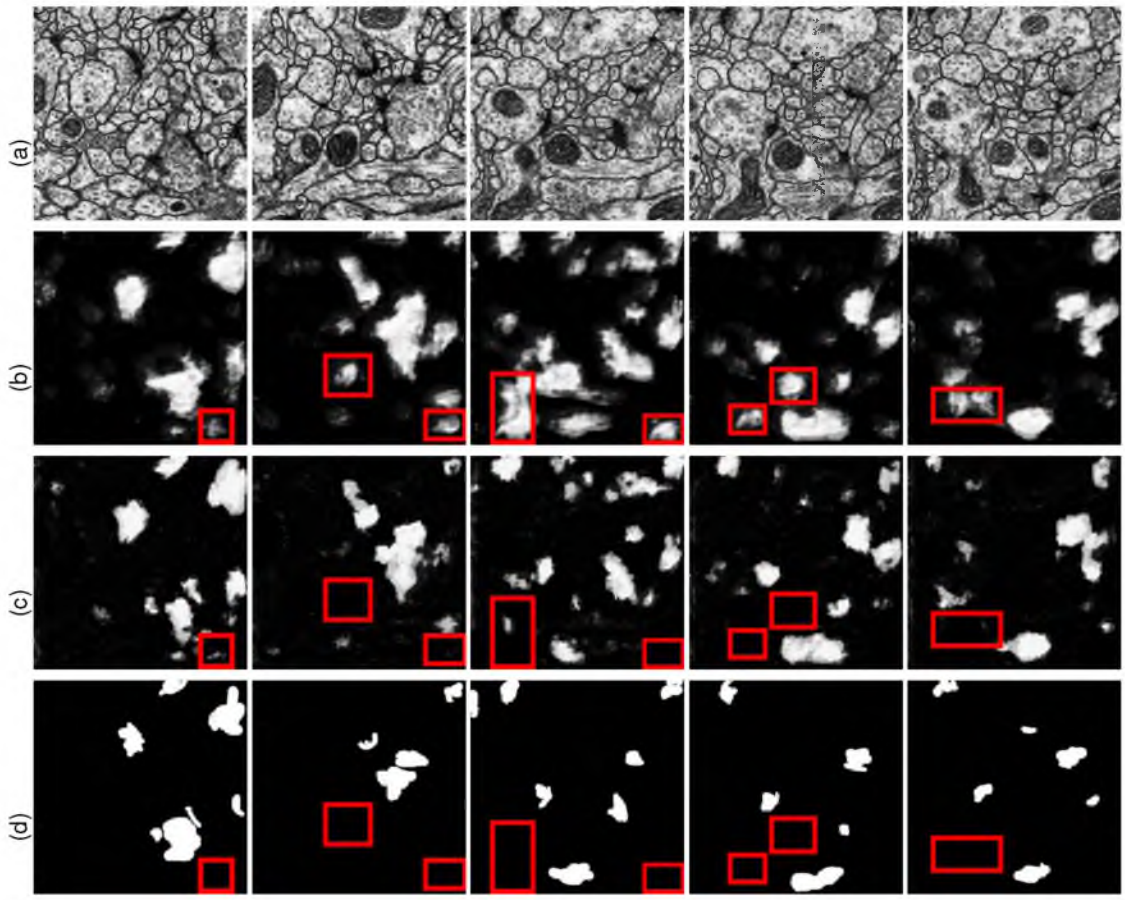


Figure 2.15: Test results for the synapse segmentation experiment (Drosophila VNC dataset). (a) Input images, (b) multiscale contextual model, (c) MCMS contextual model, (d) groundtruth images. The MCMS contextual model is more successful in correcting false-positives errors than the multiscale contextual model. Some of the improvements are marked with red rectangles.

the mitochondria class is 2.5 times smaller than the membrane class, and in the Drosophila VNC dataset the mitochondria and synapse classes are 4.5 and 6 times smaller than the membrane class, respectively.

In general image segmentation applications, other powerful techniques such as graph cuts and level sets can be applied to the results of the MCMS model to improve the segmentation accuracy. In segmentation of EM images, the final segmentation results can be improved further by applying appropriate postprocessing techniques. For example, Andres *et al.* [56] propose a hierarchical method that uses oversegmented images obtained from membrane detection results and applies a classifier to merge regions. Funke *et al.* [57] and Liu *et al.* [58] use a tree structure to merge

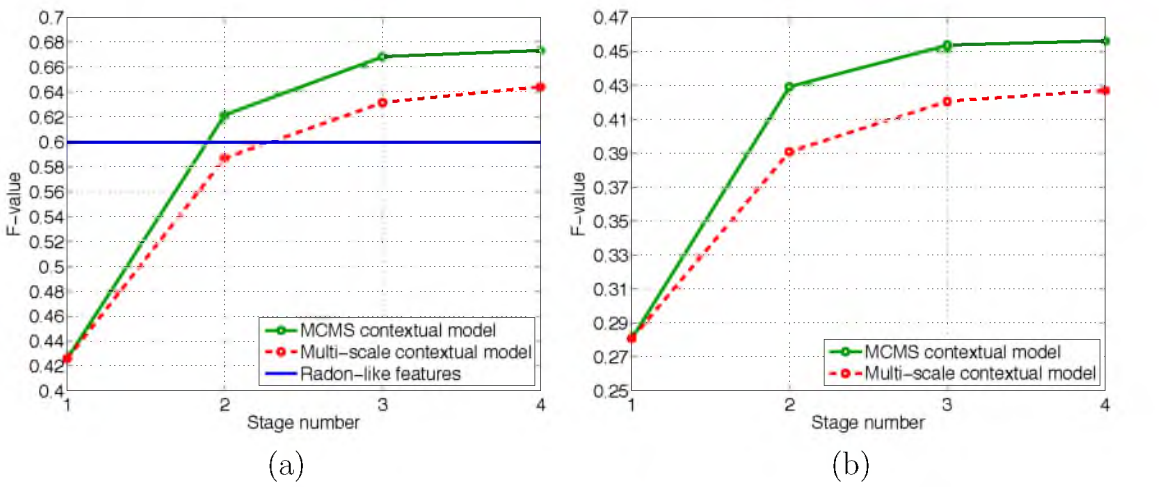


Figure 2.16: Mitochondria and synapse segmentation experiment on the Drosophila VNC dataset. (a) The test F-value at different stages of the series for different methods (mitochondria segmentation). (b) The test F-value at different stages of the series for different methods (synapse segmentation).

oversegmented regions for cell segmentation. These postprocessing approaches can improve Rand error [59] for membrane detection. However, in our proposed method we target the pixel error, and our method can be used for general computer vision datasets. The mitochondria and synapse segmentation results also can be improved by applying morphological postprocessing, which removes tiny false positive errors. Our goal in the experiment section was to validate the multiscale and the MCMS contextual models, and study of postprocessing approaches are beyond the scope of this study.

2.5 Conclusion

We develop a supervised segmentation framework that exploits contextual information from multiple objects and at different scales for learning discriminative models. Our multiclass multiscale (MCMS) contextual model enables an implicit learning of geometrical relationships and dependencies among multiple objects present in an image. We applied our method to object segmentation in EM images. Results indicate that using multiscale and cross-object contextual information can improve the segmentation results for each of the components present in EM images, such as membrane, mitochondria, and synapse. It is worth noting that the proposed method

is not restricted to this application and can be used in other image segmentation problems.

Even though our model has hundreds of parameters to learn, the complexity remains tractable since classifiers are trained one at a time separately. Our model can specially be useful in segmentation of imbalanced datasets where only a few samples of a particular object/class are available. In these datasets, large classes can improve the segmentation results of the small classes by providing informative contextual information.

We conclude by discussing a possible extension of the MCMS model presented in this chapter. Our feature extraction model only exploits pixel intensities from input images and probabilities from context images. While this reduces the computational complexity and keeps the model simple, more complex features extracted from both input and context images can improve the results.

CHAPTER 3

CONTEXTUAL HIERARCHICAL MODEL

In this chapter, we introduce a new contextual framework, called *contextual hierarchical model* (CHM), which learns contextual information in a hierarchical framework for scene labeling. At each level of the hierarchy, a classifier is trained based on downsampled input images and outputs of previous levels. Our model then incorporates the resulting multiresolution contextual information into a classifier to segment the input image at original resolution. This training strategy allows for optimization of a joint posterior probability at multiple resolutions through the hierarchy. The contextual hierarchical model is purely based on the input image patches and does not make use of any fragments or shape examples. Hence, it is applicable to a variety of problems such as object segmentation and edge detection.

Unlike MCMS, which extract multiscale contextual information in an unsupervised way, CHM learns multiscale contextual information in a supervised setting. This enables CHM to pool more discriminative contextual information at test time and thus improve the performance. We demonstrate that CHM outperforms state-of-the-art methods on Stanford background and Weizmann horse datasets. It also outperforms state-of-the-art edge detection methods on the NYU depth dataset and achieves state-of-the-art results on the Berkeley segmentation dataset (BSDS 500).

3.1 Introduction

3.1.1 Graphical models

There have been many methods that employ graphical models to take advantage of contextual information for scene labeling. Markov Random Fields (MRF) [8,60–62] and Conditional Random Fields (CRF) [37,63] are the most popular approaches. He *et al.* [37] used CRF to capture contextual information at multiple scales. Larlus and

Jurie [60] used MRF on top of a bag-of-words based object model to ensure consistency of labeling. Gould *et al.* [8] defined an energy function over scene appearance and geometry and then developed an efficient inference technique for MRFs to minimize that energy. Kumar and Koller [61] formulated the energy minimization as an integer programming problem and proposed a linear programming relaxation to solve it. Tighe and Lazebnik [62] proposed an MRF-based superpixel matching that can be easily scaled to large datasets. Ladicky *et al.* [63] introduced a hierarchical CRF, which is able to combine features extracted from pixels and segments. For inference, they used a graph-cut [64] based method to find the MAP solution. Ren *et al.* [65] used a superpixel MRF together with a segmentation tree for RGB-D scene labeling.

Many graphical methods rely on presegmentation to superpixels [62,65] or multiple segment candidates [61, 66]. More powerful region-based features can be extracted from superpixels compared to pixels. Moreover, presegmentation to superpixels improves the computational efficiency of these models. However, it is known that superpixels might not adhere to the image boundaries [67] and thus can decrease labeling accuracy [65]. This motivated approaches using multiple segments as hypotheses. However, these methods can be problematic when dealing with cluttered images [63]. This motivated methods with hierarchical segmentation [63, 68].

Unlike previously cited approaches, our proposed method does not make use of any presegmentations or exemplars and works directly on image pixels. This allows our model to be applied to different problems without any modifications. Moreover, inference is simpler in our CHM compared to graphical models. It only requires the evaluation of classifier function and does not require searching the label space as in CRFs [69].

3.1.2 Convolutional networks

Deep learning is a very active area of research and has been widely used in the computer vision field. Convolutional networks (ConvNet) [70] are one of the most popular deep architectures. They were initially proposed for character recognition [70], but later applied successfully to image classification [71, 72] and object detection [73, 74]. They have also been used for biological image segmentation [18, 75, 76] and scene labeling [2,69]. Jain *et al.* used ConvNets to restore membranes in electron microscopic

(EM) images. Turaga *et al.* [75] used ConvNets to minimize the Rand index [59] instead of pixel error to improve the segmentation of EM images. Ciresan *et al.* trained a very large ConvNet with four convolutional layers followed by two fully connected layers. This method was used in the winning entry of the International Symposium on Biomedical Imaging (ISBI) neuronal segmentation challenge [77]. Grangier *et al.* [69] trained a ConvNet by iteratively adding new layers for scene parsing. Farabet *et al.* [2] proposed a multiscale ConvNet for scene parsing. Their framework contains multiple copies of a single network that are applied to a scale-space pyramid of input images. They performed some postprocessing methods to clean up the outputs generated by the ConvNet.

ConvNets can cover large contextual area compared to other methods, but they need several hidden layers with many free parameters. Training the ConvNets is computationally expensive and might take months or even years on CPUs [76]. Hence, GPU implementations, which speed up the training process, are usually needed in practice. Unlike ConvNets, our CHM can be trained on CPUs in a reasonable time. Moreover, we will show that CHM outperforms the ConvNets proposed in [2, 18, 76].

3.1.3 Cascaded classifiers

The idea of using multiple classifiers to model context has been proven successful to solve different computer vision problems. Fink and Perona [32] proposed the mutual boosting framework, which takes advantage of multiple detectors in a boosting architecture for object detection. Torralba *et al.* [4] proposed the boosted random field (BRF), which uses boosting to learn the graph structure of CRFs, for object detection and segmentation. Heitz *et al.* [5] proposed a different architecture to combine multiple classifiers, called the cascaded classifier model, for holistic scene understanding. Their model combines several classifiers tuned for some specific subtasks to improve the performance on all subtasks. Li *et al.* [6] introduced a feedback-enabled cascaded classification model, which jointly optimizes several subtasks in a two-layer cascade of classifiers. In a more related work, Tu and Bai [3] introduced the auto-context algorithm, which integrates both image features and contextual information to learn a series of classifiers for image segmentation. A filter bank is used to extract the

image features and the output of each classifier is used as the contextual information for the next classifier in the series. Jurrus *et al.* [7] also trained a series of artificial neural networks (ANN) [54], which learns a set of convolutional filters from the data instead of applying fixed filter banks to the input image. Their series architecture was improved by employing a multiscale representation of context during training [23]. The advantage of the cascaded classifier model over ConvNets is its easier training due to treating each classifier in the series one at a time.

We also introduce a segmentation framework that takes advantage of both input image features and contextual information. Similar to the auto-context algorithm, we use a filter bank to extract input image features. But we use a hierarchical architecture to capture contextual information at different resolutions. Moreover, this multiresolution contextual information is learned in a supervised framework, which makes it more discriminative compared to the above-mentioned methods. From the Bayesian point of view, CHM optimizes a joint posterior probability at multiple resolutions simultaneously. To our knowledge, supervised multiresolution contextual information has not previously been used in a scene labeling framework.

3.1.4 Edge detection

There is a large body of work in the area of edge detection. Many unsupervised techniques have been proposed for edge detection [44, 78–80]. Seminal Canny edge detector [79] is one of the earliest, and gPb [78] is one of the latest among these approaches. More recently, supervised techniques have been explored to improve the edge detection performance [43, 81–85]. Martin *et al.* [84] computed gradients for brightness, color, and texture channels on a circular disc located at each pixel. They then combined these hand-crafted features and used them as input to a logistic regression classifier for predicting edges. Dollar *et al.* [43] extracted tens of thousands of features at each pixel and then used a probabilistic boosting tree (PBT) [86] to find edges. Mairal *et al.* [85] proposed to learn discriminative sparse dictionaries to distinguish between "patches centered on an edge pixel" and "patches centered on a non-edge pixel." Ren and Bo [82] used gradients over learned sparse codes instead of hand designed gradients of [84] to achieve state-of-the-art performance. Lim *et al.* [81]

defined a set of sketch tokens by clustering the patches extracted from groundtruth images. Then, they trained a random forest to detect those tokens at test time. Finally, Dollar and Zitnick [83] made use of different edge patterns, *e.g.,* T-junctions and Y-junctions, present in images and used a structured random forest to learn those patterns. Their method is fast and generalizes well between different datasets.

We also approach the edge detection problem as a labeling problem. Our CHM is trained to distinguish between "patches centered on an edge pixel" and "patches centered on a non-edge pixel." We will show that CHM achieves near state-of-the-art performance on the Berkeley dataset [10] and outperforms state-of-the-art methods [82, 83] on the NYU depth dataset. Moreover, we will demonstrate that generalization performance of CHM across different datasets is better compared to [82, 83].

3.2 Contextual Hierarchical Model

The contextual hierarchical model (CHM) is illustrated in Fig. 3.1. First, a multiresolution representation of the input image is obtained by applying downsampling sequentially. Next, a series of classifiers are trained at different resolutions from the finest resolution to the coarsest resolution. At each resolution, the classifier is trained based on the outputs of the previous classifiers in the hierarchy and the input image at that resolution. Finally, the outputs of these classifiers are used to train a new classifier at original resolution. This classifier exploits the rich contextual information from multiple resolutions. The whole training process targets a joint posterior probability at multiple resolutions (see section 3.2.3). We describe different steps of the model separately in the following subsections.

3.2.1 Bottom-up step

Let $X = (x(m,n))$ be the $2D$ input image with a corresponding ground truth $Y = (y(m,n))$ where $y(m,n) \in \{0,1\}$ is the class label for pixel (m,n). For notational simplicity, we use $1D$ vectors $X = (x_1, x_2, \ldots, x_n)$ and $Y = (y_1, y_2, \ldots, y_n)$ to denote the input image and corresponding ground truth, respectively. The training dataset then contains K input images, $\mathbf{X} = \{X_1, X_2, \ldots, X_K\}$, and corresponding ground

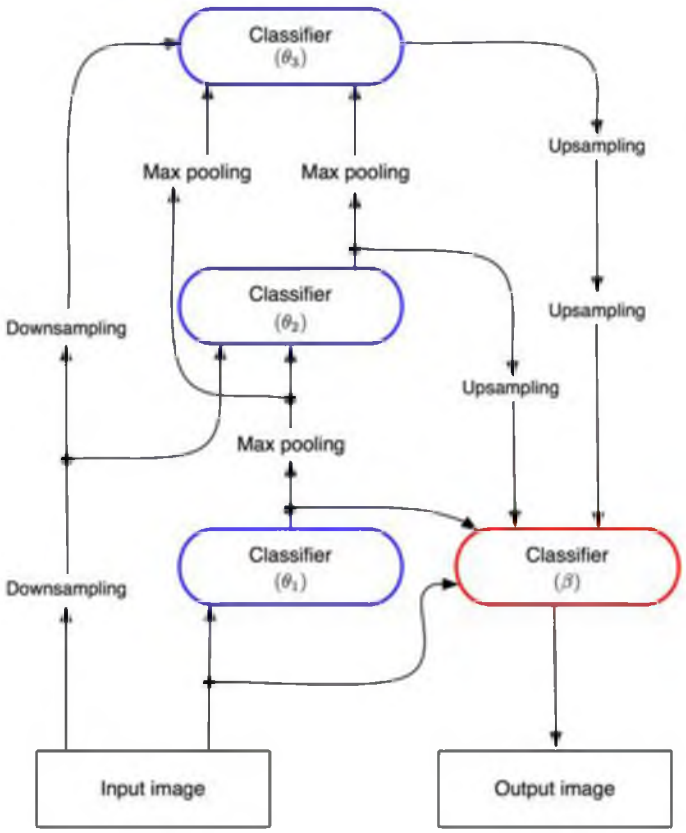


Figure 3.1: Illustration of the contextual hierarchical model. The blue classifiers are learned during the bottom-up step and the red classifier is learned during the top-down step. In the bottom-up step, each classifier takes the outputs of lower classifiers as well as the input image as input. The height of the hierarchy, L, is three in this model, but it can be extended to any arbitrary number.

truth images, $\mathbf{Y} = \{Y_1, Y_2, \ldots, Y_K\}$.[1] We also define the $\Phi(\cdot, l)$ operator, which performs down-sampling l times by averaging the pixels in each 2×2 window, and the $\Gamma(\cdot, l)$ operator, which performs max-pooling l times by finding the maximum pixel value in each 2×2 window. Each classifier in the hierarchy has some internal parameters θ_l, which are learned during training

$$\hat{\theta}_\mathbf{l} = \arg\max_{\theta_\mathbf{l}} P(\mathbf{\Gamma}(\mathbf{Y}, l-1) \mid \mathbf{\Phi}(\mathbf{X}, l-1), \mathbf{\Gamma}(\hat{\mathbf{Y}}^\mathbf{1}, l-1), \ldots, \mathbf{\Gamma}(\hat{\mathbf{Y}}^{\mathbf{l-1}}, 1); \theta_\mathbf{l}) \tag{3.1}$$

[1]Unless specified otherwise, upper case symbols, *e.g.,* X, Y, denote a particular vector, lower case symbols, *e.g.,* x, y, denote the elements of a vector, and bold-face symbols, *e.g.,* $\mathbf{X}$, $\mathbf{Y}$, denote a set of vectors.

where $\hat{\mathbf{Y}}^{\mathbf{1}}, \ldots, \hat{\mathbf{Y}}^{\mathbf{l-1}}$ are the outputs of classifiers at the lower levels of the hierarchy. The classifier output of each level is obtained using inference

$$\hat{Y}^l = \arg\max_Y P(Y \mid \Phi(X, l-1), \Gamma(\hat{Y}^1, l-1), \ldots, \Gamma(\hat{Y}^{l-1}, 1); \hat{\theta}_1) \tag{3.2}$$

Each classifier in the l'th level of the hierarchy takes outputs of all lower level classifiers, *i.e.,* $\hat{Y}^1, \ldots, \hat{Y}^{l-1}$, which provide multiresolution contextual information. For $l = 1$, no prior information is used, and the classifier parameters, θ_1, are learned only based on the input image.

It is worth mentioning that classifiers at higher levels of the hierarchy have access to contextual information from larger areas because they are trained on downsampled images.

3.2.2 Top-down step

Unlike the bottom-up step where multiple classifiers are learned, only one classifier is trained in the top-down step. Once all the classifiers are learned in the bottom-up step, a top-down path is used to feed coarser resolution contextual information into a classifier, which is trained at the finest resolution. We define $\Omega(\cdot, l)$ operator that performs upsampling l times by duplicating each pixel. For a hierarchical model with L levels, the classifier is trained based on the input image and the outputs of stages 1 to L obtained in the bottom-up step. The internal parameters of the classifier, β, are learned using the following

$$\hat{\beta} = \arg\max_\beta P(\mathbf{Y} \mid \mathbf{X}, \hat{\mathbf{Y}}^{\mathbf{1}}, \boldsymbol{\Omega}(\hat{\mathbf{Y}}^{\mathbf{2}}, 1), \ldots, \boldsymbol{\Omega}(\hat{\mathbf{Y}}^{\mathbf{L}}, L-1); \beta) \tag{3.3}$$

The output of this classifier can be obtained using the following for inference

$$\hat{Z} = \arg\max_Y P(Y \mid X, \hat{Y}^1, \Omega(\hat{Y}^2, 1), \ldots, \Omega(\hat{Y}^L, L-1); \hat{\beta}) \tag{3.4}$$

The top-down classifier takes advantage of prior information from multiple resolutions. This multiresolution prior is an efficient mixture of both local and global information

since it is drawn from different scales. In a related work, Seyedhosseini *et al.* [23] proposed a multiscale contextual model that exploits contextual information from multiple scales. The advantage of the model proposed here is that the context images are learned at different scales in a supervised framework, while the multiscale contextual model uses simple filtering to create context images at different scales. This allows CHM to optimize a joint posterior at different scales. The overall learning and inference algorithms for the contextual hierarchical model are described in Algorithm 2 and Algorithm 3, respectively.

Algorithm 2 Learning algorithm for the CHM.

Input: A set of training images together with their binary groundtruth images, $\mathbf{S} = \{(X_i, Y_i), i = 1, \ldots, K\}$ and the height of the hierarchy, L.
Output: $\mathbf{\Theta} = \{\hat{\theta}_1, \ldots, \hat{\theta}_L, \hat{\beta}\}$.

- Learn the first classifier, θ_1, using equation (3.1) without any prior information and only based on the input image features.
- Compute the output of first classifier, $\hat{\mathbf{Y}}^{\mathbf{1}}$, using equation (3.2).

for $l = 2$ *to* L **do**

- Learn the l'th classifier, $\hat{\theta}_{\mathbf{l}}$, using equation (3.1).
- Compute output of the l'th classifier, $\hat{\mathbf{Y}}^{\mathbf{l}}$, using equation (3.2).

end for

- Learn the top-down classifier, $\hat{\beta}$, using equation 3.3.

Algorithm 3 Inference algorithm for the CHM.

Input: An input image X, $\mathbf{\Theta}$, L.
Output: $\hat{Z}$.

- Compute the output of first classifier, $\hat{Y}^1$, using equation (3.2).

for $l = 2$ *to* L **do**

- Compute output of the l'th bottom-up classifier, $\hat{Y}^l$, using equation (3.2).

end for

- Compute output of the top-down classifier, $\hat{Z}$, using equation (3.4).

3.2.3 Probabilistic interpretation

Given the training set $\mathbf{X}$, containing $T = K \times n$ samples and corresponding labels $\mathbf{Y}$, a common approach is to find the optimal solution by solving the maximum aposteriori (MAP) equation

$$\log \prod_t P(Y_t \mid X_t; \boldsymbol{\Theta}) \tag{3.5}$$

There are two common strategies to solve this optimization. The first strategy, *i.e.,* the generative approach, decomposes the posterior to likelihood, $P(X_t \mid Y_t)$, and prior, $P(Y_t)$. The second strategy, *i.e.,* the discriminative approach, targets the posterior distribution directly. Our hierarchical model falls into the second category. However, it differs from other approaches in a sense that it optimizes a joint posterior at multiple resolutions, *i.e.,*

$$\begin{aligned} &\log \prod_t P\left(Y_t, \Gamma(Y_t, 0), \ldots, \Gamma(Y_t, L-1) \mid X_t; \boldsymbol{\Theta}\right) = \\ &\sum_t \log P\left(Y_t, \Gamma(Y_t, 0), \ldots, \Gamma(Y_t, L-1) \mid X_t; \boldsymbol{\Theta}\right) \end{aligned} \tag{3.6}$$

where Γ is the maxpooling operator and L is the number of levels in the hierarchy. Using $P(A, B \mid C) = P(A \mid B, C)P(B \mid C)$, (3.6) can be rewritten as

$$\begin{aligned} &\sum_t \log \Big(P\big(Y_t \mid X_t, \Gamma(Y_t, 0), \ldots, \Gamma(Y_t, L-1); \boldsymbol{\Theta}\big) \times \\ &\qquad P\big(\Gamma(Y_t, L-1) \mid X_t, \Gamma(Y_t, 0), \ldots, \Gamma(Y_t, L-2); \boldsymbol{\Theta}\big) \times \\ &\qquad \cdots \times P\big(\Gamma(Y_t, 0) \mid X_t; \boldsymbol{\Theta}\big) \Big) = \\ &\underbrace{\sum_t \log P\big(Y_t \mid X_t, \Gamma(Y_t, 0), \ldots, \Gamma(Y_t, L-1); \boldsymbol{\Theta}\big)}_{\text{Top-down:} J_2(\mathbf{X}, \mathbf{Y}; \boldsymbol{\Theta})} + \\ &\underbrace{\sum_l \sum_t \log P\big(\Gamma(Y_t, l) \mid X_t, \Gamma(Y_t, 0), \ldots, \Gamma(Y_t, l-1); \boldsymbol{\Theta}\big)}_{\text{Bottom-up:} J_1(\mathbf{X}, \mathbf{Y}; \boldsymbol{\Theta})} \end{aligned} \tag{3.7}$$

Note that the optimization problems nicely splits down to two subproblems, *i.e.,* $J_1(\mathbf{X}, \mathbf{Y}; \boldsymbol{\Theta})$ and $J_2(\mathbf{X}, \mathbf{Y}; \boldsymbol{\Theta})$, which are solved during bottom-up and top-down steps, respectively.

In practice, the optimization is done in a greedy way. The output of the classifier at level l, $\hat{Y}^l$, is used as an approximation of the groundtruth at that resolution, $\Gamma(Y, l-1)$. Therefore, the following optimization problems are solved during training

Bottom-up:

$$\max_{\mathbf{\Theta}} J_1(\mathbf{X}, \mathbf{Y}; \mathbf{\Theta}) =$$
$$\max_{\mathbf{\Theta}} \sum_{l} \sum_{t} \log P\big(\Gamma(Y_t, l) \mid X_t, \hat{Y}_t^1, \ldots, \hat{Y}_t^l); \mathbf{\Theta}\big) \tag{3.8}$$

Top-down:

$$\max_{\mathbf{\Theta}} J_2(\mathbf{X}, \mathbf{Y}; \mathbf{\Theta}) =$$
$$\max_{\mathbf{\Theta}} \sum_{t} \log P\big(Y_t \mid X_t, \hat{Y}_t^1, \ldots, \hat{Y}_t^L; \mathbf{\Theta}\big) \tag{3.9}$$

This greedy approach makes the training simple and tractable. It is noteworthy that each of the terms of the outer summation in J_1 is corresponding to one level of the hierarchy. Due to the greedy optimization, a second stage of CHM can improve the results. In the second stage, the top-down classifier of the previous stage is used as the first classifier in the bottom-up step.

3.2.4 Classifier selection

Even though our problem formulation is general and not restricted to any specific type of classifier, in practice we need a fast and accurate classifier that is robust against overfitting. Among off-the-shelf classifiers, we consider artificial neural networks (ANN), support vector machines (SVM), and random forests (RF). ANNs are slow at training time due to the computational cost of backpropagation. SVMs offer good generalization performance, but choosing the kernel function and the kernel parameters can be time consuming since they need to be adopted for each classifier in the CHM. Furthermore, SVMs are not intrinsically probabilistic and thus are not completely suitable for our CHM model. Random forests provide an unbiased estimate of testing error, but they are prone to overfitting in the presence of noise. In section 3.3.1.1 we show that overfitting can disrupt learning in the CHM model.

We adopt logistic disjunctive normal networks (LDNN) [25] as the classifier in CHM. LDNN is a powerful classifier, which consists of one adaptive layer implemented by logistic sigmoid functions followed by two fixed layers of logical units that compute

conjunctions and disjunctions, respectively. LDNN allows an intuitive initialization using k-means clustering and outperforms neural networks, SVMs, and random forests on several standard datasets [25]. Finally, LDNNs are fast to train due to the single adaptive layer, which makes them suitable for the CHM architecture. Later in Chapter 5, we will show that the same formulation can be applied to decision trees and random forests to fine tune the parameters and improve the performance.

3.2.5 Logistic disjunctive normal network architecture

Any Boolean function $b : \mathbf{B}^n \rightarrow \mathbf{B}$ where $\mathbf{B} = \{0, 1\}$ can be written as a disjunction of conjunctions, which is also known as the disjunctive normal form [87]. Now consider the binary classification problem $f : \mathbf{R}^n \rightarrow \mathbf{B}$. Let $\mathbf{X}_+ = \{X \in \mathbf{R}^n : f(X) = 1\}$ and $\mathbf{X}_- = \{X \in \mathbf{R}^n : f(X) = 0\}$. One possibility for expressing f in disjunctive normal form is to approximate $\mathbf{X}_+$ as the union of axis aligned hypercubes in $\mathbf{R}^k$. We first define the box function

$$h_{L,U}(x) = \begin{cases} 1, & L \leq x \leq U \\ 0, & otherwise \end{cases} \tag{3.10}$$

where $L \in \mathbf{R}$, $U \in \mathbf{R}$ and $L \leq U$. Then the disjunctive normal form can be rewritten as

$$\tilde{f}(X) = \bigvee_i \left(\bigwedge_{j=1}^{n} h_{L_{ij},U_{ij}}(x_j) \right) \tag{3.11}$$

where x_j denotes the j'th element of the vector X. This formulation is also known as a fuzzy min-max neural network [88]. The most important drawback of this model is its limitation to axis aligned decision boundaries, which can significantly increase the number of conjunctions necessary for a good approximation. We propose to construct a significantly more efficient approximation in disjunctive normal form by approximating $\mathbf{X}_+$ as the union of convex sets, which are defined as the intersection of arbitrary half-spaces in $\mathbf{R}^n$. By using hyperplanes to define the half-spaces, we get the approximation

$$\tilde{f}(X) = \bigvee_i \underbrace{\left(\bigwedge_j h_{ij}(X) \right)}_{q_i(X)} \tag{3.12}$$

where the half-spaces are defined as

$$h_{ij}(X) = \begin{cases} 1, & \sum_{k=1}^{n} w_{ijk} x_k + b_{ij} \geq 0 \\ 0, & otherwise \end{cases} \tag{3.13}$$

Our next step is to replace equation (3.12) with a differentiable approximation. First, a conjunction of binary variables $\bigwedge_j h_{ij}(X)$ can be replaced by their product $\prod_j h_{ij}(X)$. Then, using De Morgan's laws we can replace the disjunction of binary variables $\bigvee_i q_i(X)$ with $\neg \bigwedge_i \neg q_i(X)$, which in turn can be replaced by the expression $1-\prod_i(1-q_i(X))$. Finally, we can approximate the half-spaces $h_{ij}(X)$ with the logistic sigmoid function

$$\sigma_{ij}(X) = \frac{1}{1 + e^{-\sum_{k=1}^{n} w_{ijk} x_k + b_{ij}}} \tag{3.14}$$

This gives in the differentiable disjunctive normal form approximation to $\mathbf{f}$

$$\tilde{f}(X) = 1 - \prod_i (1 - \underbrace{\prod_j \sigma_{ij}(X)}_{g_i(X)}) \tag{3.15}$$

This formulation can be interpreted as a 3-layer network. The input vector, *i.e.,* X, is mapped to the first layer by sigmoid functions in (3.14). The first layer consists of N groups of nodes with M nodes each. The nodes in each group are connected to a single node in the second layer. Each node in the second layer implements the logical negations of the conjunctions $g_i(X)$ in (3.15). The output layer is a single node, which implements the disjunction using De Morgan's law. We will refer to such a network as a $N \times M$ LDNN. Notice that the only parameters of the network are the weights, w_{ijk}, and biases, b_{ij}, of the connections between the inputs and the first layer of sigmoid functions. This is an advantage of using parameterless functions, *i.e.,* the products, for representing the conjunctions.

Given a set of training examples $\mathbf{T}$ of pairs (X, y) where y denotes the desired binary class corresponding to X and a classifier $f(X)$, the quadratic error over the training set is

$$E(f, \mathbf{T}) = \sum_{(X,y) \in \mathbf{T}} (y - f(X))^2 \tag{3.16}$$

The gradient of the error function with respect to the parameter w_{ijk} in the LDNN architecture, evaluated for the training pair (X, y), is

$$\frac{\partial E}{\partial w_{ijk}} = -2(y - f(X)) \prod_{r \neq i} (1 - g_r(X)) \\ g_i(X)\,(1 - \sigma_{ij}(X))\,x_k \tag{3.17}$$

Similarly the gradient of the error function with respect to the bias term b_{ij} is

$$\frac{\partial E}{\partial b_{ij}} = -2(y - f(X)) \prod_{r \neq i} (1 - g_r(X)) \\ g_i(X)\,(1 - \sigma_{ij}(X)) \tag{3.18}$$

The parameters of the LDNN can be learned by minimizing (3.16) using the gradient descent algorithm and (3.17) and (3.18).

Finally, the disjunctive normal form used in the the LDNN permits a very simple and intuitive initialization of the model parameters. Since each conjunction is a convex set in $\mathbf{R}^n$ and $\mathbf{X}_+$ is approximated as the union of N such conjunctions, we can view the convex sets generated by the conjunctions as subclusters of $\mathbf{X}_+$. To initialize a model with N conjunctions and M sigmoid functions per conjunction, we

- Use the k-means algorithm to partition $\mathbf{X}_+$ into N clusters. Let $C_{+,i}$ be the centroid of the i'th cluster.
- Use the k-means algorithm to partition $\mathbf{X}_-$ into M clusters. Let $C_{-,j}$ be the centroid of the j'th cluster.
- Initialize the weight vectors W_{ij} as the unit length vectors from the negative to the positive centroids. In other words, let $V_{ij} = C_{+,i} - C_{-,j}$ and let $W_{ij} = \frac{V_{ij}}{|V_{ij}|}$.
- Initialize the bias terms b_{ij} such that the sigmoid functions $\sigma_{ij}(X)$ take the value 0.5 at the midpoints of the lines connecting the positive and negative cluster centroids. In other words, let $b_{ij} = < W_{ij}, 0.5(C_{+,i} + C_{-,j}) >$ where $< \cdot, \cdot >$ denotes the inner product of vectors.

3.2.6 Feature selection

In this section, we describe the set of features extracted from input and context images in CHM. The features that we extract from input images include Haar features [89] and histogram of oriented gradients (HOG) features [90]. These features are efficient to compute and somewhat complementary to each other [3]. For color images, Haar and HOG features are computed for each channel separately. We also use dense SIFT flow features [91] computed at each pixel. In addition, we apply a set of Gabor

filters with different parameters and Canny edge detector to obtain more features. Beside these appearance features, we also use position and its higher moments (up to 2^{nd} order), which are known to be informative for scene labeling [65, 68]. Finally, we use a 15×15 sparse stencil structure, which contains 57 samples, to sample the neighborhood around each pixel. In summary, we extract 647 features from color images and 457 features from gray scale images.

Context features are obtained from the outputs of classifiers in the hierarchy. We used a 15×15 stencil to sample context images around each pixel. We also tried larger and more dense sampling structures, *e.g.,* 21×21 patch, but they had negligible impact on the performance. We do not extract any other features beside the neighborhood samples from context images.

3.3 Experimental Results

We perform experimental studies to evaluate the performance of CHM on three different applications: Scene labeling, edge detection, and biomedical image segmentation. The diversity among these applications shows the broad applicability of our method. In all the applications, we used a set of nearly identical parameters, including the number of levels in CHM and the features parameters. Following the reproducible research instructions [28], we maintain a web page containing the source codes and scripts used to generate the results in this section.[2]

3.3.1 Scene labeling

We show the performance of CHM on a binary scene labeling dataset, *i.e.,* Weizmann dataset [9], as well as an outdoor scene labeling dataset with multiple classes, *i.e.,* Stanford background dataset [8].

3.3.1.1 Weizmann dataset

The Weizmann dataset [9] contains 328 gray scale horse images with corresponding foreground-background truth maps. Similar to Tu *et al.* [3], we used half of the images for training and the remaining images were used for testing. The task is to segment horses in each image. We used the features described in section 6.2.2. Note that we

[2] `http://www.sci.utah.edu/~mseyed/Mojtaba_Seyedhosseini/CHM.html`

do not use location information for this dataset since horses are mostly centered in the images, which would create an unfair advantage.

We used a 24×24 LDNN as the classifier in a CHM with two stages and 5 levels per stage. To improve the generalization performance, we adopted the dropout idea. Hinton *et al.* [92] showed that removing 50% of the hidden nodes in a neural network during the training can improve the performance on the test data. Using the same idea, we randomly removed half of the nodes in the second layer and half of the nodes per group in the first layer at each iteration during the training. At test time, we used the LDNN that contains all of the nodes with their outputs square rooted to compensate for the fact that half of them were active during the training time.

For comparison, we trained a CHM with random forest as the classifier. To avoid overfitting, only $\frac{1}{20}$ of samples were used to train 100 trees in the random forest. We also trained a multiscale series of artificial neural networks (MSANN) as in [23]. Three metrics were used to evaluate the segmentation accuracy: Pixel accuracy, F-value $= \frac{2 \times precision \times recall}{precision + recall}$, and G-mean$= \sqrt{recall \times TNR}$ where $TNR = \frac{true\ negative}{true\ negative + false\ positive}$. Unlike F-value, G-mean is symmetric with respect to positive and negative classes. In Table 3.1 we compare the performance of CHM with some state-of-the-art methods. CHM outperforms other state-of-the-art methods. It is worth noting that CHM does not make use of fragments, and it is based purely on discriminative classifiers that use neighborhood information.

The CHM-LDNN outperforms the state-of-the-art methods while the CHM-RF

Table 3.1: Testing performance of different methods on the Weizmann horse dataset.

Method	F-value	G-mean	Pixel accuracy
KSSVM [93]	–	–	94.60%
TWM [94]	–	–	94.70%
Auto-context [3]	84%	–	–
Levin & Weiss [95]	–	–	95.2%
MSANN [23]	87.58%	92.76%	94.34%
CHM-RF	83.15%	90.20%	92.33%
CHM-LDNN	**89.89%**	**94.39%**	**95.37%**

performs worse than those methods. The training and testing F-value of the classifiers trained at the original resolution in the CHM, *i.e.,* the classifiers at the bottom of hierarchy, for both LDNN and random forest are shown in Fig. 3.2. It shows how overfitting propagates through the stages of the CHM when the random forest is used as the classifier. The overfitting disrupts the learning process because there are too few mistakes in the training set compared to the testing set as we go through the stages. For example, the overfitting in the first stage does not permit the second stage to learn the typical mistakes from the first stage that will be encountered at testing time. We tried random forests with different parameters to overcome this problem but were unsuccessful. Fig. 3.3 shows four examples of our test images and their segmentation results using different methods. The CHM-LDNN outperforms the other methods in filling the body of horses.

3.3.1.2 Stanford background dataset

The Stanford background dataset [8] contains 715 images of urban and rural scenes, collected from other public datasets such that each image is approximately 240×320 pixels and contains at least one foreground object. This dataset is composed of eight classes, one foreground and seven other classes, and the groundtruth images,

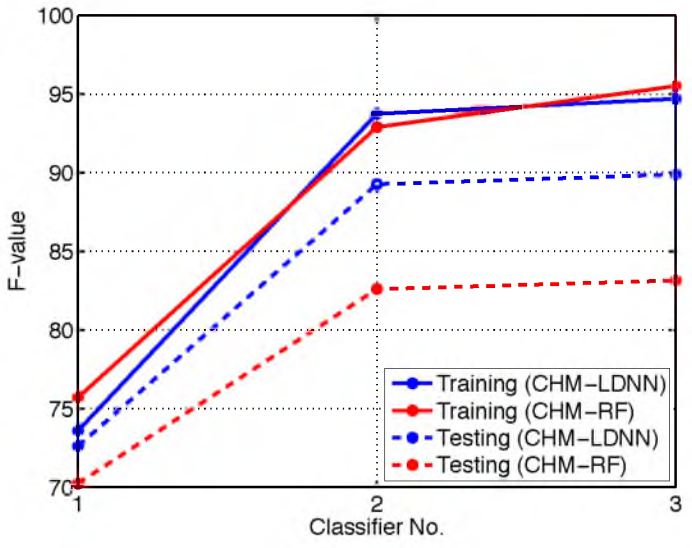


Figure 3.2: F-value of the classifiers trained at the original resolution in the CHM with LDNN and random forest. The overfitting in the random forest makes it useless in the CHM architecture.

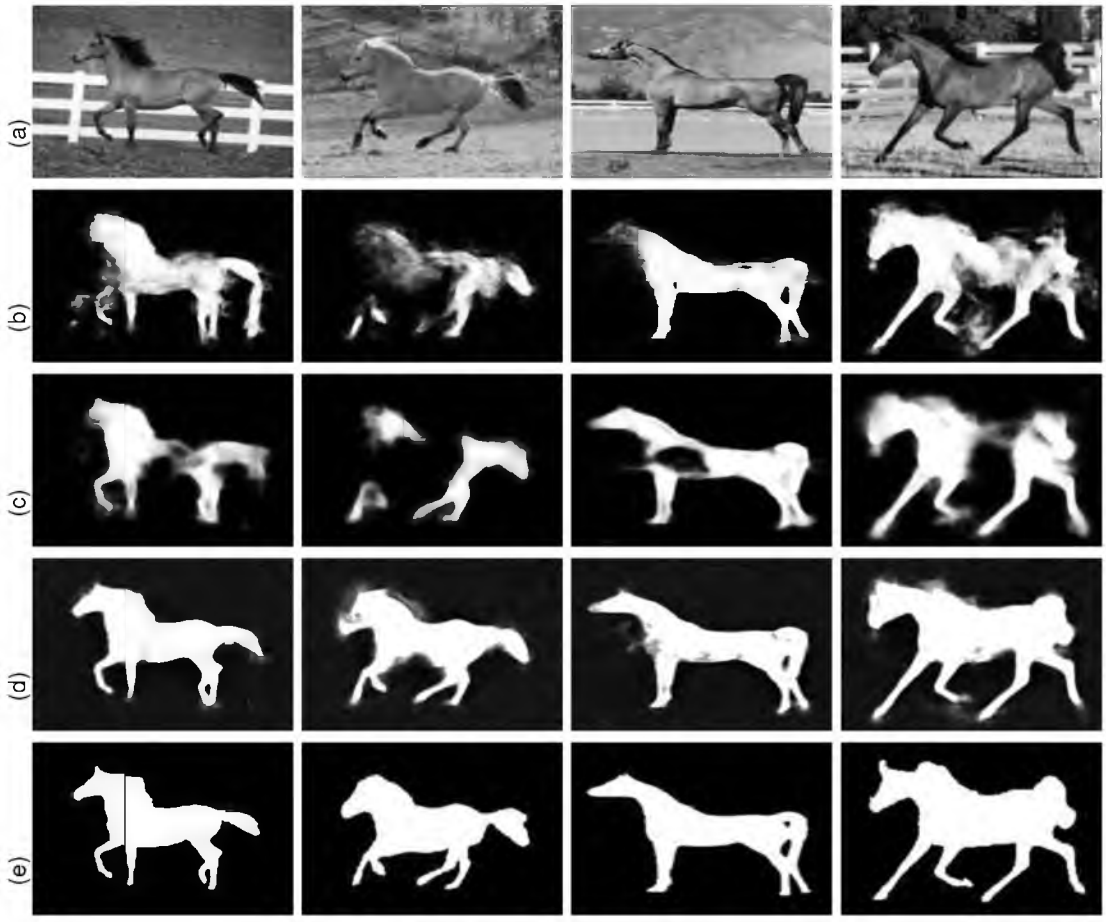


Figure 3.3: Test results of the Weizmann horse dataset. (a) Input image, (b) MSANN [23], (c) CHM-RF, (d) CHM-LDNN, (e) ground truth images. The CHM-LDNN is more successful in completing the body of horses.

obtained from Amazon Mechanical Turk, are included in the dataset. We followed the standard evaluation procedure for this dataset, which is performing 5-fold cross-validation with the dataset randomly split into 572 training images and 143 test images.

We trained eight CHMs in a one-versus-all architecture. To take advantage of intraclass contextual information, we allowed CHMs to communicate with each other at three upper levels of the hierarchy. At those levels, classifiers get samples of context images of other classes as well as their own class. The performance of CHM with and without intraclass connection is reported in Table 3.2. Our CHM achieves state-of-the-art performance in terms of pixel accuracy. Due to the absence of any global constraint for label consistency, CHM performs worse than [2,65] in terms of

Table 3.2: Testing performance of different methods on Stanford background dataset [8]: Pixelwise accuracy, class-average accuracy, and computation time.

Method	Pixel Acc.	Class Acc.	CT (sec.)
Region-based Energy [8]	76.4%	–	$10-600$
Selecting Regions [61]	79.4%	–	600
Stacked Hierarchical Labeling [68]	76.9%	66.2%	12
Superparsing [62]	77.5%	–	10
Recursive Neural Networks [96]	78.1%	–	–
Pylon Model [97]	81.9%	72.4%	60
Ren *et al.* [65]	**82.9**%	74.5%	–
Singlescale ConvNet [2]	66%	56.5%	**0.35**
Multiscale ConvNet [2]	78.8%	72.4%	0.6
Multiscale ConvNet+ CRF on gPb [2]	81.4%	**76.0**%	60.5
CHM	82.30%	73.70%	60
CHM with Intra-class Connection	**82.95**%	74.32%	65

class-average accuracy. Similar to [2], we computed superpixels [98] for each image and then assign the most common label, based on CHM output, to each superpixel. Unlike [2], this approach had negligible impact on the performance and improved the pixel accuracy only to 83%. This shows CHM is a powerful pixel classifier. In our experiment, inference took about 65 seconds for each image (half of it was spent on computing the features). A few test samples of the Stanford background dataset, and corresponding CHM results are shown in Fig. 3.4. Using intraclass connection improves the label consistency in the results.

The 8-class confusion matrix of CHM is shown in Fig. 3.5. The hard classes are mountain, water, and foreground. This is consistent with the reported results in [65, 68]. Even though the performance of CHM is similar to [65] for most of the classes, it performs significantly better on the foreground category compared to [65] achieving 74.1% vs 63%.

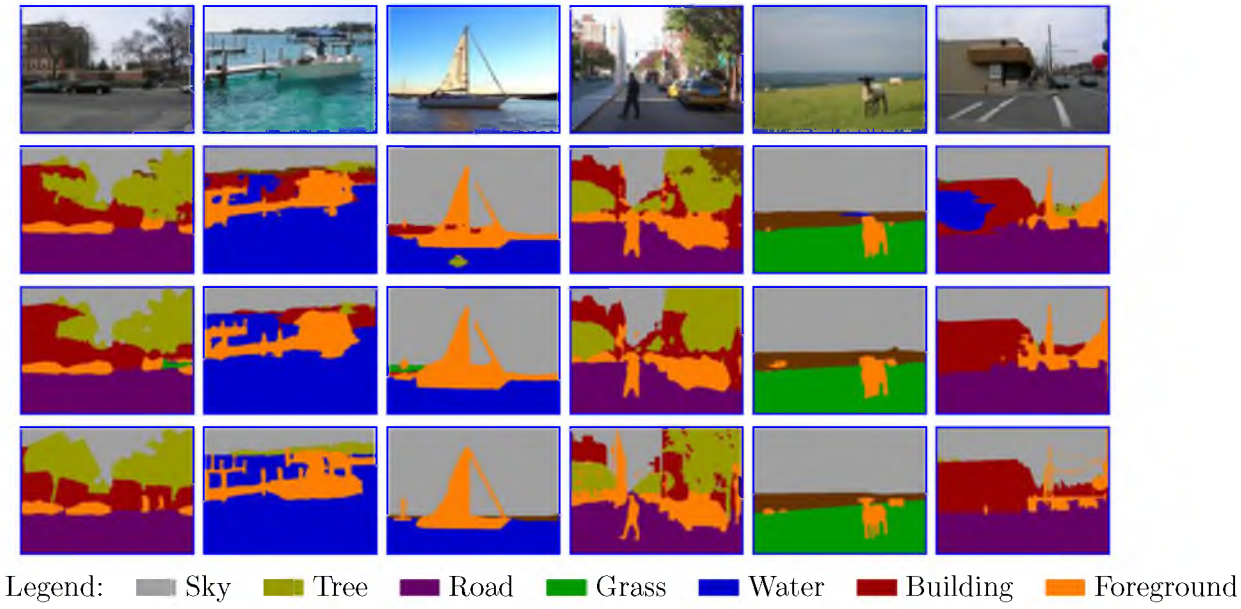


Figure 3.4: Test samples of scene labeling on Stanford background dataset [8]. **First row:** Input image, **second row:** CHM, **third row:** CHM with intraclass connection, **Fourth row:** Groundtruth. Using intraclass contextual information improves the performance.

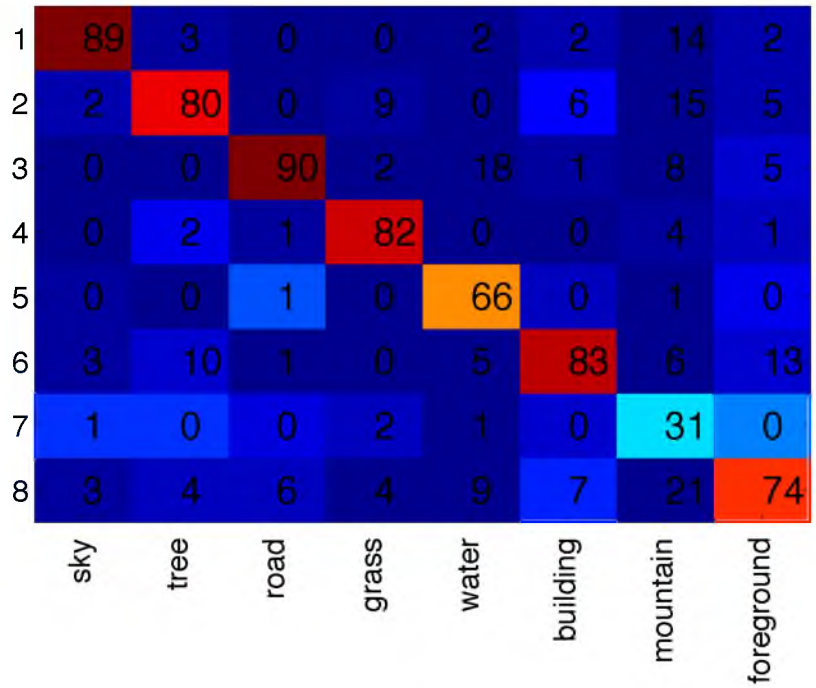


Figure 3.5: The confusion matrix of CHM results on the Stanford background dataset [8]. The overall class-average accuracy is 74.32%.

3.3.2 Edge detection

In this section we show the performance of CHM on two edge detection datasets: BSDS 500 [10] and NYU Depth (v2) [99]. We used the popular evaluation framework available in the gPb package [78] to compare CHM performance with other methods. The evaluation framework computes three metrics: F-value computed with a fixed threshold for the entire dataset (ODS), F-value computed with per-image best thresholds (OIS), and the average precision (AP).

We trained a CHM with 5 levels for both datasets. Similar to [81,83], we adopted a multiscale strategy to compute edge maps. That is, at test time, we ran the trained CHM on the original, as well as double and half resolution versions of each input image. We then resized the results to the original image resolution and averaged them to obtain the edge map. We also used the standard nonmaximal suppression, suggested in [78,81–83], to obtain thinned edges.

3.3.2.1 BSDS 500 dataset

Berkeley segmentation dataset and benchmarks (BSDS 500) [10,78] is an extension of BSDS 300 dataset and used widely for the evaluation of edge detection techniques. It contains 200 training, 100 validation, and 200 testing images of resolution 321×481 pixels (roughly). The human annotations for each image is included in the dataset. The evaluation metrics are reported in Table 3.3. The precision-recall curves for CHM and four other methods are shown in Fig. 3.6. Note that CHM achieves high precision and recall at both ends of the precision-recall curve. While CHM performs

Table 3.3: Testing performance of different methods on BSDS 500 dataset [10]. CHM achieves near state-of-the-art performance in terms of ODS and OIS and improves over other methods significantly in terms of AP.

Method	ODS	OIS	AP
gPb-OWT-UCM [78]	0.726	0.760	0.727
Sketch Tokens [81]	0.728	0.746	0.780
SCG [82]	0.739	0.758	0.773
SE [83]	**0.741**	**0.760**	0.780
CHM	0.735	0.751	**0.804**

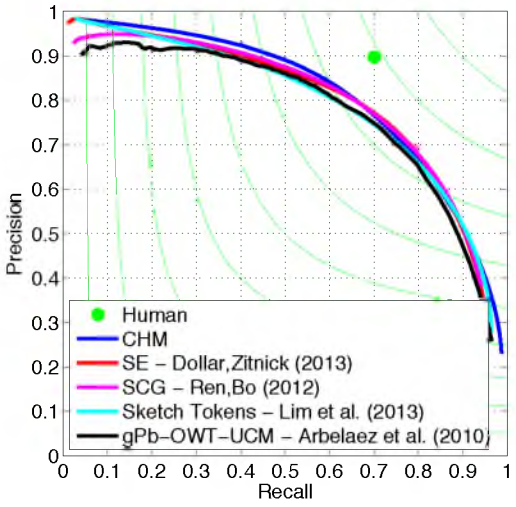


Figure 3.6: Precision-recall curves of CHM in comparison with other methods for BSDS 500 dataset [10].

about the same as SCG [82] and SE [83] in terms od ODS and OIS, it achieves state-of-the-art performance in terms of AP. It must be emphasized that unlike gPb [78] and SCG [82], our CHM does not include any globalization step and only relies on the local patch information. In addition, our CHM is a general patch-based model and unlike gPb [78], SCG [82], and SE [83] can be used in general scene labeling frameworks. Finally we will show in section 3.3.2.3 that the cross-dataset generalization performance of CHM is significantly better than other learning-based approaches, *i.e.,* sketch tokens [81], SCG [82], and SE [83]. A few test examples of BSDS 500 dataset and corresponding edge detection results are shown in Fig. 3.7. As shown in our results, CHM captures finer details such as upper stairs in the first row, steeples in the second row, and wheels in the third row.

3.3.2.2 NYU depth dataset (v2)

The NYU depth dataset (v2) [99] is an RGB-D dataset containing 1449 pairs of RGB and depth images of resolution 480×640 pixels, with corresponding groundtruth semantic segmentations. We used the scripts provided by the authors of [82] to adopt this dataset for edge detection.[3] They used 60% of the images for training (869

[3]The scripts are available at `http://homes.cs.washington.edu/~xren/research/nips2012/sparse_contour_gradients_v1.1.zip`

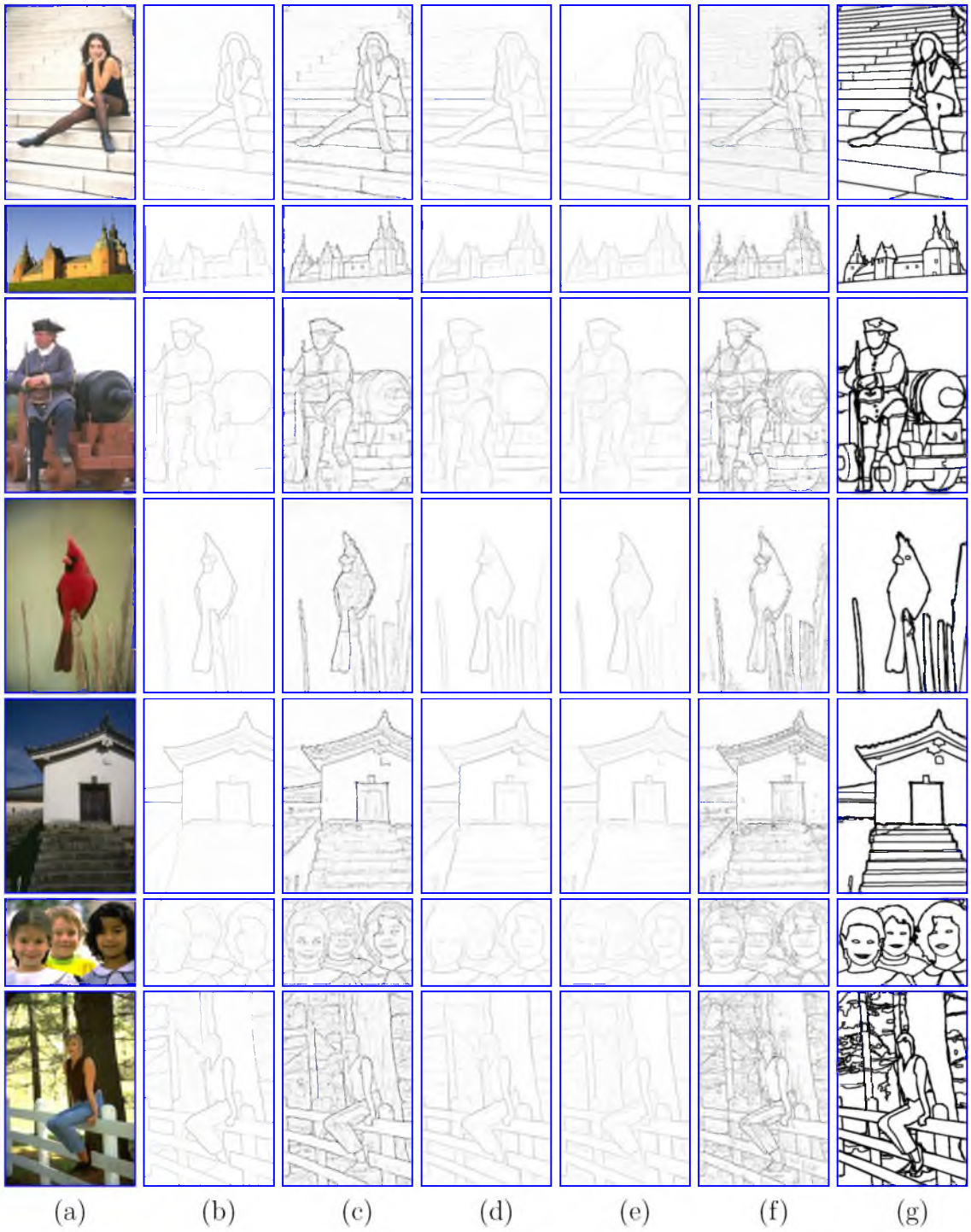


Figure 3.7: Test samples of edge detection on BSDS 500 [10] dataset. (a) Input image, (b) gPb-OWT-UCM [78], (c) Sketch tokens [81], (d) SCG [82], (e) SE [83], (f) CHM, (g) Groundtruth. CHM is able to capture finer details like upper stairs in the first row, steeples in the second row, and wheels in the third row.

images) and the remaining 40% for testing (580 images). The images were also resized to 240 × 320 resolution. We evaluated the performance of CHM using RGB and RGBD modalities. For the depth channel, we computed the same set of features that we extract from the RGB color channels. In Table 3.4, we compare CHM with SCG [82] and SE [83]. CHM performs significantly better than other methods and reaches an F-value of 0.649 for RGB and 0.678 for RGBD. The precision-recall curves are shown in Fig. 3.8 and qualitative comparisons are shown in Fig. 3.9.

Table 3.4: Testing performance of different methods on NYU depth dataset [99] using RGB (top), and RGBD (bottom) modalities. CHM achieves state-of-the-art performance for both cases.

Method	ODS	OIS	AP
SCG [82] (RGB)	0.557	0.569	0.438
SE [83] (RGB)	0.596	0.608	0.541
CHM (RGB)	**0.649**	**0.661**	**0.625**
SCG [82] (RGBD)	0.621	0.632	0.534
SE [83] (RGBD)	0.636	0.647	0.601
CHM (RGBD)	**0.678**	**0.690**	**0.665**

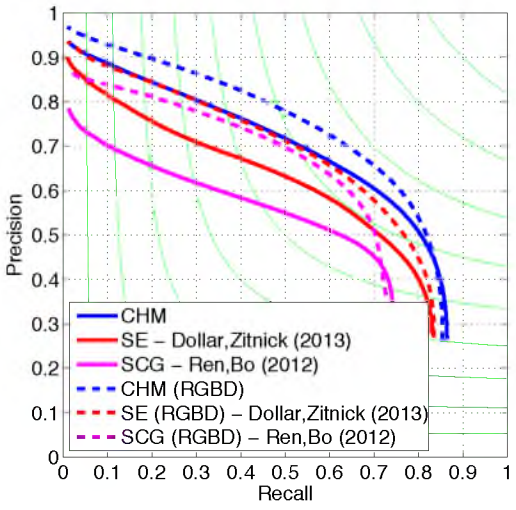


Figure 3.8: Precision-recall curves of different methods for NYU depth dataset [99] using RGB (solid lines) and RGBD(dashed lines) modalities.

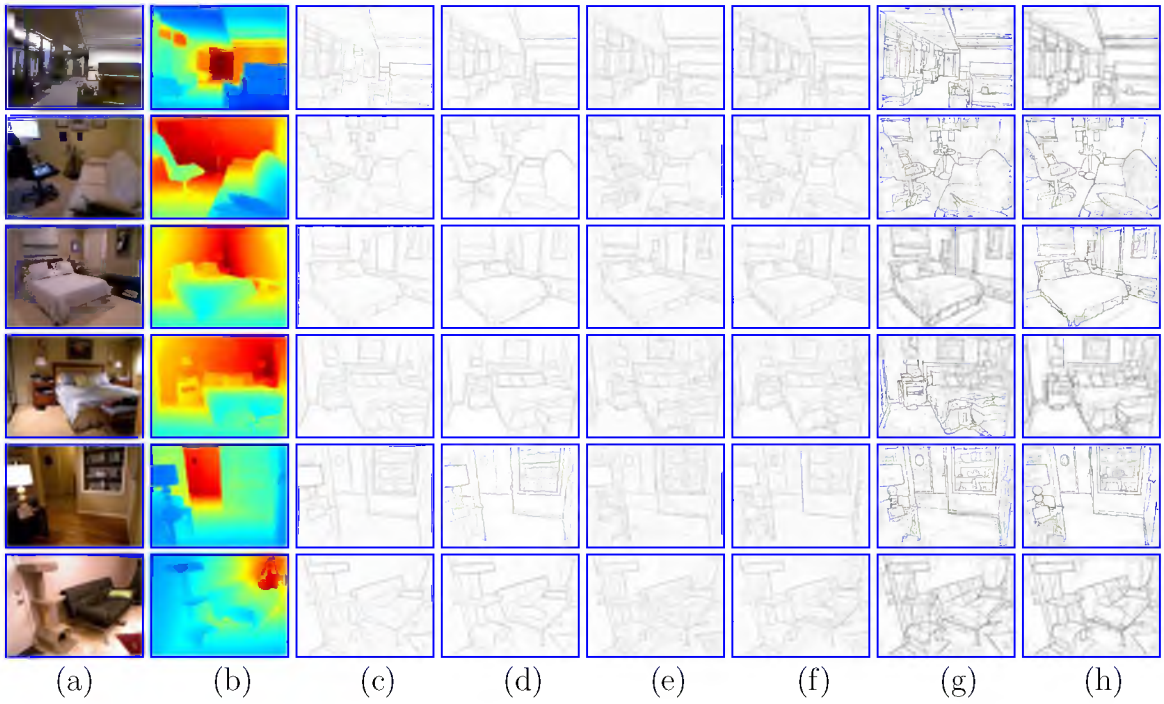


Figure 3.9: Test samples of edge detection on NYU depth (v2) dataset [99]. (a) Input image, (b) Depth image, (c) SCG (RGB) [82], (d) SCG (RGBD) [82], (e) SE (RGB) [83], (f) SE (RGBD) [83], (g) CHM (RGB), and (h) CHM (RGBD).

3.3.2.3 Cross-dataset generalization

Inspired by the work of Dollar and Zitnick [83], we performed a set of experiments to examine the generalization performance of CHM in comparison to other learning-based methods. We used the trained CHM on BSDS 500 dataset and ran it on NYU depth dataset for RGB modality. The authors of sketch tokens [81], SCG [82], and SE [83] have provided their models for BSDS 500 dataset, so we could run the same experiment for their methods. The performance metrics for different methods are reported in Table 3.5 and corresponding precision-recall curves are shown in Fig. 3.10.

CHM performs significantly better than other methods. Note that all methods perform about the same on BSDS 500 dataset (Table 3.3). We believe this asserts that our CHM can be used as a general edge detection technique.

3.3.3 Biomedical image segmentation

In the last set of experiments, we applied CHM to the membrane detection problem in electron microscopy (EM) images. This is a challenging problem because of the

Table 3.5: Testing performance of different methods on NYU depth dataset [99] using BSDS 500 dataset [10] for training. CHM outperforms other learning-based approaches significantly.

Method	ODS	OIS	AP
Sketch Tokens [81]	0.567	0.581	0.490
SCG [82]	0.568	0.579	0.441
SE [83]	0.552	0.566	0.462
CHM	**0.595**	**0.606**	**0.528**

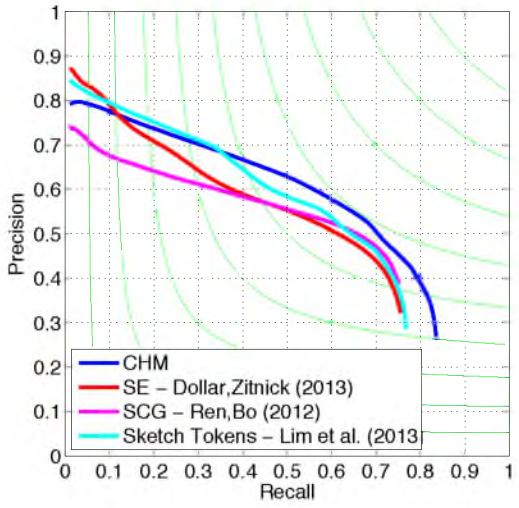


Figure 3.10: Precision-recall curves of different methods for NYU depth dataset [99] using BSDS 500 dataset [10] for training. Cross-dataset generalization performance of CHM is better compared to other methods.

noisy texture, complex intracellular structures, and similar local appearances among different objects [18, 19]. In these experiments, we used a CHM with 2 stages and 5 levels per stage. A 24×24 LDNN was used as the classifier. In addition to the feature set described in section 6.2.2, we included Radon-like features (RLF) [47], which proved to be informative for membrane detection.

3.3.4 Mouse neuropil dataset

This dataset is a stack of 70 images from the mouse neuropil acquired using serial block face scanning electron microscopy (SBFSEM [12]). It has a resolution of $10 \times 10 \times 50$ nm/pixel and each $2D$ image is 700 by 700 pixels. An expert anatomist

annotated membranes, *i.e.,* cell boundaries, in these images. From those 70 images, 14 images were randomly selected and used for training and the 56 remaining images were used for testing. The task is to detect membranes in each $2D$ section.

Since the task is detecting the boundary of cells, we compared our method with two general boundary detection methods, gPb-OWT-UCM (global probability of boundary followed by the oriented watershed transform and ultrametric contour maps) [44] and boosted edge learning (BEL) [43]. The testing results for different methods are given in Table 3.6. The CHM-LDNN outperforms the other methods with a notably large margin.

A few examples of the test images and corresponding membrane detection results using different methods are shown in Fig. 3.11. As shown in our results, the CHM outperforms MSANN in removing undesired parts from the background and closing some gaps.

3.3.5 Drosophila VNC dataset

This dataset contains 30 images from Drosophila first instar larva ventral nerve cord (VNC) [51, 52] acquired using serial-section transmission electron microscopy (ssTEM [17, 53]). Each image is 512 by 512 pixels and the resolution is $4 \times 4 \times 50$ nm/pixel. The membranes are marked by a human expert in each image. We used 15 images for training and 15 images for testing. The testing performance for different methods are reported in Table 3.6. It can be seen that the CHM outperforms the other methods in terms of pixel error. A few test samples and membrane detection

Table 3.6: Testing performance of different methods for the mouse neuropil and Drosophila VNC datasets.

	Mouse neuropil		Drosophila VNC	
Method	F-value	G-mean	F-value	G-mean
gPb-OWT -UCM [44]	45.68%	64.75%	49.90%	69.57%
BEL [43]	71.68%	84.46%	70.21%	84.20%
MSANN [23]	81.99%	90.48%	78.89%	88.74%
CHM	**86.00%**	**92.48%**	**80.72%**	**90.02%**

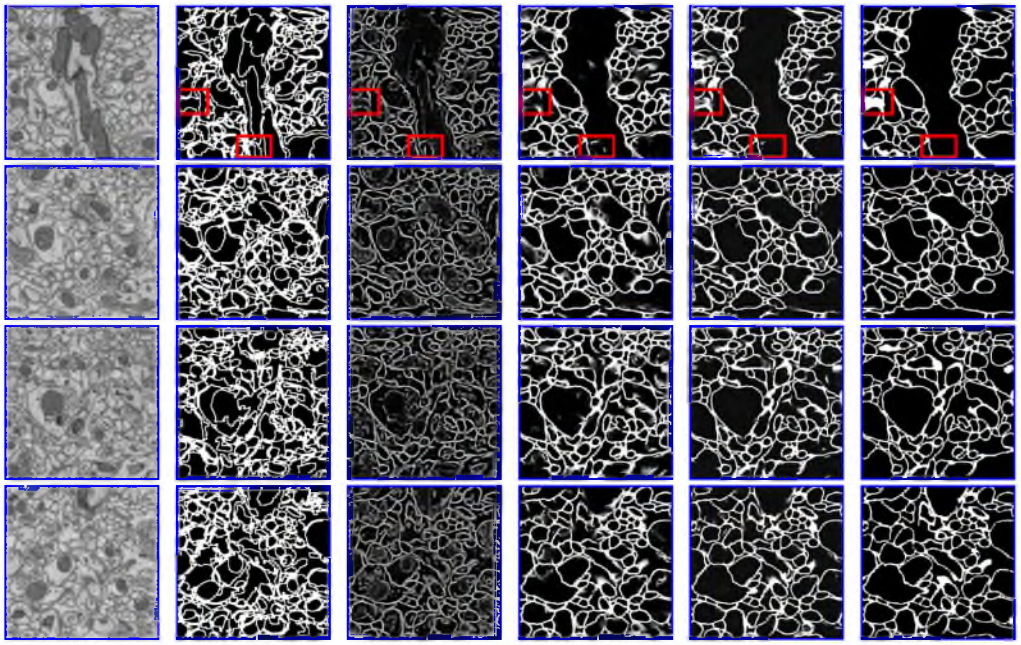


Figure 3.11: Test results of the mouse neuropil dataset. (a) Input image, (b) gPb-OWT-UCM [44], (c) BEL [43], (d) MSANN [23], (e) CHM-LDNN, (f) ground truth images. The CHM is more successful in removing undesired parts and closing small gaps. Some of the improvements are marked with red rectangles. For gPb-OWT-UCM method, the best threshold was picked and the edges were dilated to the true membrane thickness.

results for different methods are shown in Fig. 3.12.

The same dataset was used as the training set for the ISBI 2012 EM challenge [77]. The participants were asked to submit the results on a different test set (the same size as the training set) to the challenge server. We trained the same model on the whole 30 images and submitted the results for the testing volume to the challenge server [77]. The pixel error (1−F-value) of different methods are reported in Table 3.7. CHM achieved pixel error of 0.063, which is better than the human error, *i.e.,* how much a second human labeling differed from the first one. It also outperformed the convolutional networks proposed in [76] and [18]. It is noteworthy that CHM is significantly faster than deep neural networks (DNN) [76] at training. While DNN needs 85 hours on GPU for training, CHM only needs 30 hours on CPU.

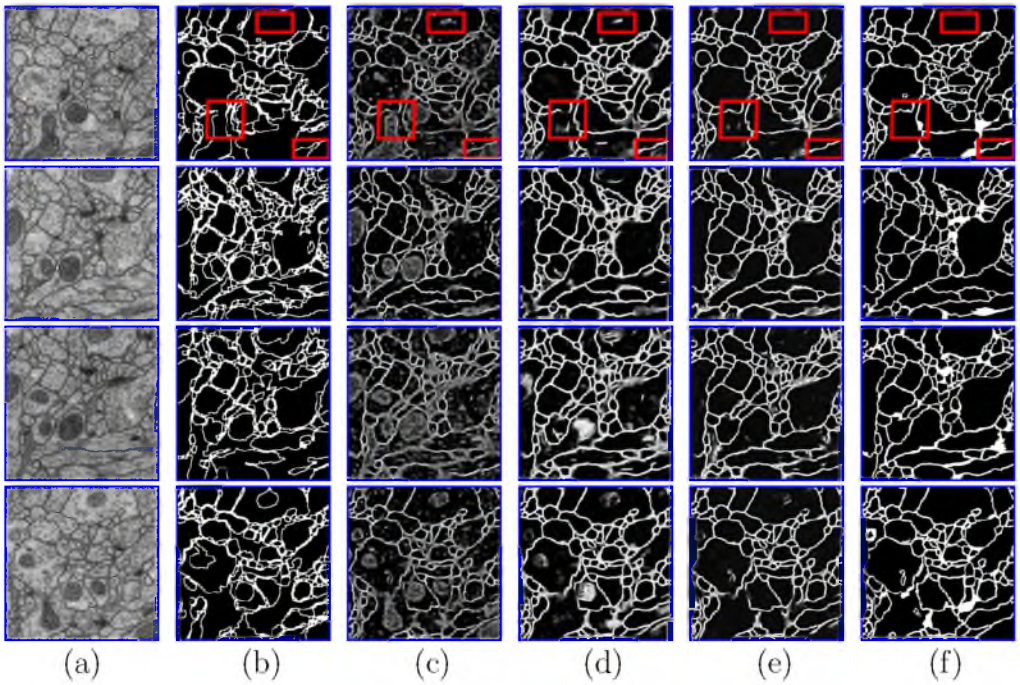


Figure 3.12: Test results of the Drosophila VNC dataset (second row). (a) Input image, (b) gPb-OWT-UCM [44], (c) BEL [43], (d) MSANN [23], (e) CHM, (f) ground truth images. The CHM is more successful in removing undesired parts and closing small gaps. Some of the improvements are marked with red rectangles. For gPb-OWT-UCM method, the best threshold was picked and the edges were dilated to the true membrane thickness.

Table 3.7: Pixel error (1−F-value) and training time (hours) of different methods on ISBI challenge [77] test set. Numbers are available on the challenge leader board.

Method	1−F-value	Training Time
Laptev *et al.* [100]	0.067	–
Convolutional Networks [18]	0.067	–
Human	0.066	–
Deep Neural Networks [76]	0.065	85(GPU)
CHM	**0.063**	30(CPU)

3.4 Conclusion

We develop a discriminative learning scheme for scene labeling, called CHM, which takes advantage of contextual information at multiple resolutions in a hierarchy. The main advantage of CHM is its ability to optimize a posterior probability at multiple resolutions. To our knowledge, this is the first time that a posterior at multiple resolutions is optimized for scene labeling. CHM performs this optimization efficiently in a greedy manner. To achieve this goal, CHM trains several classifiers at multiple resolutions and leverages the obtained results for learning a classifier at the original resolution. We applied our model to several challenging datasets for scene labeling, edge detection, and biomedical image segmentation. Results indicate that CHM achieves state-of-the-art performance on all of these applications.

An important characteristic of CHM is that it is only based on patch information and does not make use of any exemplars or shape models. This enables CHM to serve as a general labeling method with high accuracy. The other advantage of CHM is its simple training. Even though our model needs to learn hundreds of parameters, the training remains tractable since classifiers are trained one at a time separately.

We conclude by discussing a possible extension of the CHM. Even though CHM is able to model global contextual information within a scene, it can be prone to error due to absence of any global constrains. Therefore, CHM can be used as a first step in a scene labeling pipeline. Postprocessing such as CRF proposed in [2] can be used to enforce label consistency and global constraints

CHAPTER 4

FAST ADABOOST TRAINING USING WEIGHTED NOVELTY SELECTION

Training time can be a bottleneck in our contextual models. A fast classifier that can be trained on large datasets would reduce the computational complexity of our contextual models. Moreover, it can be useful for general classification problems. In this chapter, a new AdaBoost learning framework, called WNS-AdaBoost, is proposed for training discriminative models. The proposed approach significantly speeds up the learning process of adaptive boosting (AdaBoost) by reducing the number of data points. For this purpose, we introduce the weighted novelty selection (WNS) sampling strategy and combine it with AdaBoost to obtain an efficient and fast learning algorithm. WNS selects a representative subset of data thereby reducing the number of data points onto which AdaBoost is applied. In addition, WNS associates a weight with each selected data point such that the weighted subset approximates the distribution of all the training data. This ensures that AdaBoost can trained efficiently and with minimal loss of accuracy. The performance of WNS-AdaBoost is first demonstrated in a classification task. Then, WNS is employed in a probabilistic boosting-tree (PBT) structure for image segmentation. Results in these two applications show that the training time using WNS-AdaBoost is greatly reduced at the cost of only a few percent in accuracy.

4.1 Introduction

Boosting is a general learning concept to train a single strong learner by combining a set of weak learners [101]. Based on this concept, many methods have been proposed in the literature to solve several problems, such as classification [86], clustering [102], recognition [103], etc. The first practical polynomial-time boosting algorithm was

developed by Schapire in 1990 [104]. And, in 1995, Freund and Schapire proposed AdaBoost [105]. AdaBoost learns a strong classifier by linearly combining a set of weak classifiers. In order to focus the learning on the most difficult samples, it uses a sample weighting strategy. After the addition of each weak classifier, a sample weight is updated indicating the importance of the sample for classification by subsequent weak classifiers. The weights of the misclassified samples are increased, and the weights of correctly classified samples are decreased to lead the new weak classifiers to focus on the more difficult samples. Despite the simplicity of the method, AdaBoost has been shown to achieve good bounds on its training and generalization error [106].

Following the success of AdaBoost, several related approaches have been proposed. These include Gentle AdaBoost (GAB) [107], FloatBoost [103], robust alternating AdaBoost (RAAB) [108], Modest AdaBoost [109], AdaTree [110], and probabilistic boosting tree (PBT) [86], among others. GAB uses Newton's algorithm instead of greedy steps for optimization, thereby improving generalization performance [107]. FloatBoost also improves the generalization performance by backtracing and pruning previously learnt weak classifiers deemed irrelevant. RAAB and modest AdaBoost use modified loss functions to reduce the effect of outliers in the optimization. AdaTree and PBT both use a tree structure and implement AdaBoost as the classifier on the nodes. The advantage of using a tree structure is that by focusing the training of AdaBoost on simpler subproblems, training of the classifiers is simpler and requires less iterations. This makes AdaTree and PBT particularly useful for learning with larger datasets. Another significant advantage of PBT is that it provides an approach for learning the posterior distribution, which is useful in vision problems.

Although the AdaBoost learning algorithm is considered to be computationally efficient, training can be time consuming in some cases. An obvious example, is that of learning from large datasets, and this issue can aggravated depending on the learning complexity of the weak classifier. Convergence of the learning algorithm can also be slow for problems with a very complex decision boundary. In those problems the first weak classifiers influence the reweighting process, making it difficult for later weak classifiers to focus exclusively on the harder examples. As mentioned earlier, AdaTree and PBT are helpful in these cases because the strong classifier is obtained

by combining simpler AdaBoost classifiers and because classifiers in lower levels of the tree learn from subsets of the original data [86,110]. The tree structure is also helpful in the testing phase because the classification of some samples can be obtained without fully traversing the tree. Another approach for fast training proposed in the literature involves resampling the original data according to the distribution weights [111].

In this chapter, we propose a new learning framework which speeds up the training of AdaBoost and any other boosting based algorithms, including all of the aforementioned methods. For this purpose, we introduce a novel sampling strategy, weighted novelty selection (WNS), and combine it with AdaBoost to obtain the WNS-AdaBoost framework (Figure 4.1). WNS is a sampling method that reduces the number of data points by selecting representative points from the dataset. It also determines a corresponding weight for each of these selected points, which shows the importance of that point and aims at preserving the distribution of the original data. By reducing the number of training samples, the proposed framework significantly reduces the training time. The output of the WNS algorithm is then used by AdaBoost, or any of its variants, to learn a discriminative model. This is achieved by training AdaBoost on the representative set of data points and initializing the weight distribution with the weights obtained from WNS after normalization.

4.2 Weighted Novelty Selection

Weighted novelty selection is the preprocessing sampling method in the WNS-AdaBoost framework. The main idea is to provide the boosting algorithm with a *concise* summary of the training dataset such that the learning algorithm can quickly and efficiently train the classifier. WNS achieves this by selecting representative points from the training dataset and by deriving a corresponding set of weights such

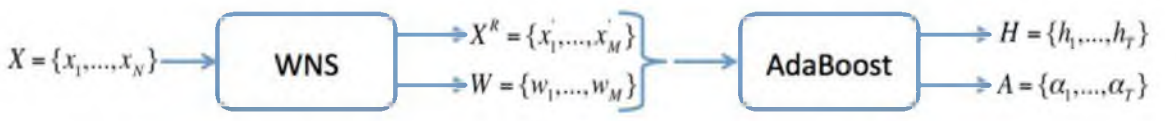


Figure 4.1: Illustration of the WNS-AdaBoost training model. X is the set of input data points, X^R and W denote the representative set and corresponding weights, and H and A represent the set of weak classifiers and corresponding combination factor that determine the classifier.

that the two pieces of information summarize the original data distribution.

WNS was inspired by Platt's work on resource-allocating networks [112]. Platt introduced a criterion to decide whether a given input point should be added to a growing radial basis function neural network in order to minimize network error. The point was added if the distance to the other points already in the network was larger than a threshold and the network error was above another threshold. Fundamentally, Platt's criterion aims to select a reduced set of data points that preserves the data structure relevant to the reduction of the modeling error.

Similarly, WNS picks a data point as a representative point if the smallest distance to all previous representative points is larger than a threshold δ. Hence, δ is a parameter that indirectly controls the number of representative points. Smaller δ increases the number of representative points and vice versa. This procedure ensures that enough points are picked to cover the whole space while keeping the number of them to a minimum.

The set of representative points provides a limited characterization, however, because it does not accurately reflect the density of the input data points. For example, in a classification problem with classes separated by a low density region, one wants to place the decision boundary between the two classes to minimize the error. However, the representative set alone would fail to properly provide this information, unless the clusters are clearly separated and the separation is larger than $\delta/2$. Hence, to more accurately capture the structure of the original training dataset, WNS associates a weight to each representative point. This weight corresponds to the number of input data points assigned to a representative data point, which captures information about the data distribution. Intuitively, since the weight states how many data points are summarized by a representative point, one can think that representative points with larger weights correspond to areas with higher density and thus are more relevant.

The WNS sampling strategy is quite simple and follows directly the ideas described above. Consider a set of N input data points $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, and denote the representative set by $X^R = \{\mathbf{x}'_j\}$ and the corresponding set of weights by $W = \{w_j\}$, $j = 1, 2, \ldots$. In addition, denote by $I_X = \{j_1, \ldots, j_N\}$ the indices of the representative

points in X^R for each $\mathbf{x}_i \in X$, such that $\mathbf{x}_i \in X$ is represented by $x'_{j_i} \in X^R$.

Accordingly, the WNS sampling algorithm proceeds as follows:

1. Initialization: set $X^R = \{\mathbf{x}_1\}$, $W = \{1\}$, $I_X = \{1, 0, \ldots, 0\}$, and $Y = \emptyset$;
2. For each $\mathbf{x}_i \in X \setminus X^R$,
   (a) Compute the distance of $\mathbf{x}_i$ to all $\mathbf{x}'_l \in X^R$, $d(\mathbf{x}_i, \mathbf{x}'_l)$;
   (b) Find $n = \underset{l}{\operatorname{argmin}}\, d(\mathbf{x}_i, \mathbf{x}'_l)$;
   (c) If $d(\mathbf{x}_i, \mathbf{x}'_n) > \delta$,
   Add the point to X^R and set the corresponding weight in W to 1;
   else if $d(\mathbf{x}_i, \mathbf{x}'_n) \leq \delta/2$,
   Set $j_i = n$ and $w_n = w_n + 1$;
   else
   Add $\mathbf{x}_i$ to Y.
3. For each $\mathbf{x}_i \in Y$,
   (a) Compute the distance of $\mathbf{x}_i$ to all $\mathbf{x}'_l \in X^R$, $d(\mathbf{x}_i, \mathbf{x}'_l)$;
   (b) Find $n = \arg\min_l d(\mathbf{x}_i, \mathbf{x}'_l)$;
   (c) Set $j_i = n$ and $w_n = w_n + 1$.

Note that even though the algorithm depicted here uses distances, the algorithm can be readily adapted to use similarities. This can be obtained simply by inverting the inequality comparisons. For dissimilarities, the distance metric can be relaxed to a semimetric (which does not verify the triangle inequality) without affecting the outcome.

Computationally, the WNS sampling algorithm is fast and efficient since the algorithm proceeds in a single pass through the data. The computational complexity is $O(NM)$, where N is the number of points in the original dataset and M is the number of points in the representative set. Although in theory it is possible for the computationally complexity to be $O(N^2)$, this corresponds to the limiting case $\delta \to 0$, in which case the representative set equals the input data. Typically, M is

much smaller than N. An additional advantage is that the WNS algorithm can be easily parallelized.

It is noteworthy that WNS is conceptually similar to the weighted Nyström approach proposed for kernel methods by Zhang and Kwok [113]. Both methods provide a weighted sampling strategy for summarizing the dataset. The weighting have slightly different roles, however. In weighted Nyström, the weights are utilized to approximate the computation on large kernel systems by compressing the kernel matrix and expanding the eigendecomposition. In contrast, WNS explicitly summarizes the data distribution and passes that information directly to the boosting algorithm using the weight data distribution.

Clearly, many density-preserving data characterization methods exist in the literature, including mixture models [114,115], mean shift [116], vector quantization [117], etc. However, these methods have typically several data-dependent parameters that need to be carefully set. In comparison, WNS has only one parameter, δ, which is largely independent of the data if the data range is normalized. Moreover, WNS makes no assumptions on the data distribution.

We propose to employ WNS to speed up AdaBoost. In this regard, WNS is applied to each of the classes to reduce the number of points in them. Afterward, the selected points and corresponding weights are passed to AdaBoost. In addition to reducing the number of points, which can speed up any boosting algorithm, in this framework AdaBoost can take the advantage of the corresponding weights. In other words, not only the reduction of the number of data points improves the speed of the AdaBoost but also the corresponding weights help to keep the performance of the AdaBoost at reasonable rate. So unlike to the usual method that the weights are the same for all the input points, this time each input point has its specific weight which shows how important it is.

4.3 WNS-AdaBoost

The AdaBoost algorithm learns a strong classifier by linearly combining (simpler) weak classifiers according to,

$$H(x) = \sum_t \alpha_t h_t(x) \tag{4.1}$$

where $h_t(x)$ denotes a weak classifier. Different weak classifiers can be used in this framework. The key contribution of AdaBoost is to use a distribution, *i.e.,* a set of weights, over the training samples. These weights are updated adaptively at each iteration of AdaBoost and play an important role in determining the combination factor for each weak classifier, *i.e.,* $\{\alpha_t\}$ in (4.1).

At each iteration, AdaBoost selects the weak classifier that minimizes the weighted error,

$$\epsilon_t = \sum_i w_i [h_t(x_i) \neq y_i] \tag{4.2}$$

where w_i is the sample weight and y_i denotes the desired output for input x_i. This error is calculated with respect to the weights w_i on which the weak classifier is trained. The vote weight of each classifier is computed using this error

$$\alpha_t = \frac{1}{2} \ln(\frac{1-\epsilon_t}{\epsilon_t}) \tag{4.3}$$

Accordingly, the weights are updated with

$$w_i^{t+1} = \frac{w_i^t \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \tag{4.4}$$

where w_i^t denotes the weight of training sample x_i at iteration t and Z_t is a normalization factor, which is chosen so that w^{t+1} will be a probability distribution. This update rule increases the weights of the samples which are difficult to classify and decreases the weights of the samples that are easy to classify, so the next weak classifier focuses on the more difficult samples.

Typically, the weights are initialized uniformly because prior knowledge about the importance of the training samples is not available. Put differently, AdaBoost is left to infer the distribution of the samples solely based on their relative amount. In the proposed method, however, WNS is used to explicitly capture and summarize the data distribution using a reduced number of training samples. This information is they transferred to AdaBoost by setting the weights according to the values obtained from WNS. Hence, the weights are no longer initialized uniformly, and each representative point has its own weight which can be different from the other points. Using this strategy, instead of a large number of training points with the same weights, AdaBoost

is given a smaller number of training points together with prior information about the importance of them. To summarize, WNS speeds up AdaBoost in the training stage by reducing the number of training samples and maintains also the performance of AdaBoost at a good level by providing prior information about the importance of the selected representative points.

Given a training set $X = \{x_1, ..., x_N\}$ and the corresponding labels $L = \{l_1, ..., l_N\}$, $l_i \in \{-1, 1\}$, the WNS-AdaBoost training algorithm proceeds as follows:

1. Separate the classes and make two sets: $X_1 = \{x_i | l_i = -1\}$, $X_2 = \{x_i | l_i = 1\}$.
2. Choose a δ and run WNS for X_1 and X_2. The output of WNS, *i.e.,* representative points and weights for each class are $X_1 \rightarrow (X_1^R, W_1)$, $X_2 \rightarrow (X_2^R, W_2)$.
3. Construct a new training set $X^R = \{X_1^R, X_2^R\}$ and $W = \{W_1, W_2\}$.
4. Normalize W so it will be a probability distribution.
5. Use X^R, W to train AdaBoost classifier.

Although we described the WNS-AdaBoost for training the AdaBoost, one can notice that it can be used also in other AdaBoost based frameworks, *e.g.,* PBT, AdaTree, etc. This generalization can be described by considering the WNS as a preprocessing step. In other words, WNS gets the training set and provides a new training set with corresponding weights.

4.4 Experimental Results

We illustrate the performance of WNS-AdaBoost in terms of accuracy and speed on two different problems: Poker hand classification and texture segmentation. In the first experiment we verify the effectiveness of WNS-AdaBoost in a simple AdaBoost structure, while in the second experiment we show its performance in the probabilistic boosting tree (PBT) framework.

4.4.1 Poker hand classification

The poker hand dataset is available from the UCI Machine Learning Repository [118]. The dataset contains 25010 data points for training and 1000000 data

points for testing distributed over 11 classes. This dataset was used in a two-class form where the first class represents the hands that are not a recognized poker hand, and the second class contains the poker hands from one pair to royal flush. The size of the feature vector is ten, *i.e.,* suit and rank for each card. A decision tree with 7 nodes was used as the weak classifier and boosting was run for 600 iterations.

Table 4.1 shows the classifier training time and its accuracy for different parameter values of δ. As we can see by using $\delta = 3$ the WNS-AdaBoost is more than two times faster than the conventional AdaBoost algorithm while its accuracy is 4% less than the AdaBoost. As δ decreases the performance improves at the cost of speed, *e.g.,* WNS-AdaBoost with parameter $\delta = 2.7$ performs almost the same as AdaBoost while it is 1.3 times faster than AdaBoost.

4.4.2 Texture segmentation

In order to show that proposed method is not restricted to regular AdaBoost algorithm and can be used in any AdaBoost based classifier, we adopted the probabilistic boosting tree (PBT) [86] together with WNS-AdaBoost for texture segmentation. The PBT learns a discriminative model in a hierarchical structure. At each level of the hierarchy, PBT learns some AdaBoost classifiers and use them to split data to smaller groups. The details can be found in [86]. In our experiment the depth of the tree in PBT is two, and we used a decision tree with five nodes as our weak classifier.

The dataset used in this experiment contains 20 star images generated from five different textures for foreground and four different textures for background using textures from Brodatz database [119]. Eight of these images were used for training

Table 4.1: Training time and performance for the "poker hand" dataset.

Method	δ	No. of training samples	Time for applying WNS (s)	Training time	Training error	Testing error	Speedup
WNS-AdaBoost	3	13396	8.85	135s	13%	20%	2.23
WNS-AdaBoost	2.7	18278	10.05	236.10s	11%	17%	1.3
AdaBoost	–	25010	–	320.19s	10.3%	16%	–

and the remaining 12 images were used for testing. The input feature vector to the PBT classifier was formed by sampling the input image at every pixel using an 11×11 stencil (Fig. 4.2). The size of the feature vector is 41.

The classifier training time for PBT and WNS-PBT with different parameter values of δ is shown in Table 4.2. One can notice that there is a trade-off between the accuracy and the speed of the classifier. In other words, we can make the classifier faster at the cost of accuracy. In this experiment it seems that $\delta = .7$ is a reasonable choice that makes the classifier much faster at the cost of few percents decreasing in the accuracy. The ROC curves for training and testing images are shown in Fig. 4.3. The accuracy performance of WNS-PBT is close to the PBT while it is much faster. It must be emphasized that speedup in this experiment is much higher compared to the previous experiment due to the size of the dataset, so WNS-AdaBoost is more useful for the large dataset cases. The segmentation results on some test images for WNS-PBT and PBT are shown in Fig. 4.4. The results for WNS-PBT with $\delta = 0.7$ and PBT are shown in the third and fourth columns. The results are really close while the WNS-PBT method is 444 times faster than PBT in training.

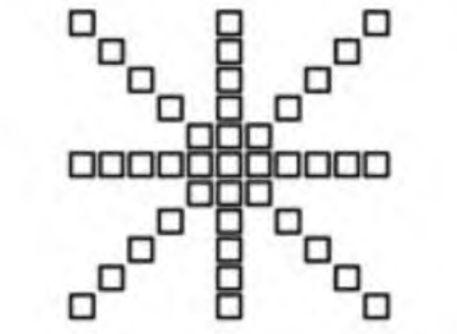
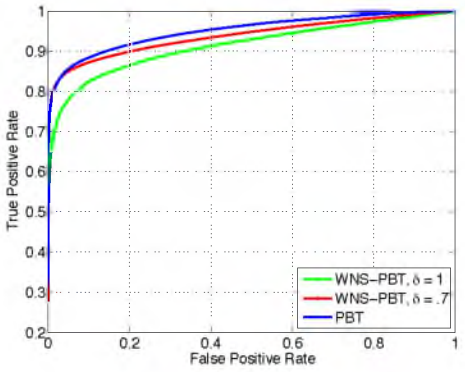


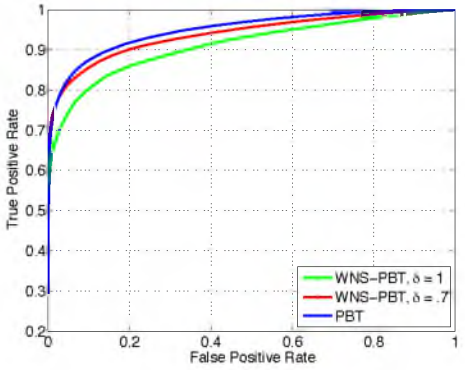
Figure 4.2: The stencil which is used to sample the input image.

Table 4.2: Training time for the "texture segmentation" experiment.

Method	δ	No. of training samples	Time for applying WNS (s)	Training time	Speedup
WNS-PBT	1	1808	108.62	5.82s	38351
WNS-PBT	0.7	27206	11186.96	502.64s	444
PBT	–	524288	–	62hours	–



(a)



(b)

Figure 4.3: ROC curves for the texture segmentation experiment. (a) training, (b) testing.

4.5 Conclusions

We introduced a new framework WNS-AdaBoost for efficient learning of discriminative boosting models. The WNS-AdaBoost framework efficiently selects a reduced set of representative training points, thus reducing the overall computational complexity for training and increasing the speed of the training process. Moreover, by returning the weights for each of representative point, WNS provides a compact representation of the distribution of the training data in a way that is naturally amenable to AdaBoost. The combination of these two characteristics ensure faster training and with minimal loss of accuracy.

The improvement in training speed is achieved potentially at the expense of a small reduction in accuracy. This behavior is regulated by the sampling parameter δ. If δ is increased from zero, the size of the representative training set given to AdaBoost is reduced, thereby increasing the training speed but decreasing the accuracy because of

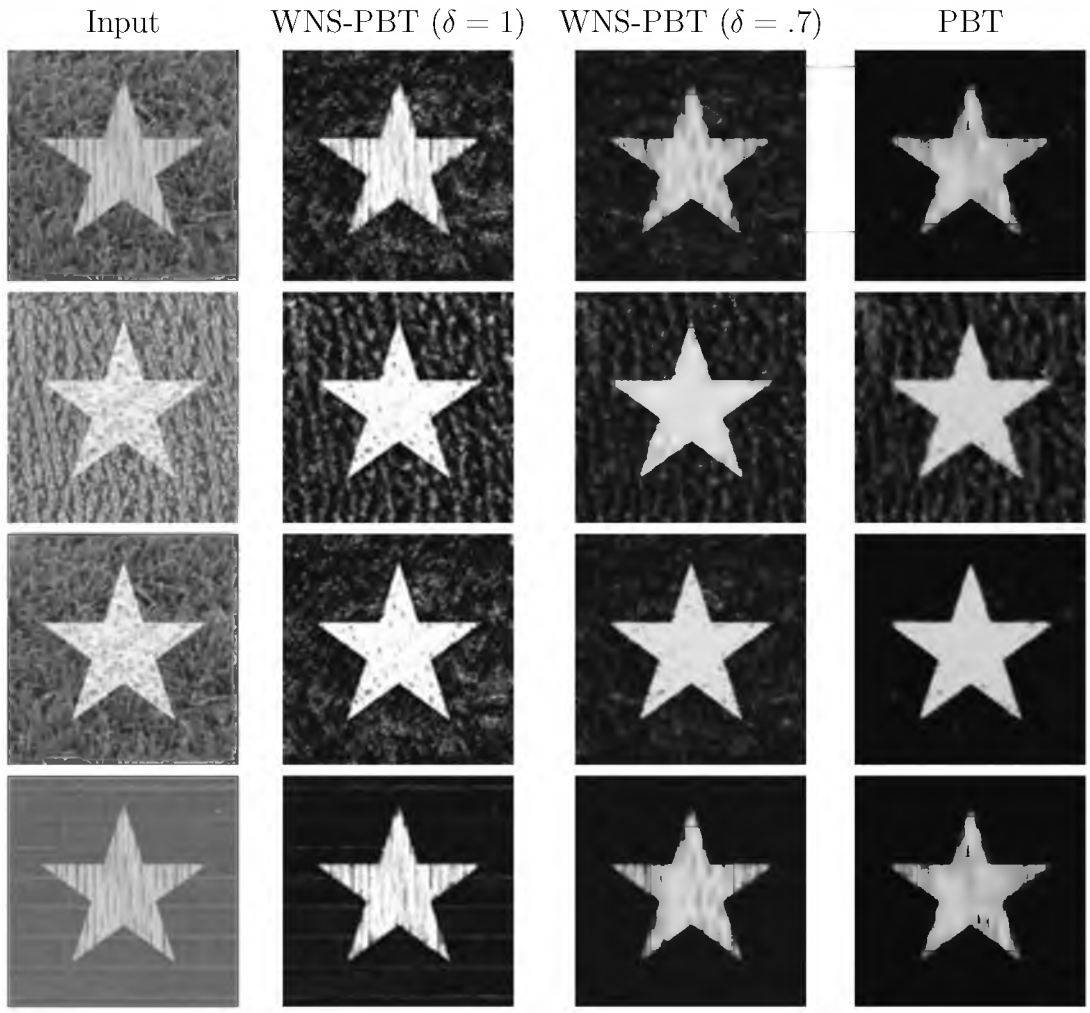


Figure 4.4: Test results for the texture segmentation experiment. The first column shows the input image and the remaining columns show the output of WNS-PBT with $\delta = 1$, WNS-PBT with $\delta = 0.7$, and PBT classifiers respectively.

the increasingly crude representation of the data. Conversely, as δ tends to zero, WNS outputs the original training data, which is equivalent to the direct use of AdaBoost. Still, the experiments show that by appropriately choosing δ, it is possible to achieve large improvements in training speed with negligible loss of accuracy.

It must be emphasized that the WNS-AdaBoost framework extends beyond AdaBoost alone to any other AdaBoost-based classifier. As an example, this generality was explicitly demonstrated in the application of the framework to the PBT classifier. Additionally, it is noted that although the algorithm was described here only for the

two-class case for ease of presentation, the framework is not restricted to this case and can be generalized to multiclass problems in a straightforward way.

CHAPTER 5

DISJUNCTIVE NORMAL RANDOM FORESTS

We develop a novel supervised learning/classification method, called *disjunctive normal random forest (DNRF)*. A DNRF is an ensemble of randomly trained *disjunctive normal decision trees (DNDT)*. To construct a DNDT, we formulate each decision tree in the random forest as a disjunction of rules, which are conjunctions of Boolean functions. We then approximate this disjunction of conjunctions with a differentiable function and approach the learning process as a risk minimization problem that incorporates the classification error into a single global objective function. The minimization problem is solved using gradient descent. DNRFs are able to learn complex decision boundaries and achieve low generalization error. We present experimental results demonstrating the improved performance of DNDTs and DNRFs over conventional decision trees and random forests. We also show the superior performance of DNRFs over state-of-the-art classification methods on benchmark datasets.

5.1 Introduction

Random forests became popular with Breiman's seminal paper [120] in 2001 due to their ease of use and good classification accuracy. The main idea of random forest classification is to grow an ensemble of decision trees such that the correlation between the trees remains as low as possible. This is achieved by injecting randomness into the forest using a different set of training samples for each tree. These sets are obtained by sampling the original training set with replacement, *i.e.*, bagging. Another source of randomness in random forests is the subset of features randomly selected to consider at each node as the splitting function. This parameter can directly control the correlation

between the trees and also affects the accuracy performance of each individual tree. At test time, each tree in the random forest casts a unique vote for the given input and the most popular class among the trees is selected as the predicted label for that input. Random forests have been shown to be effective in many applications like image segmentation/classification [121, 122], object detection [123], and biomedical image analysis [100, 124].

Random forests have certain advantages over other widely used classification algorithms. For instance, support vector machines (SVM) [125] offer good generalization performance due to the fact that they guarantee maximum margin, but choosing the kernel function and the kernel parameters can be time consuming. Boosting [101] is another popular classification approach, which trains a single strong classifier by combining multiple weak classifiers. However, convergence of the learning algorithm can be slow for problems with complex decision boundaries. Artificial neural networks (ANN) [54] are powerful but slow at training due to the computational cost of backpropagation [126]. In addition to all the aforementioned shortcomings of ANNs, SVMs, and boosting methods, these techniques do not naturally handle multiclass problems [127–129]. On the other hand, random forests are fast to train and handle multiclass problems intrinsically [130]. Moreover, they perform consistently well for high dimensional problems [131].

The weak learner used at each node of the decision trees plays an important role in the behavior and performance of random forests. The conventional random forest exploits axis-aligned decision stumps, which partition the feature space with orthogonal hyperplanes. While this type of partitioning can be suitable for certain types of datasets, it results in overfitting and produces "blocky artifacts" in general datasets [130]. It has been shown that using linear discriminants that can be at any arbitrary orientation to the axes improves the performance of random forests [132]. Nonlinear weak learners like conic sections have also been proved successful in increasing the accuracy and generalization performance of random forests [130].

A lot of work has been put into improving the random forest, through the use of more powerful node models and less correlated trees. Rodriguez *et al.* [133] used PCA to make a linear combination of features at each node. Bernard *et al.* [134] focused

on the number of features randomly selected at each node of the tree. They showed that using a random number of features, which can be different at each node, can improve the performance. Tripoliti *et al.* [135] improved the prediction peformance of random forests by modifying the node split function as well as the voting procedure in the forest.

In this chapter, we propose a novel approach for learning linear discriminants of arbitrary orientation at each node of a decision tree. However, the main advantage of our approach over the above-mentioned methods such as [132] and [133] is that it learns all the weak learners of the decision tree in a unified framework. To be clear, unlike conventional decision trees and their variants that learn the splitting function at each node independently, our approach allows weak learners of different nodes to interact with each other during the training because it minimizes a single global objective function. To achieve this goal, we formulate each decision tree as a single disjunction of conjunctions [87] and approximate it with a differentiable function. Next, we use this approximation in a quadratic error cost to construct a single unified objective function. Finally, we minimize this objective function using the gradient descent rule to update the parameters of the discriminants in the decision tree. We call this type of decision tree a disjunctive normal decision tree (DNDT). It is worth mentioning that this formulation is closely related to the formulation of LDNN in Chapter 3 but the number of groups and discriminants per group are determined automatically by decision tree.

Many researchers have proposed converting decision trees into a differentiable form and performing some global parameter tuning to make a smooth decision boundary with high generalization performance. For example [136–139] propose to convert decision trees into artificial neural networks (ANN) and use back-propagation to fine tune the weights and improve the performance. These methods speed up the training of ANNs by using decision trees to initialize the weights of ANNs. However, it would be hard to generalize these methods to random forest framework due to the slowness of back-propagation. Our approach is different from these methods in the sense that unlike the neural networks that have at least two layers of adaptive weights, our disjunctive normal form has only one adaptive layer and thus is faster than

back-propagation. Moreover, we will show that DNDTs outperform ANNs.

Fuzzy/soft decision trees are another technique that have been developed to improve the performance of decision trees. Olaru and Wehenkel [140] build a decision tree by introducing a third state at each node. The samples which fall in the third state go to both children nodes. Using this strategy, a sample might contribute to the final decision through multiple paths. Irsoy *et al.* [141] also propose a soft decision tree that uses a gate function to redirect each sample to all the children with a certain probability. This strategy results in more accurate and simpler trees. The fundamental difference of our approach with soft decision trees is that we propose a global objective function and learn all the splits simultaneously. We will show that DNDTs outperform soft decision trees.

We follow the idea of random forests and use DNDTs as building blocks of a new random forest, called a disjunctive normal random forest (DNRF). While DNRFs have all the advantages of conventional random forests, they outperform them due to their stronger building blocks, *i.e.*, DNDTs. Fig. 5.1 demonstrates the superior performance of DNRF over conventional random forest with artificial examples. We observe that conventional random forest results in box-like decision boundaries and overfits to the training data, while DNRF produces a smooth boundary with lower generalization error. In the results section, we show that, similar to random forests, DNRFs are able to handle multiclass classification problems, but with improved accuracy. We also show that DNRFs outperform state-the-art algorithms such as space partitioning method [142] and multiclass boosting [143].

5.2 Disjunctive Normal Random Forests

The disjunctive normal random forest (DNRF) is a forest of simpler structures called disjunctive normal decision trees (DNDT). DNDT is a special form of decision tree in which different nodes interact with each other during training and are learned simultaneously. In this section, we first describe DNDTs and then show how they can be used in constructing a DNRF. For the sake of simplicity, we consider only the binary classification problem in this section. In the next section, we show how DNRF can be generalized to multiclass problems.

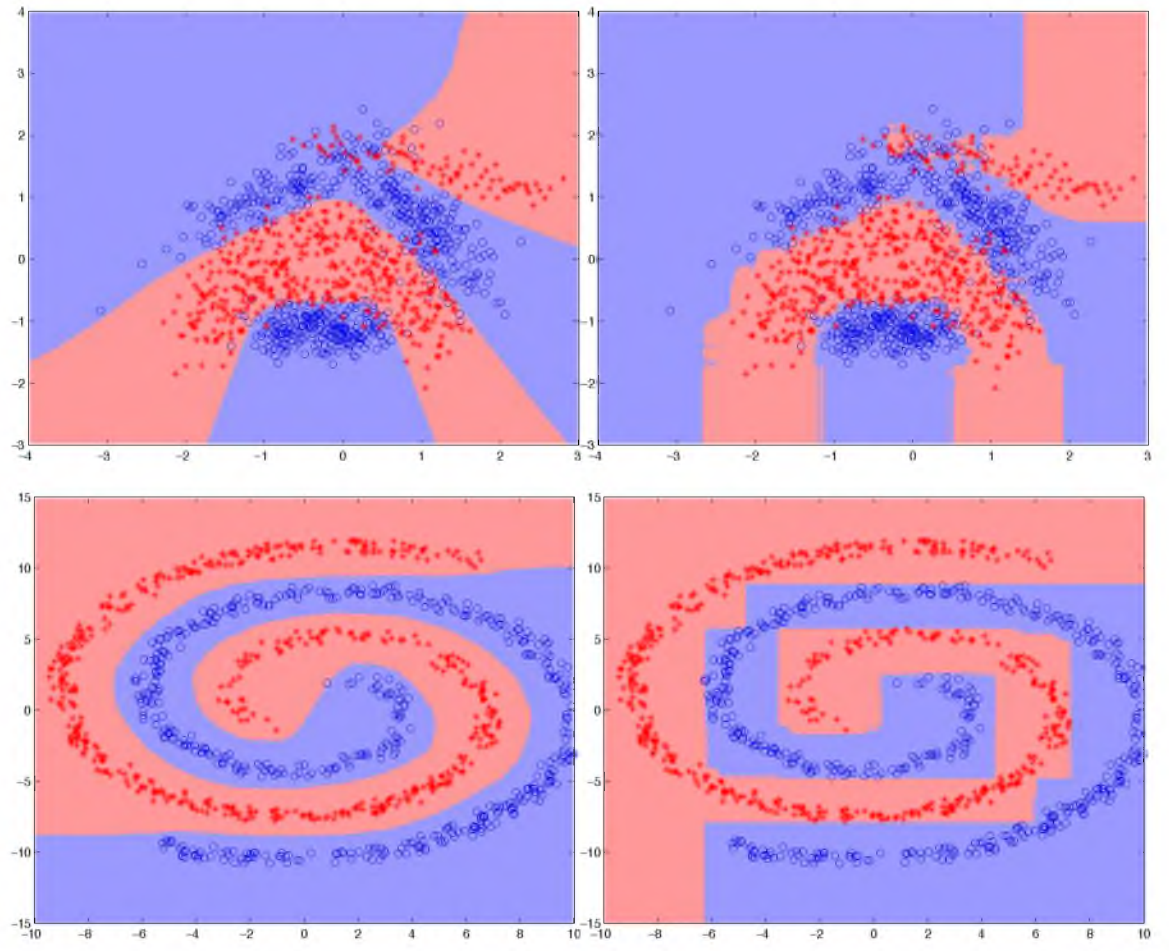


Figure 5.1: Comparison of DNRF (left panel) with random forest (right panel) on the banana dataset [144] (upper panel) and two-spiral dataset (lower panel). DNRF results in a smoother decision boundary and, unlike random forests, does not overtrain.

Notation: Unless specified otherwise, we denote vectors with lower case bold symbols, *e.g.*, $\mathbf{w_k}$, elements of vectors with lower case symbols, *e.g.*, w_{kj}, and sets with calligraphic symbols, *e.g.*, $\mathcal{S}$.

5.2.1 Disjunctive normal decision tree

A decision tree is a set of "rules" organized hierarchically in a tree-like graph [130]. An example is shown in Fig. 5.2. The goal is to predict the label of an input data point based on these rules. During the training, a "split function/rule" is learned at each node of the decision tree. This split function is a binary function, which

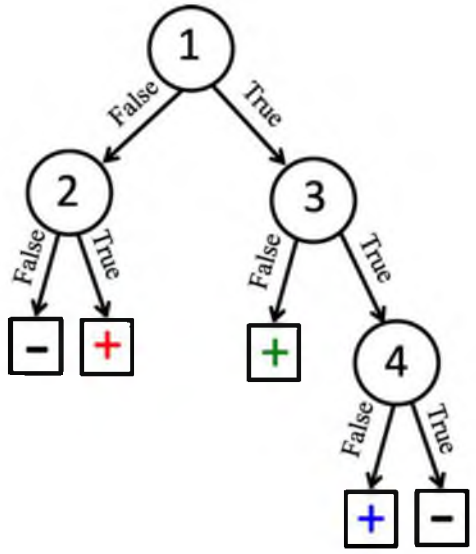


Figure 5.2: An example of a decision tree. Nonleaf nodes are denoted with circles and leaf nodes are denoted with squares. A split function is learned at each nonleaf node. Each leaf node represents a class label, "+" for class "1" and "−" for class "0." The first, second, and third positive leaf nodes are colored in red, green, and blue, respectively.

determines whether incoming data points are sent to the left or right child node. The split function of node k for a d dimensional data point $\mathbf{x}$ can be written as follows:

$$f_k(\mathbf{x}, \mathbf{w_k}) = \mathbb{I}\left(\mathbf{\Phi}\left(\mathbf{x}\right)^T \mathbf{w_k} > 0\right) : \mathbb{R}^{d+1} \times \mathbb{R}^{d+1} \rightarrow \{0, 1\} \tag{5.1}$$

where $\mathbf{w_k}$ is an axis-aligned line (it only has two nonzero elements: the bias and a 1 for the chosen splitting axis), which is learned during the training. $\mathbb{I}(\cdot)$ is the binary indicator function and $\mathbf{\Phi}(\mathbf{x})$ is $[\phi_1, \ldots, \phi_{d+1}]^T = [x_1, \ldots, x_d, 1]^T$. We drop $\mathbf{w_k}$ and use $f_k(\mathbf{x})$ instead of $f_k(\mathbf{x}, \mathbf{w_k})$ for notational simplicity. Each decision tree can be written as a disjunction of conjunctions, which is also known as the disjunctive normal form [87]:

$$h(\mathbf{x}) = \bigvee_{i=1}^{n} \left(\bigwedge_{j \in R_i} f_j(\mathbf{x}) \bigwedge_{j \in L_i} \neg f_j(\mathbf{x}) \right) \tag{5.2}$$

where n is the number of positive leaf nodes, R_i denotes the set of nodes visited from the root to the i^{th} positive leaf node for which the right child is taken on the path, and similarly L_i denotes the set of visited nodes for which the left child is taken on the

same path. For example, for the tree given in Fig. 5.2, $n = 3$, $R_1 = \{2\}$, $L_1 = \{1\}$, $R_2 = \{1\}$, $L_2 = \{3\}$, $R_3 = \{1, 3\}$, $L_3 = \{4\}$, and $h(\mathbf{x})$ can be written as

$$\begin{aligned} h(\mathbf{x}) = (\neg f_1(\mathbf{x}) \wedge f_2(\mathbf{x})) \vee (f_1(\mathbf{x}) \wedge \neg f_3(\mathbf{x})) \\ \vee (f_1(\mathbf{x}) \wedge f_3(\mathbf{x}) \wedge \neg f_4(\mathbf{x})) \end{aligned} \tag{5.3}$$

The data point $\mathbf{x}$ is classified in class "0" if $h(\mathbf{x}) = 0$ and is classified in class "1" if $h(\mathbf{x}) = 1$.

5.2.1.1 The differentiable disjunctive normal form

Once the decision tree is initialized in the conventional manner, we would like to modify (5.2) to be able to fine tune it with gradient descent. The first step is to replace (5.2) with a differentiable approximation. First, any conjunction of binary variables $\bigwedge_i b_i$ can be replaced by their product $\prod_i b_i$. Also, using De Morgan's laws we can replace the disjunction of binary variables $\bigvee_i b_i$ with $\neg \bigwedge_i \neg b_i$, which in turn can be replaced by the expression $1 - \prod_i (1 - b_i)$. Note that we use $(1 - b_i)$ to compute $\neg b_i$. Finally, we can approximate the split function with the logistic sigmoid function:

$$\tilde{f}_k(\mathbf{x}, \mathbf{w_k}) = \frac{1}{1 + e^{-\sum_{j=1}^{d+1} w_{kj} \phi_j}}. \tag{5.4}$$

This gives in the differentiable disjunctive normal form approximation of h

$$\begin{aligned} \tilde{h}(\mathbf{x}) &= 1 - \prod_{i=1}^{n} [1 - \underbrace{\prod_{j \in R_i} (\tilde{f}_j(\mathbf{x}, \mathbf{w_j}) \prod_{j \in L_i} (1 - \tilde{f}_j(\mathbf{x}, \mathbf{w_j}))}_{g_i(\mathbf{x})}] \\ &= 1 - \prod_{i=1}^{n} (1 - g_i(\mathbf{x})) \end{aligned} \tag{5.5}$$

For the example in Fig. 5.2 the approximation of h can be written as

$$\begin{aligned} \tilde{h}(\mathbf{x}) = 1 - &\left(1 - \tilde{f}_2(\mathbf{x}) \left(1 - \tilde{f}_1(\mathbf{x})\right)\right) \\ &\times \left(1 - \tilde{f}_1(\mathbf{x}) \left(1 - \tilde{f}_3(\mathbf{x})\right)\right) \\ &\times \left(1 - \tilde{f}_1(\mathbf{x}) \tilde{f}_3(\mathbf{x}) \left(1 - \tilde{f}_4(\mathbf{x})\right)\right) \end{aligned} \tag{5.6}$$

The next step is to update the weights $\mathbf{w_k}$ to improve the performance of the classifier. Unlike decision trees for which weights at each node are learned separately,

the disjunctive normal form allows us to update all weights simultaneously; therefore, the obtained decision boundary will be smoother and the generalization performance will be higher compared to decision tree.

Given a set of training samples $\mathcal{S} = \{(\mathbf{x_m}, y_m); m = 1, \ldots, M\}$ where $y_m \in \{0, 1\}$ denotes the desired binary class corresponding to $\mathbf{x_m}$, M denotes the number of training samples and a disjunctive normal classifier $\tilde{h}(\mathbf{x})$, the quadratic error over the training set is

$$E(\tilde{h}, \mathcal{S}) = \sum_{m=1}^{M} \Big(y_m - \tilde{h}(\mathbf{x_m})\Big)^2 \tag{5.7}$$

This error function can be minimized using gradient descent. The gradient of the error function with respect to the parameter w_{kj} in the disjunctive normal form, evaluated for the training pair $(\mathbf{x}, y)$, is

$$\begin{aligned}\frac{\partial E}{\partial w_{kj}} = &-2x_j(y - \tilde{h}(\mathbf{x}))\times \\ &\Bigg(\sum_{\substack{l \\ k \in R_l}} \Bigg(\prod_{r \neq l}(1 - g_r(\mathbf{x}))g_l(\mathbf{x})(1 - \tilde{f}_k(\mathbf{x}))\Bigg) \\ &- \sum_{\substack{l \\ k \in L_l}} \Bigg(\prod_{r \neq l}(1 - g_r(\mathbf{x}))g_l(\mathbf{x})\tilde{f}_k(\mathbf{x})\Bigg)\Bigg)\end{aligned} \tag{5.8}$$

The derivation of (5.8) is given in A. At test time, the weights found by gradient descent are used in (5.5) followed by thresholding to predict the label for a new data point.

5.2.2 Decision tree to random forest

Random forests are an ensemble of randomly trained decision trees [120]. The randomness comes from the fact that each tree is trained using a random subset of training samples. Moreover, at each node of tree a random subset of input features are used to learn the split function. The main idea is to make the decision trees as independent as possible. This improves the robustness and generalization of the ensemble.

Using the same idea, we can use an ensemble of DNDT to generate a DNRF. DNRF takes advantage of more powerful DNDTs compared to the conventional random forest

and thus results in better performance. The overall training algorithm for the DNRF is given in Algorithm 4. Note that in the first step a conventional random forest is trained, which allows DNRFs to take advantage of the randomness existing in the random forest.

Algorithm 4 Training algorithm for the DNRF.

Input: Training data, $\mathcal{S} = \{(\mathbf{x_m}, y_m); m = 1, \ldots, M\}$, number of trees, N, ratio of training samples per tree, r, and number of features per node, F.

Output: A set of weights, $\{\mathcal{W}_t, t = 1, \ldots, N\}$.

- Train a conventional random forest with parameters N, r, F.

for $t = 1$ *to* N **do**

- $p \leftarrow$ Number of nodes which are visited to reach positive leaf nodes in tree t.
- Convert tree t to disjunctive normal form using equation (5.5).
- Compute updated weights, $\mathbf{w_1}, \ldots, \mathbf{w_p}$, using gardient descent (equation (5.8)).
- $\mathcal{W}_t \leftarrow \{\mathbf{w_1}, \ldots, \mathbf{w_p}\}$.

end for

At test time, the predicted label for a given data point $\mathbf{x}$ can be computed as follows:

$$\tilde{y} = \mathbb{I}\left(\sum_{t=1}^{N} \mathbb{I}\left(\tilde{h}_t(\mathbf{x}) > 0.5\right) > \frac{N}{2}\right) \tag{5.9}$$

where N denotes the number of trees in DNRF and $\tilde{h}_t(\mathbf{x})$ is computed using the weights $\mathcal{W}_t$ obtained from the training in (5.5).

5.3 Multiclass DNRF

The concept of DNDT can be extended to multiclass problems. In this case, given a single decision tree, instead of binary leaf nodes, *i.e.*, "+" and "−" leaf nodes, there are leaf nodes with labels $1, \ldots, C$, where C is the number of classes. Each tree can be represented with C disjunctive normal functions of type (5.2):

$$h^c(\mathbf{x}) = \bigvee_{i=1}^{n_c} \left(\bigwedge_{j \in R_i^c} f_j(\mathbf{x}) \bigwedge_{j \in L_i^c} \neg f_j(\mathbf{x}) \right), \quad c = 1, \ldots, C \tag{5.10}$$

where n_c denotes the number of leaf nodes with label c and R_i^c, L_i^c are similar to the binary case for the leaf nodes with label c. Each of these $h^c(\mathbf{x})$ then can be converted

to the differentiable form of (5.5). Finally, the weights of each of these functions can be updated using (5.8). Note that, each $\tilde{h}^c(\mathbf{x})$ is updated independently and thus, the update process can be done in parallel. At test time, the label of an input data point $\mathbf{x}$ can be predicted as follows:

$$\tilde{y} = \arg\max_c \tilde{h}^c(\mathbf{x}) \tag{5.11}$$

$$\tilde{h}^c(\mathbf{x}) = 1 - \prod_{i=1}^{n_c} [1 - \prod_{j \in R_i^c} (\tilde{f}_j(\mathbf{x}, \mathbf{w_j^c}) \times \prod_{j \in L_i^c} (1 - \tilde{f}_j(\mathbf{x}, \mathbf{w_j^c}))] \tag{5.12}$$

Note that, in the above equation the updated weights from the training are used to compute $\tilde{h}^c(\mathbf{x})$. It must be emphasized that although different classes share same initial weights, the final updated weights, *i.e.*, $\mathbf{w_j^c}$, can be different since the gradient descent is run for different classes separately.

Similar to the binary case, the multiclass DNDT can be used in a forest structure. The training algorithm for multiclass DNRF is described in Algorithm 5. At test

---

Algorithm 5 Training algorithm for the multiclass DNRF.

Input: Training data, $\mathcal{S} = \{(\mathbf{x_m}, y_m); m = 1, \ldots, M\}$, number of trees, N, ratio of training samples per tree, r, and number of features per node, F.

Output: A set of weights, $\{\mathcal{W}_t^c, \quad t = 1, \ldots, N, \quad c = 1, \ldots, C\}$.

- $C \leftarrow$ Number of classes.
- Train a conventional random forest with parameters N, r, F.

for $t = 1$ *to* N **do**
  for $c = 1$ *to* C **do**

- $p \leftarrow$ Number of nodes which are visited to reach leaf nodes with label c in tree t.
- Form $\tilde{h}_c^t(\mathbf{x})$ in equation (5.12).
- Compute updated weights, $\mathbf{w_1^c}, \ldots, \mathbf{w_p^c}$, by updating $\tilde{h}_c^t(\mathbf{x})$ using gradient descent (equation (5.8)).
- $\mathcal{W}_t^c \leftarrow \{\mathbf{w_1^c}, \ldots, \mathbf{w_p^c}\}$.

  end for
end for

---

time, the label for a given data point is computed using voting among all trees:

$$\tilde{y} = \arg\max_l \sum_{t=1}^{N} \mathbb{I}\left(\arg\max_c \tilde{h}_t^c(\mathbf{x}) = l\right). \tag{5.13}$$

A comparison of DNRF against random forest on the four-spiral dataset is shown in Fig. 5.3. The superior performance of DNRF can be seen in the areas where the spirals end. Furthermore, the decision boundaries are more equidistant to the different classes.

5.4 Experimental Results

We performed experimental studies to evaluate the performance of DNDTs and DNRFs in comparison to different classification techniques. The experiments were performed on both binary and multiclass classification problems. We used six datasets for the binary case and four datasets for the multiclass case from the UCI repository [145] and LIBSVM datasets [146]. Before training, the data were normalized by subtracting the mean of each dimension and dividing by the standard deviation of that dimension.

5.4.1 Binary classification

The six datasets tested for binary classification were Ionosphere ($M_{tr} = 234$, $M_{te} = 117$, $d = 33$), Wisconsin breast cancer ($M_{tr} = 380$, $M_{te} = 189$, $d = 30$), German credit ($M_{tr} = 667$, $M_{te} = 333$, $d = 24$), PIMA Diabetes ($M_{tr} = 513$, $M_{te} = 255$, $d = 8$), Hearts ($M_{tr} = 180$, $M_{te} = 90$, $d = 13$), and IJCNN ($M_{tr} = 49990$, $M_{te} = 91701$, $d = 22$), where M_{tr} is the number of training samples, M_{te} is the

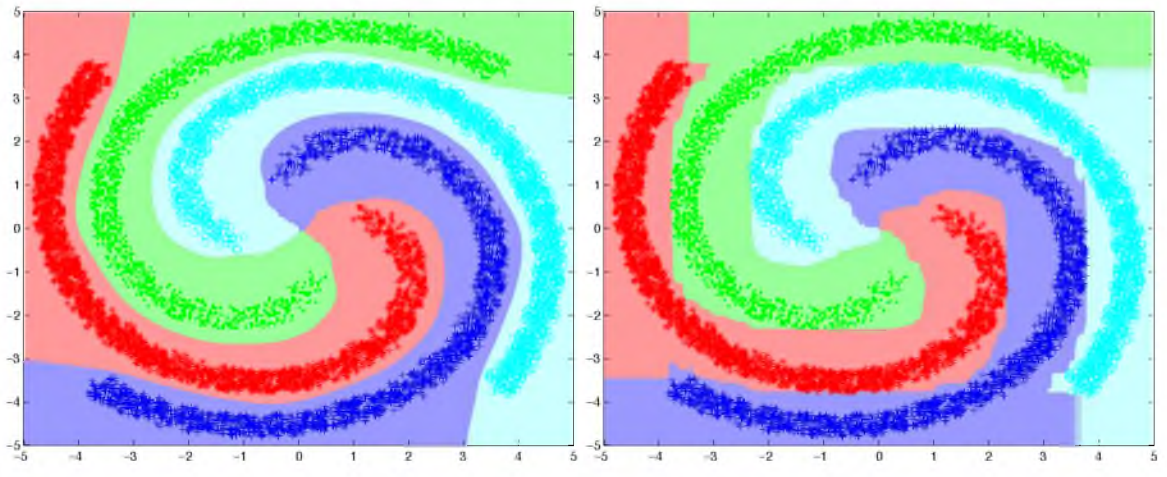


Figure 5.3: Comparison of DNRF (left panel) with conventional random forest (right panel) on the four-spiral dataset. DNRF is robust against overfitting and results in better testing performance.

number of testing samples, and d is the number of features. We used $\frac{2}{3}$ of the samples for training (10% of these samples were used for validation) and $\frac{1}{3}$ of the samples for testing.

We compared the performance of DNDT with decision trees, ANNs, and soft decision trees [141]. The test errors are reported in Table 5.1. DNDT outperforms decision trees with a large margin. It also outperforms both ANNs and soft decision trees. These results assert that the superior performance of DNDT comes from both nonorthogonal splits in a tree structure, as opposed to the decision tree, and unified learning of all the learners, as opposed to the soft decision tree. For soft decision trees, we used the code publicly available by the authors of [141]. The training times of different methods for each dataset are given in Table 5.2. While DNDT is slower than decision trees, it is faster compared to soft decision trees and ANNs. It is worth mentioning that each epoch of update in DNDT is nearly 4 times faster than the back-propagation in ANNs due to the simpler structure of the disjunctive normal form compared to ANN. This simplicity comes from the fact that there is only one set of parameters, $\mathbf{w_k}$, in the differentiable disjunctive normal form (equation 5.5), while in ANNs there are at least two sets of parameters, *i.e.,* weights from the input layer to the hidden layer and weights from the hidden layer to the output layer. The time complexities of decision trees and DNDTs at test time are similar.

We also compared the performance of DNRF with SVMs, boosted trees, oblique random forests (ORF) [132], rotation forests [133], and random forests. For SVMs, we used RBF kernel and the parameters of kernel were found using the search code

Table 5.1: Test errors of different methods for six binary datasets.

	Ionosphere	Wisconsin breast cancer	German credit	PIMA diabetes	Hearts	IJCNN
Decision Tree	12.48%	6.08%	32.73%	31.59%	27.62%	4.79%
ANN	12.10%	2.28%	26.96%	22.11%	20.26%	2.02%
Soft Decision Tree [141]	11.97%	2.12%	25.53%	20.78%	13.33%	2.27%
DNDT	7.15%	1.89%	24.44%	20.56%	13.11%	1.94%

Table 5.2: Training time (seconds) of different methods for six binary datasets.

	Ionosphere	Wisconsin breast cancer	German credit	PIMA diabetes	Hearts	IJCNN
Decision Tree	0.009	0.009	0.010	0.008	0.022	0.500
ANN	0.395	0.854	0.746	0.356	0.956	1860.81
Soft Decision Tree [141]	0.267	0.657	1.858	0.410	0.073	34958.626
DNDT	0.036	0.033	0.228	0.095	0.031	535.36

available in the LIBSVM library [146]. For boosted trees, we used the code publicly available by the authors of [147]. We also used the publicly available R package "obliqueRF" provided by the authors of [132] to run ORF on the binary datasets. Their code supports three different node models: ridge regression, SVM, and logistic regression. For rotation forest, we used the publicly available code provided as part of Weka by the authors of [133]. For all the datasets we used $F = \sqrt{d}$ as the number of features per node in the random forest and used 10% of the training set as the validation set to fine tune the number of trees, N. The same validation set was used to control the number of epochs and tune the step size in the gradient descent algorithm for the DNRF. We ran each classifier 50 times, except for SVMs, which give deterministic results, for each dataset, and the average testing errors for different methods are reported in Table 5.3. The standard deviations are given in parentheses. DNRFs outperform SVMs, boosted trees, random forests, rotation forests, and ORFs. These results again assert that the main advantage of DNRFs come from both the nonorthogonal splits, as opposed to random forests, and the unified learning of all splits, as opposed to ORFs and rotation forests.

Similar to [148], we examined the effect of noise features and tree size on the performance of random forests and DNRFs. In the first experiment, we incrementally added noise/random features to the German credit dataset. Fig. 5.4 shows the test errors for different number of noise features. As the number of noise features increases, the chance that relevant features are selected at each node decreases. While this degrades the performance of random forests, DNRFs remain stable due to the unified

Table 5.3: Test errors of different methods for six binary datasets (average over 50 iterations). The standard deviations are given in parentheses.

	Ionosphere	Wisconsin breast cancer	German credit	PIMA diabetes	Hearts	IJCNN
SVM	4.27% (–)	1.59% (–)	26.12% (–)	22.35% (–)	21.11% (–)	1.31% (–)
Boosted Trees [147]	5.16% (0.9)	2.38% (0.42)	24.29% (0.96)	23.29% (0.99)	15.00% (1.68)	1.39% (0.05)
ORF-ridge [132]	5.69% (0.56)	1.63% (0.21)	25.10% (0.94)	23.25% (0.85)	15.07% (0.90)	1.68% (0.02)
ORF-svm [132]	4.32% (0.50)	1.58% (0.07)	24.23% (0.56)	23.25% (0.81)	13.18% (0.9)	1.87% (0.04)
ORF-log [132]	4.50% (0.62)	1.74% (0.26)	25.23% (0.82)	22.43% (1.26)	15.27% (0.99)	1.62% (0.05)
Rotation Forest [133]	5.04% (1.35)	1.82% (0.76)	25.23% (1.22)	21.63% (1.42)	19.05% (2.56)	2.14% (0.24)
Random Forest [120]	6.99% (0.94)	2.15% (0.41)	24.71% (0.82)	23.92% (0.64)	15.11% (1.14)	2.00% (0.04)
DNRF	3.38% (0.57)	0.53% (≈ 0)	22.91% (0.37)	19.41% (0.42)	12.22% (≈ 0)	1.14% (0.01)

learning strategy. Optimizing all the nodes together decreases the effect of nodes that contain only noise features. In the second experiment, we incrementally decreased the tree size by increasing the minimum node size. We stop splitting the nodes that contain less than samples than the minimum node size. Fig. 5.5 shows the test errors on the German credit dataset for different values of minimum node size. DNRF is less sensitive to the size of tree compared to random forest. This can be seen as the effectiveness of the unified learning strategy which gives more degrees of freedom to DNRF.

5.4.2 Multiclass classification

The four datasets tested for multiclass classification were Pendigit ($M_{tr} = 7494$, $M_{te} = 3498$, $d = 16$, $C = 10$), Optdigit ($M_{tr} = 3823$, $M_{te} = 1797$, $d = 62$, $C = 10$), Landsat ($M_{tr} = 4435$, $M_{te} = 2000$, $d = 36$, $C = 6$), and Letter ($M_{tr} = 16000$, $M_{te} = 4000$, $d = 16$, $C = 26$), where M_{tr} is the number of training samples, M_{te} is

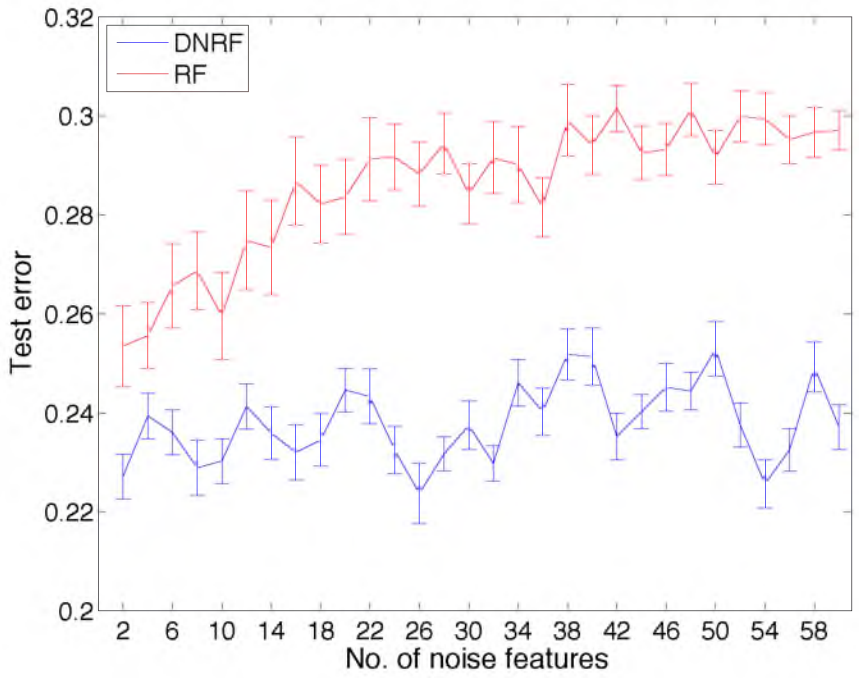


Figure 5.4: The robustness of random forest and DNRF on problems with increasing number of noise/random variables. The dataset, *i.e.,* German credit dataset, has 24 features and an increasing number of noise variables were added. Each experiment was run 50 times, and the error bars illustrate the standard deviations. For all the experiments, random forest used $F = \sqrt{d}$ as the number of features per node.

the number of testing samples, d is the number of features, and C is the number of classes. Similar to binary case, we used $F = \sqrt{d}$ as the number of features per node in the random forest and used 10% of the training set as the validation set to fine tune the number of trees, N, and the ratio of training samples per tree, r.

We ran the experiments 20 times for each dataset and the average testing errors are reported in Table 5.4. DNRF outperforms random forests and state-of-the-art algorithms [142] and [143] in all cases.

5.5 Conclusion

We introduced a new learning scheme for random forests, called DNRF, based on the disjunctive normal form of decision trees. Unlike conventional random forests with orthogonal axis-aligned splits, DNRFs can learn arbitrary non-axis-aligned splits. Moreover, DNRFs allow different nodes of a decision tree interact with each other during training in a unified optimization framework. We showed that DNRFs outper-

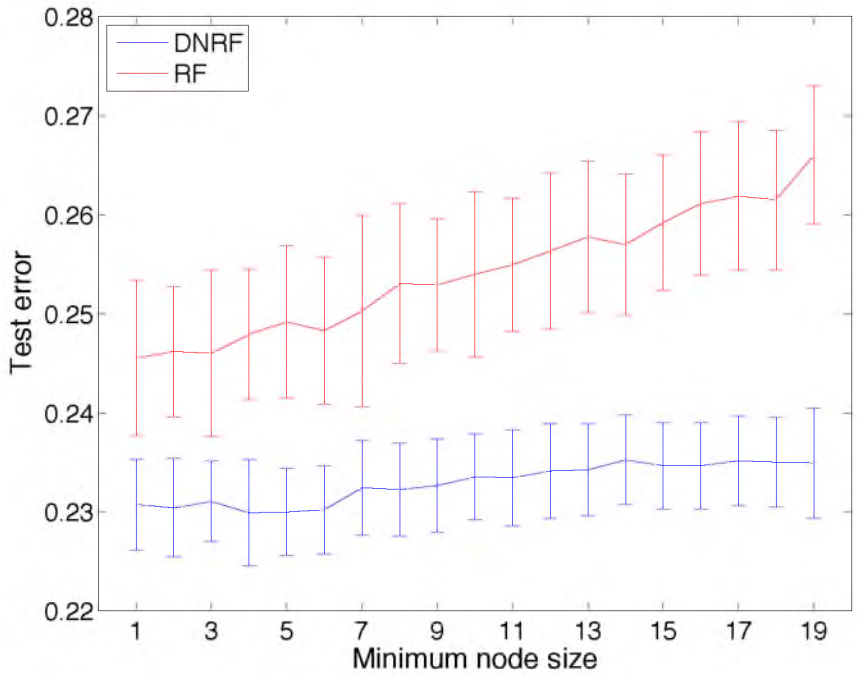


Figure 5.5: The effect of tree size, *i.e.,* tree depth, on the performance of random forest and DNRF. The depth of tree was controlled by the minimum node size. Smaller node size results in deeper trees. Each model was run 50 times, and the error bars represent standard deviations. The German credit dataset was used in this experiment.

Table 5.4: Test errors of different methods on four UCI datasets (multiclass classification).

	Pendigit	Optdigit	Landsat	Letter
GD-MCBoost [143]	7.06%	7.68%	13.35%	40.35%
Space Partitioning [142]	4.32%	4.23%	13.95%	13.08%
Random Forest [120]	3.87%	3.22%	10.49%	4.70%
DNRF	2.17%	2.30%	9.63%	2.05%

form conventional random forests on binary and multiclass benchmark datasets. Our results are also better than oblique random forests [132] which learns nonorthogonal learners at each node.

It must be emphasized that optimizing all the individual trees together in DNRF would increase the correlation between the trees and increasing the correlation decreases the forest performance [120]. Hence, treating each tree individually, as men-

tioned in Algorithm 5, is crucial to the performance of DNRF. The initialization is also important to the performance of DNRF. If the DNRF is initialized with a random tree, it performs poorly. The reason is that the cost function is not convex and gradient descent might get stuck in a local minima.

DNRFs can handle categorical data similar to conventional random forests. After a conventional RF is trained, the same optimization approach can be applied to construct a DNRF. However, DNRFs do not handle missing values in the current format. One possible solution is to assign zero weight to the missing features in the paths containing samples with missing values, but this is a topic of future research.

Finally, even though we described DNRF for the cases that weak learners are linear, the DNRF formulation can be extended to any differentiable nonlinear weak learners theoretically. The performance, advantages, and disadvantages of nonlinear DNRFs can be a topic of future research.

CHAPTER 6

SEGMENTATION OF MITOCHONDRIA IN ELECTRON MICROSCOPY IMAGES USING ALGEBRAIC CURVES

High-resolution microscopy techniques have been used to generate large volumes of data with enough details for understanding the complex structure of the nervous system. However, automatic techniques are required to segment cells and intracellular structures in these multiterabyte datasets and make anatomical analysis possible on a large scale. We propose a fully automated method that exploits both shape information and regional statistics to segment irregularly shaped intracellular structures such as mitochondria in electron microscopy (EM) images. The main idea is to use algebraic curves to extract shape features together with texture features from inside and outside of mitochondria. Then, these powerful features are used to learn a random forest classifier, which can predict mitochondria locations precisely. Finally, the algebraic curves together with regional information are used to segment the mitochondria at the predicted locations. We demonstrate that our method outperforms the state-of-the-art algorithms in segmentation of mitochondria in EM images.

6.1 Introduction

The morphology and distribution of intracellular components is of substantial biological importance for neuroscientists. For example, abnormal mitochondria morphology can be seen in Parkinson's disease-related genes [20], or geometrical properties of mitochondria can be used to distinguish cancer cells from normal cells [21]. In addition, an accurate mitochondria segmentation would improve cell segmentation results by distinguishing mitochondria membranes from other cell membranes [22].

Electron microscopy (EM) imaging techniques generate nanoscale images that contain enough details for study of intracellular components, such as mitochondria. However, the sheer size of a typical EM dataset, often approaching ten of terabytes [13], makes manual analysis infeasible [14]. Hence, automated image analysis is required. However, fully automatic analysis of EM datasets is challenging because numerous intracellular components exhibit irregular shapes and have similar local appearances [19]. Moreover, the texture and physical topologies of intracellular components are highly variable [18](Fig. 6.1). Therefore, a robust automated method must overcome these issues to segment complex intracellular objects such as mitochondria.

General segmentation methods which have been proposed for natural image datasets yield poor results when applied to EM images [19]. Jain *et al.* [41] showed that global probability boundary [44] and boosted edge learning [43], which result in outstanding segmentation performance on natural images, perform poorly on EM datasets. Therefore, a successful method for segmenting specific structures such as mitochondria must be optimized for EM images.

There are several segmentation methods that handle EM images specifically. Vu and Manjunath [45] proposed a graph-cut method that minimizes an energy function over the pixel intensity and flux of the gradient field for cell segmentation. However,

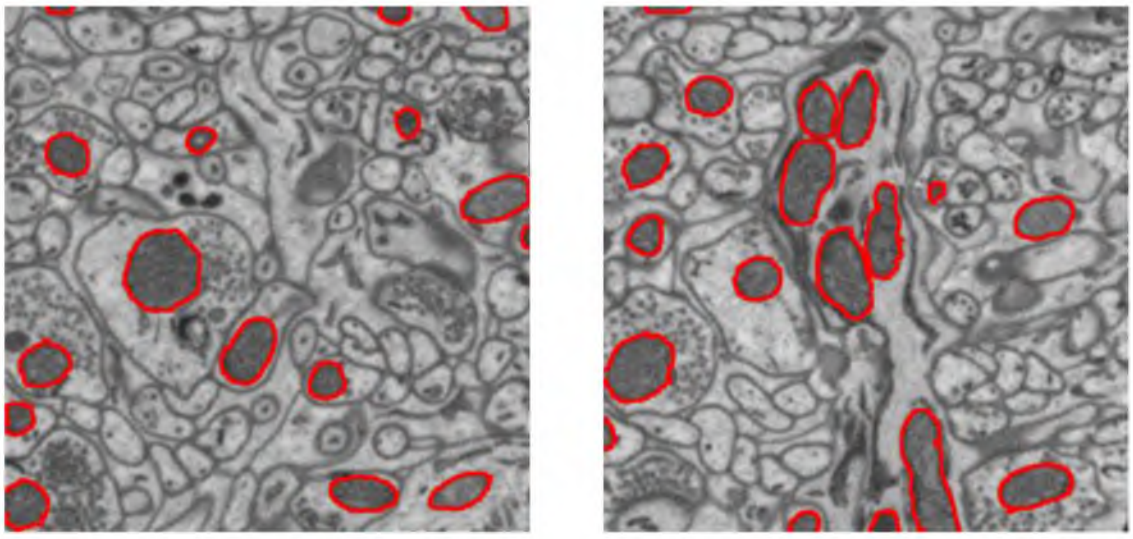


Figure 6.1: Mitochondria (objects with red boundary) appear in different shapes and different local intensities in EM images. The variety in shape and the existence of other similar intracellular components make segmentation a challenging task.

their model might be confused by the complex intracellular structures and requires user interaction to correct segmentation errors. The contour propagation model [46] that minimizes an energy function for contour tracing of cell membranes can also get stuck in local minima due to complex intracellular structures. In [149], textural information is used to train a Gentle-boost classifier for mitochondria segmentation of the lateral part of the rat's brain. In other previous work [21], texton-based region features are used with different classification methods to segment mitochondria in MNT-1 cells. Even though these methods make extensive use of textural information, they ignore the shape information. In [150], ray features were proposed to capture shape information for detection of irregular shapes, such as mitochondria, but they only rely on geometric information of shapes and ignore texture information. Radon like features [47] are another set of features designed to take both texture and geometric information into account and can be tuned to segment different objects in EM images.

More powerful mitochondria segmentation methods working on 3D volumes have been proposed recently. Lucchi *et al.* [151] solved a graph partitioning problem by learning a classifier based on the textural and shape information to segment mitochondria. Giuly *et al.* [22] proposed a multistep approach that exploits a patch classifier followed by a contour pair classification and level sets. We also propose a multistep approach that combines textural and shape information to provide a high-accuracy mitochondria segmentation. As a first step, we extract patches with different sizes from the input image and fit algebraic curves, *i.e.,* polynomials of different degrees, to each patch. Next, shape and texture features are extracted based on the fitted polynomials from each patch. The extracted features are then used to train a classifier that predicts if a patch belongs to a mitochondrion or not. Finally, in the patches containing mitochondria, based on the classifier decision, we use the connected component of the center pixel bounded by the fitted polynomial to segment the mitochondria.

Algebraic curves, *i.e.,* the zero set of polynomials in two variables, are suitable for modelling complicated shapes [152]. Moreover, they take advantage of all data in an image patch and thus are able to find weak edges embedded in noise [153].

The power of algebraic curves in finding ambiguous edges in cluttered backgrounds can be used to estimate the boundary of mitochondria and extract informative shape and textural features from images. The regional features, *i.e.,* textural features from image regions, are more robust and informative compared to pixel features. We use the extracted features in a supervised model to increase the accuracy of segmentation.

6.2 Mitochondria Segmentation

Our proposed method is composed of four steps: Curve fitting, feature extraction, classifier training, and automatic pixel labelling.

6.2.1 Curve fitting

A dth degree polynomial can be represented by $f(x,y) = \sum_{0 \leq l+m \leq d} a_{lm} x^l y^m$. Given an $n \times n$ image patch $P(x,y)$, we fit a polynomial to the patch by minimizing the cost function E:

$$E = \sum_{i,j=1}^{n} (w_{ij}^2 (f_d^2(x_i, y_j) + (\frac{\nabla P(x_i, y_j)}{w_{ij}} . \nabla f(x_i, y_j) - 1)^2)) \tag{6.1}$$

where $\nabla P(x_i, y_j)$ denotes the gradient vector at pixel (x_i, y_j), $\nabla f(x_i, y_j)$ is the gradient vector of polynomial f at (x_i, y_j), and w_{ij} is the length of $\nabla P(x_i, y_j)$. This minimization problem can be solved using linear least squares [153].

In (6.1), the $f_d^2(x_i, y_j)$ term determines the zero level set of the fitted polynomial, *i.e.,* $f(x,y) = 0$, and the $(\frac{\nabla P(x_i, y_j)}{w_{ij}} . \nabla f(x_i, y_j) - 1)^2$ term forces the $\nabla f(x,y)$ to have the same direction as $\nabla P(x,y)$ with unit magnitude at each point. The effect of large gradients in noisy areas is damped by this unit magnitude constraint. Finally, the w_{ij}^2 term increases the influence of pixels with large gradient magnitude. These pixels are most likely on the target contour and have larger gradient magnitudes compared to noisy pixels. The above-mentioned properties make this fitting strategy appropriate for noisy EM images with complex regional textures. In addition, this fitting strategy is rotation and scale invariant and thus is suitable for shape representation. Fig. 6.2 shows the fitted polynomials to two patches with mitochondria and two patches without mitochondria in them.

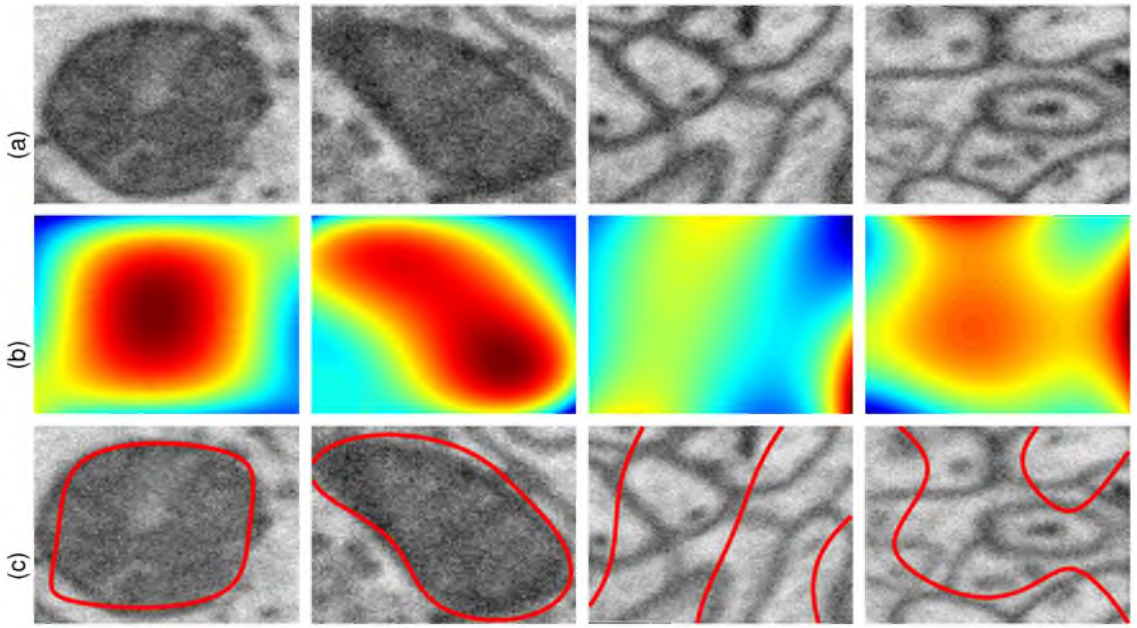


Figure 6.2: Curve fitting samples. (a) Two patches with mitochondria (two left columns) and two patches without mitochondria (two right columns) in them, (b) fitted polynomials of degree 4 to the patches in (a), and (c) the zero level sets overlayed on the input patches.

6.2.2 Feature extraction

We use zero level sets of the fitted polynomials (Fig. 6.2(c)) to extract both shape and textural features from each patch. The zero level set divides each patch to two disjoint regions: inside, *i.e.*, $f(x, y) > 0$ and outside, *i.e.*, $f(x, y) < 0$. Each polynomial thus forms a hypothesis of existence of a mitochondrion in the patch. The inside region and the zero level set curve exhibit similar features among the patches with mitochondria (two left columns in Fig. 6.2) which are different from features of the patches without mitochondria (two right columns in Fig. 6.2). The textural features are extracted from inside of the zero level set curve and include Hu's invariant moments, mean, variance, skewness, kurtosis, and entropy of the pixel intensities. The shape features are extracted from the zero level set curve itself. They contain Hu's invariant moments of the curve and the average intensity of pixels on the curve. We also use the ratio of the inner area to the curve length as another shape feature. The combination of the textural and shape features provide a rich set of features that can be used to detect mitochondria in an image.

6.2.3 Classifier training

The extracted features from each patch are used to train a binary random forest classifier that predicts whether that patch belongs to a mitochondrion or not. In practice, we extract many patches with different sizes at different locations and fit polynomials of different degrees to each of them. It is worth noting that we use different patch sizes due to different sizes of mitochondria and fit polynomials of different degree due to different shape complexities of mitochondria. The shape and textural features are then extracted from each patch. To train the classifier, the patches that their centers are close to centers of mitochondria are considered as positive samples and the remaining patches are considered as negative samples. The centers of mitochondria are the center of mass of connected components in groundtruth images. The classifier indeed tests the hypothesis that made by the polynomials. It must me emphasized that many of them will be false because there are few mitochondria in each image.

6.2.4 Automatic pixel labelling

For a given input image, overlapping patches with different sizes are extracted. Next, polynomials of different degrees are fitted to each patch and the shape and textural features are computed for each patch. These features are then passed to the random forest classifier. If a patch is classified as positive by the random forest classifier, all the connected pixels of the center pixel in that patch are marked as mitochondria in the input image. The connected pixels of the center pixel are found in a certain threshold around the intensity of the center pixel. To add more certainty to the labelling process, we only mark the connected pixels inside the zero level set curve as mitochondria and consider the remaining pixels as background.

The segmentation accuracy can be improved by applying morphological postprocessing. We apply the morphological dilation followed by the region filling to fill holes in the segmented mitochondria. This results in a clean segmented image.

6.3 Experimental Results

We test the performance of our proposed method on two different sets of EM images: mouse neuropil and Drosophila first instar larva ventral nerve cord (VNC) [52].

The mouse neuropil dataset contains 40 images of size 700×700. It has a pixel resolution of 10×10 nm/pixel. 14 of these images were used for training and the remaining images were used for testing. The Drosophila VNC datset contains 30 images of size 512×512 with 4×4 nm/pixel resolution. 15 of these images were used for training and the remaining images were used for testing. The groundtruth images of mitochondria were annotated by a neuroanatomist.

For both of the datasets, we extracted patches with four different sizes, 48×48, 64×64, 88×88, 104×104, and fit polynomials of two different degrees, 2 and 4. The discussed features in section 6.2.2 were then extracted and a random forest classifier with 100 trees were trained.

We compared the accuracy of our proposed method with a patch-based pixel classifier, radon-like features method [47], and the MCMS model. An artificial neural networks classifier with 10 hidden nodes was used as the pixel classifier. For the pixel classifier,RLF method, and the MCMS model, the best threshold was found using the training results. Table 6.1 shows the segmentation accuracy of different methods for the testing set in the mouse neuropil dataset. It can be observed that our proposed method has better performance than the pixel classifier and RLF method, a 14.9% and 2.4% improvement in the testing F-value compared to the pixel classifier and the RLF method, respectively. It is worth noting that even though the MCMS model performs better than the proposed method, its computational complexity id higher. Moreover, it needs membrane labels as well as the mitochondria label for training.

For the Drosophila VNC dataset, we compared our proposed method with Giuly *et al.* [22] method in addition to the pixel classifier, RLF method, and the MCMS model. This dataset is more difficult and the quality of images is lower than the mouse

Table 6.1: Testing performance of different methods for the mouse neuropil dataset.

Method	Precision	Recall	Fvalue
Pixel classifier	67.18%	68.05%	67.61%
RLF [47]	78.07%	82.31%	80.14%
MCMS	83.17%	85.04%	84.09%
Proposed method	82.51%	82.47%	82.49%

neuropil dataset. While the performance of RLF method was close to the performance of our proposed method for the mouse neuropil dataset, our method outperformed the RLF method with more than 20% in the testing F-value for this dataset. One of the advantages of our method is that it is robust against the texture and noise in the EM images and thus performs reasonably even for low quality datasets like Drosophila VNC dataset. Note that the proposed method outperforms MCMS model on this dataset. The segmentation accuracy results are shown in Table 6.2. Fig. 6.3 shows the segmentation results of mitochondria for two test images from the mouse neuropil and the Drosophila VNC dataset.

Table 6.2: Testing performance of different methods for the Drosophila VNC dataset.

Method	Precision	Recall	Fvalue
Pixel classifier	31.29%	60.44%	41.24%
RLF [47]	46.12%	57.67%	51.25%
Giuly et.al. [22]	64.22%	57.01%	60.40%
MCMS	57.51%	82.42%	67.32%
Proposed method	78.57%	68.08%	72.95%

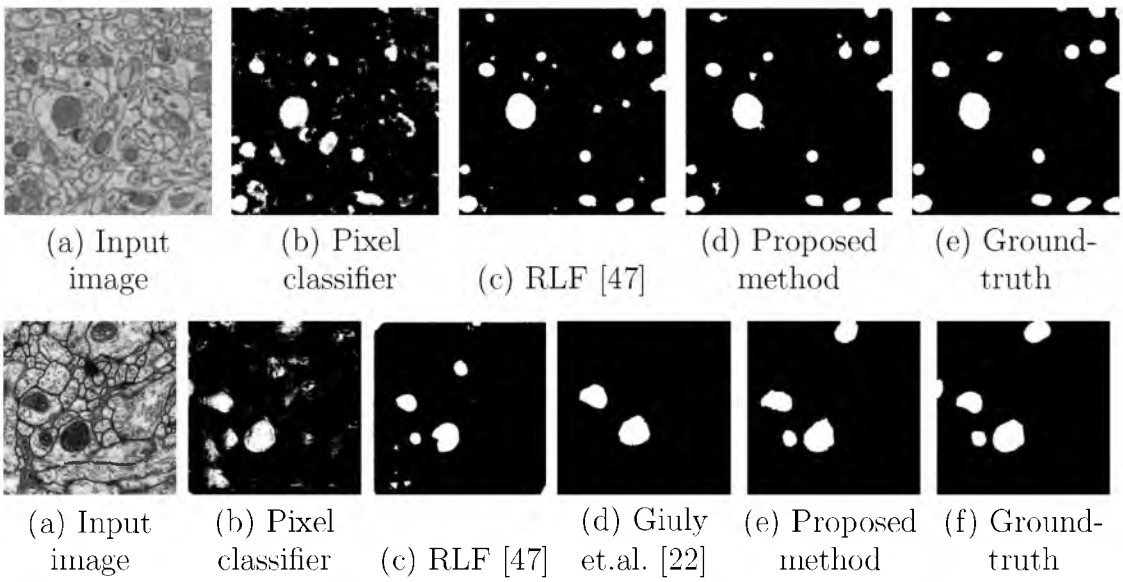


Figure 6.3: Test results for the mitochondria segmentation. First row: mouse neuropil daaset, second row: Drosophila VNC dataset.

6.4 Conclusion

We introduced a mitochondria segmentation framework using algebraic curves. The main idea of our method is to use the power of algebraic curves to extract both shape and textural features from input images. The algebraic curves use all the information in a window and are robust against noise and texture. Moreover, algebraic curves enable our method to use regional features that are more informative compared to pixel-wise features. We use the extracted feature to train a random forest that detects mitochondria in input images. Finally, we apply an automatic pixel labelling approach by finding connected components of the center pixels in the patches that the classifier classifies them as positive samples.

APPENDIX

DERIVATION OF GRADIENT FOR DNRF

In this section we show the derivation of equation (5.8). The gradient for the training pair $(\mathbf{x}, y)$ can be computed using the chain rule for differentiation:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial \tilde{h}} \sum_{l} \frac{\partial \tilde{h}}{\partial g_l} \frac{\partial g_l}{\partial \tilde{f}_j} \frac{\partial \tilde{f}_j}{\partial w_{kj}}. \tag{A.1}$$

Each of the derivatives on the right hand side can be computed as follows:

$$\frac{\partial E}{\partial \tilde{h}} = -2(y - \tilde{h}(\mathbf{x})),$$

$$\frac{\partial \tilde{h}}{\partial g_l} = \prod_{r \neq l} (1 - g_r(\mathbf{x})),$$

$$\frac{\partial g_l}{\partial \tilde{f}_j} = \begin{cases} \prod_{\substack{r \neq j \\ r \in R_l}} \tilde{f}_r(\mathbf{x}) \prod_{r \in L_l} (1 - \tilde{f}_r(\mathbf{x})) & \text{if } \ j \in R_l \\ -\prod_{r \in R_l} \tilde{f}_r(\mathbf{x}) \prod_{\substack{r \neq j \\ r \in L_l}} (1 - \tilde{f}_r(\mathbf{x})) & \text{if } \ j \in L_l \\ 0 & \textit{otherwise} \end{cases},$$

$$\frac{\partial \tilde{f}_j}{\partial w_{kj}} = x_j \tilde{f}_j(\mathbf{x})(1 - \tilde{f}_j(\mathbf{x})).$$

By multiplying these derivatives the gradient in (5.8) is obtained.

REFERENCES

[1] S. Z. Li, *Markov Random Field Modeling in Computer Vision.* New York:Springer-Verlag, 1995.

[2] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1925–1929, 2013.

[3] Z. Tu and X. Bai, "Auto-context and its application to high-level vision tasks and 3d brain image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 10, pp. 1744–1757, 2010.

[4] A. Torralba, K. P. Murphy, and W. T. Freeman, "Contextual models for object detection using boosted random fields," *Proc. Neural Inform. Process. Syst.*, 2004.

[5] G. Heitz, S. Gould, A. Saxena, and D. Koller, "Cascaded classification models: Combining models for holistic scene understanding," *Proc. Neural Inform. Process. Syst.*, pp. 641–648, 2008.

[6] C. Li, A. Kowdle, A. Saxena, and T. Chen, "Toward holistic scene understanding: Feedback enabled cascaded classification models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 7, pp. 1394–1408, 2012.

[7] E. Jurrus, A. R. C. Paiva, S. Watanabe, J. R. Anderson, B. W. Jones, R. T. Whitaker, E. M. Jorgensen, R. E. Marc, and T. Tasdizen, "Detection of neuron membranes in electron microscopy images using a serial neural network architecture," *Medical Image Anal.*, vol. 14, no. 6, pp. 770–783, 2010.

[8] S. Gould, R. Fulton, and D. Koller, "Decomposing a scene into geometric and semantically consistent regions," in *Proc. Int. Conf. Comput. Vision.* IEEE, 2009, pp. 1–8.

[9] E. Borenstein, E. Sharon, and S. Ullman, "Combining top-down and bottom-up segmentation," *Proc. Comput. Vision and Pattern Recognition Workshop*, 2004.

[10] D. Martin, C. Fowlkes, D. Tal, and J. Malik, "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics," in *Proc. Int. Conf. Comput. Vision*, vol. 2, July 2001, pp. 416–423.

[11] O. Sporns, G. Tononi, and R. Ktter, "The human connectome: a structural description of the human brain," *PLoS Comput. Biol.*, vol. 1, p. e42, 2005.

[12] W. Denk and H. Horstmann, "Serial block-face scanning electron microscopy to reconstruct three-dimensional tissue nanostructure," *PLoS Biol.*, vol. 2, p. e329, 2004.

[13] J. Anderson, B. Jones, C. Watt, M. Shaw, J.-H. Yang, D. DeMill, J. Lauritzen, Y. Lin, K. Rapp, D. Mastronarde, P. Koshevoy, B. Grimm, T. Tasdizen, R. Whitaker, and R. Marc, "Exploring the retinal connectome," *Mol. Vision*, no. 17, pp. 355–379, 2011.

[14] K. L. Briggman and W. Denk, "Towards neural circuit reconstruction with volume electron microscopy techniques," *Curr. Opin. Neurobio.*, vol. 16, no. 5, pp. 562–570, 2006.

[15] D. H. Hall and R. Russell, "The posterior nervous system of the nematode caenorhabditis elegans: serial reconstruction of identified neurons and complete pattern of synaptic interactions," *J. Neurosci.*, vol. 11, no. 1, pp. 1–22, 1991.

[16] J. G. White, E. Southgate, J. N. Thomson, and S. Brenner, "The structure of the nervous system of the nematode caenorhabditis elegans," *Philos. Trans. R. Soc., B*, vol. 314, no. 1165, pp. 1–340, 1986.

[17] J. R. Anderson, B. W. Jones, J.-H. Yang, M. V. Shaw, C. B. Watt, P. Koshevoy, J. Spaltenstein, E. Jurrus, K. UV, R. T. Whitaker, D. Mastronarde, T. Tasdizen, and R. E. Marc, "A computational framework for ultrastructural mapping of neural circuitry," *PLoS Biol.*, vol. 7, no. 3, p. e1000074, 03 2009.

[18] 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," *Proc. Int. Conf. Comput. Vision*, pp. 1–8, 2007.

[19] A. Lucchi, K. Smith, R. Achanta, V. Lepetit, and P. Fua, "A fully automated approach to segmentation of irregularly shaped cellular structures in em images," in *Proc. Medical Image Computing and Comput. Assisted Intervention*, 2010, pp. 463–471.

[20] D. Cho, T. Nakamura, and S. Lipton, "Mitochondrial dynamics in cell death and neurodegeneration," *Cell. Mol. Life Sci.*, vol. 67, no. 20, pp. 3435–3447, 2010.

[21] R. Narasimha, H. Ouyang, A. Gray, S. W. McLaughlin, and S. Subramaniam, "Automatic joint classification and segmentation of whole cell 3d images," *Pattern Recogn.*, vol. 42, no. 6, pp. 1067–1079, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2008.08.009

[22] R. Giuly, M. Martone, and M. Ellisman, "Method: automatic segmentation of mitochondria utilizing patch classification, contour pair classification, and automatically seeded level sets," *BMC Bioinformatics*, vol. 13, no. 1, p. 29, 2012. [Online]. Available: http://www.biomedcentral.com/1471-2105/13/29

[23] M. Seyedhosseini, R. Kumar, E. Jurrus, R. Guily, M. Ellisman, H. Pfister, and T. Tasdizen, "Detection of neuron membranes in electron microscopy images using multi-scale context and radon-like features," *Proc. Medical Image Computing and Comput. Assisted Intervention*, 2011.

[24] M. Seyedhosseini and T. Tasdizen, "Multi-class multi-scale series contextual model for image segmentation," *IEEE Trans. Image Process.*, vol. 22, no. 11, pp. 4486–4496, 2013.

[25] M. Seyedhosseini, M. Sajjadi, and T. Tasdizen, "Image segmentation with cascaded hierarchical models and logistic disjunctive normal networks," in *Proc. Int. Conf. Comput. Vision*, 2013.

[26] M. Seyedhosseini, A. Paiva, and T. Tasdizen, "Fast AdaBoost training using weighted novelty selection," in *Proc. IEEE Intl. Joint Conf. Neural Networks*, San Jose, CA, USA, August 2011, pp. 1245–1250.

[27] M. Seyedhosseini, M. Ellisman, and T. Tasdizen, "Segmentation of mitochondria in electron microscopy images using algebraic curves," in *Proc. Int. Symp. Biomedical Imaging*, 2013, pp. 860–863.

[28] P. Vandewalle, J. Kovacevic, and M. Vetterli, "Reproducible research," http://reproducibleresearch.net/index.php/Main_Page.

[29] G. Bradski, *Dr. Dobb's J. Software Tools*, 2000, software available at http://www.opencv.org.

[30] "Boost c++ library," 2000, software available at http://www.boost.org.

[31] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 4, pp. 509–522, 2002.

[32] M. Fink and P. Perona, "Mutual boosting for contextual inference," *Proc. Neural Inform. Process. Syst.*, 2004.

[33] A. Singhal, J. Luo, and W. Zhu, "Probabilistic spatial context models for scene content understanding," *Proc. Comput. Vision and Pattern Recognition*, vol. 1, pp. 235–241, 2003.

[34] K. Murphy, A. Torralba, and W. T. Freeman, "Using the forest to see the trees: A graphical model relating features, objects, and scenes," *Proc. Neural Inform. Process. Syst.*, 2003.

[35] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions, and the bayesian restoration of images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 6, no. 6, pp. 721–741, 1984.

[36] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," *Proc. Int. Conf. Mach. Learning*, pp. 282–289, 2001.

[37] X. He, R. Zemel, and M. Carreira-Perpinan, "Multiscale conditional random fields for image labeling," *Proc. Comput. Vision and Pattern Recognition*, vol. 2, pp. 695–702, 2004.

[38] Y. LeCun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," *Proc. Comput. Vision and Pattern Recognition*, vol. 2, pp. 97 –104, 2004.

[39] C. Desai, D. Ramanan, and C. Fowlkes, "Discriminative models for multi-class object layout," *Proc. Int. Conf. Comput. Vision*, 2009.

[40] M. J. Choi, A. Torralba, and A. S. Willsky, "A tree-based context model for object recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 2, pp. 240–252, 2012.

[41] V. Jain, B. Bollmann, M. Richardson, D. Berger, M. Helmstaedter, K. Briggman, W. Denk, J. Bowden, J. Mendenhall, W. Abraham, K. Harris, N. Kasthuri, K. Hayworth, R. Schalek, J. Tapia, J. Lichtman, and H. Seung, "Boundary learning by optimization with topological constraints," *Proc. Comput. Vision and Pattern Recognition*, pp. 2488 –2495, 2010.

[42] T. Cour, F. Benezit, and J. Shi, "Spectral segmentation with multiscale graph decomposition," *Proc. Comput. Vision and Pattern Recognition*, vol. 2, pp. 1124–1131, 2005.

[43] P. Dollar, Z. Tu, and S. Belongie, "Supervised learning of edges and object boundaries," *Proc. Comput. Vision and Pattern Recognition*, 2006.

[44] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "From contours to regions: an empirical evaluation," *Proc. Comput. Vision and Pattern Recognition*, vol. 0, pp. 2294–2301, 2009.

[45] N. Vu and B. S. Manjunath, "Graph cut segmentation of neuronal structures from transmission electron micrographs," in *Proc. ICIP*, 2008, pp. 725–728.

[46] J. H. Macke, N. Maack, R. Gupta, W. Denk, B. Schlkopf, and A. Borst, "Contour-propagation algorithms for semi-automated reconstruction of neural processes," *J. Neurosci. Methods*, vol. 167, no. 2, pp. 349–357, 2008.

[47] R. Kumar, A. Va andzquez Reina, and H. Pfister, "Radon-like features and their application to connectomics," in *Proc. Comput. Vision and Pattern Recognition Workshop*, June 2010, pp. 186–193.

[48] C. Becker, K. Ali, G. Knott, and P. Fua, "Learning context cues for synapse segmentation in em volumes," *Proc. Medical Image Computing and Comput. Assisted Intervention*, 2012.

[49] A. Kreshuk, C. N. Straehle, C. Sommer, U. Köthe, G. Knott, and F. A. Hamprecht, "Automated segmentation of synapses in 3d em data," in *Proc. Int. Symp. Biomedical Imaging*, 2011, pp. 220–223.

[50] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," *Proc. ISCAS*, pp. 253–256, 2010.

[51] A. Cardona, S. Saalfeld, S. Preibisch, B. Schmid, A. Cheng, J. Pulokas, P. Tomančák, and V. Hartenstein, "An integrated micro- and macroarchitectural analysis of the *Drosophila* brain by computer-assisted serial section electron microscopy," *PLoS Biol.*, vol. 8, no. 10, p. e1000502, 10 2010.

[52] A. Cardona, S. Saalfeld, J. Schindelin, I. Arganda-Carreras, S. Preibisch, M. Longair, P. Tomancak, V. Hartenstein, and R. J. Douglas, "Trakem2 software for neural circuit reconstruction," *PLoS One*, vol. 7, no. 6, p. e38011, 06 2012.

[53] D. B. Chklovskii, S. Vitaladevuni, and L. K. Scheffer, "Semi-automated reconstruction of neural circuits using electron microscopy," *Curr. Opin. Neurobio.*, vol. 20, no. 5, pp. 667 – 675, 2010.

[54] S. Haykin, *Neural networks - a comprehensive foundation*, 2nd ed. Upper Saddle River: Prentice-Hall, 1999.

[55] J. C. Principe, N. R. Euliano, and W. C. Lefebvre, *Neural and adaptive systems: fundamentals through simulations.* New York: Wiley, 2000.

[56] B. Andres, U. Kothe, M. Helmstaedter, W. Denk, and F. A. Hamprecht, "Segmentation of sbfsem volume data of neural tissue by hierarchical classification," in *Proc. DAGM symp. Pattern Recognition*, 2008, pp. 142–152.

[57] J. Funke, B. Andres, F. A. Hamprecht, A. Cardona, and M. Cook, "Efficient automatic 3D-reconstruction of branching neurons from EM data," in *Proc. Comput. Vision and Pattern Recognition*, 2012.

[58] T. Liu, E. Jurrus, M. Seyedhosseini, M. Ellisman, and T. Tasdizen, "Watershed merge tree classification for electron microscopy image segmentation," *Proc. Int. Conf. Pattern Recognition*, 2012.

[59] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *J. Am. Stat. Assoc.*, vol. 66, no. 336, pp. 846–850, 1971.

[60] D. Larlus and F. Jurie, "Combining appearance models and markov random fields for category level object segmentation," in *Proc. Comput. Vision and Pattern Recognition.* IEEE, 2008, pp. 1–7.

[61] M. P. Kumar and D. Koller, "Efficiently selecting regions for scene understanding," in *Proc. Comput. Vision and Pattern Recognition.* IEEE, 2010, pp. 3217–3224.

[62] J. Tighe and S. Lazebnik, "Superparsing: scalable nonparametric image parsing with superpixels," in *Proc. European Conf. Comput. Vision.* Springer, 2010, pp. 352–365.

[63] L. Ladicky, C. Russell, P. Kohli, and P. H. Torr, "Associative hierarchical crfs for object class image segmentation," in *Proc. Int. Conf. Comput. Vision.* IEEE, 2009, pp. 739–746.

[64] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 9, pp. 1124–1137, 2004.

[65] X. Ren, L. Bo, and D. Fox, "Rgb-(d) scene labeling: Features and algorithms," in *Proc. Comput. Vision and Pattern Recognition*, 2012, pp. 2759–2766.

[66] P. Kohli, P. H. Torr *et al.*, "Robust higher order potentials for enforcing label consistency," *Int. J. Comput. Vision*, vol. 82, no. 3, pp. 302–324, 2009.

[67] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "Slic superpixels compared to state-of-the-art superpixel methods," *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 2274–2282, 2012.

[68] D. Munoz, J. A. Bagnell, and M. Hebert, "Stacked hierarchical labeling," in *Proc. European Conf. Comput. Vision.* Springer, 2010, pp. 57–70.

[69] D. Grangier, L. Bottou, and R. Collobert, "Deep convolutional networks for scene parsing," in *Proc. Int. Conf. Mach. learning*, 2009.

[70] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[71] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Neural Inform. Process. Syst.*, 2012, pp. 1106–1114.

[72] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc. Comput. Vision and Pattern Recognition*, 2012, pp. 3642–3649.

[73] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," in *Proc. Neural Inform. Process. Syst.*, 2013, pp. 2553–2561.

[74] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Proc. Int. Conf. Comput. Vision*, 2009, pp. 2146–2153.

[75] S. C. Turaga, K. L. Briggman, M. Helmstaedter, W. Denk, and H. S. Seung, "Maximin affinity learning of image segmentation," *Proc. Neural Inform. Process. Syst.*, 2009.

[76] D. Ciresan, A. Giusti, J. Schmidhuber *et al.*, "Deep neural networks segment neuronal membranes in electron microscopy images," in *Proc. Neural Inform. Process. Syst.*, 2012, pp. 2852–2860.

[77] I. Arganda-Carreras, S. Seung, A. Cardona, and J. Schindelin, "ISBI2012 segmentation of neuronal structures in em stacks," http://brainiac2.mit.edu/isbi_challenge/, 2012.

[78] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 5, pp. 898–916, 2011.

[79] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, no. 6, pp. 679–698, 1986.

[80] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 7, pp. 629–639, jul. 1990.

[81] J. J. Lim, C. L. Zitnick, and P. Dollár, "Sketch tokens: a learned mid-level representation for contour and object detection," in *Proc. Comput. Vision and Pattern Recognition*, 2013.

[82] R. Xiaofeng and L. Bo, "Discriminatively trained sparse code gradients for contour detection," in *Proc. Neural Inform. Process. Syst.*, 2012, pp. 593–601.

[83] P. Dollár and C. L. Zitnick, "Structured forests for fast edge detection," in *Proc. Int. Conf. Comput. Vision*, 2013.

[84] D. R. Martin, C. C. Fowlkes, and J. Malik, "Learning to detect natural image boundaries using local brightness, color, and texture cues," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 5, pp. 530–549, 2004.

[85] J. Mairal, M. Leordeanu, F. Bach, M. Hebert, and J. Ponce, "Discriminative sparse image models for class-specific edge detection and image interpretation," in *Proc. European Conf. Comput. Vision*. Springer, 2008, pp. 43–56.

[86] Z. Tu, "Probabilistic boosting-tree: learning discriminative models for classification, recognition, and clustering," *Proc. Int. Conf. Comput. Vision*, vol. 2, pp. 1589–1596, 2005.

[87] M. Hazewinkel, *Encyclopaedia of Mathematics, Supplement III*. Springer, 2001, vol. 13.

[88] P. K. Simpson, "Fuzzy min-max neural networks. i. classification," *IEEE Trans. Neural Netw.*, vol. 3, no. 5, pp. 776–786, 1992.

[89] P. Viola and M. J. Jones, "Robust real-time face detection," *Int. J. comput. vision*, vol. 57, no. 2, pp. 137–154, 2004.

[90] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *Proc. Comput. Vision and Pattern Recognition*, vol. 1, pp. 886–893, 2005.

[91] C. Liu, J. Yuen, and A. Torralba, "Sift flow: Dense correspondence across scenes and its applications," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 5, pp. 978–994, 2011.

[92] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[93] L. Bertelli, T. Yu, D. Vu, and B. Gokturk, "Kernelized structural svm learning for supervised object segmentation," in *Proc. Comput. Vision and Pattern Recognition*, 2011.

[94] D. Kuettel and V. Ferrari, "Figure-ground segmentation by transferring window masks," in *Proc. Comput. Vision and Pattern Recognition*, 2012.

[95] A. Levin and Y. Weiss, "Learning to combine bottom-up and top-down segmentation," in *Proc. European Conf. Comput. Vision*, 2006.

[96] R. Socher, C. C. Lin, A. Ng, and C. Manning, "Parsing natural scenes and natural language with recursive neural networks," in *Proc. Int. Conf. Mach. Learning*, 2011, pp. 129–136.

[97] V. Lempitsky, A. Vedaldi, and A. Zisserman, "A pylon model for semantic segmentation," *Proc. Neural Inform. Process. Syst.*, 2011.

[98] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *Int. J. Comput. Vision*, vol. 59, no. 2, pp. 167–181, 2004.

[99] N. Silberman and R. Fergus, "Indoor scene segmentation using a structured light sensor," in *Proc. Int. Conf. Comput. Vision Workshops*. IEEE, 2011, pp. 601–608.

[100] D. Laptev, A. Vezhnevets, S. Dwivedi, and J. Buhmann, "Anisotropic sstem image segmentation using dense correspondence across sections," in *Proc. Medical Image Computing and Comput. Assisted Intervention*, 2012, pp. 323–330.

[101] M. Kearns and L. Valiant, "Cryptographic limitations on learning boolean formulae and finite automata," *J. Assoc. Comput. Mach.*, vol. 41, no. 1, pp. 67–95, 1994.

[102] A. Demiriz, K. Bennett, and J. Shawe-Taylor, "Linear programming boosting via column generation," *J. Mach. Learning Research*, vol. 46, pp. 225–254, 2002.

[103] S. Li and Z. Zhang, "Floatboost learning and statistical face detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 9, pp. 1112 –1123, Sep. 2004.

[104] R. E. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, pp. 197–227, 1990.

[105] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *European Conf. Comput. Learning Theory*, pp. 23–37, 1995.

[106] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Proc. Ann. Conf. Comput. Learning Theory*, pp. 80–91, 1998.

[107] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," *Ann. Stat.*, vol. 28, no. 2, pp. 337–407, 2000.

[108] H. Allende-Cid, R. Salas, H. Allende, and R. anculef, "Robust alternating adaboost," in *Progress Pattern Recog., Image Anal. and Applicat.*, vol. 4756/2007, 2007, pp. 427–436.

[109] A. Vezhnevets and V. Vezhnevets, "'modest adaboost' – teaching adaboost to generalize better," in *Proc. Int. Conf. Comput. Graphics and Vision*, 2005.

[110] E. Grossmann, "Adatree: Boosting a weak classifier into a decision tree," in *Proc. Comput. Vision and Pattern Recognition Workshop*, 2004.

[111] H. chuan Wang and L. ming Zhang, "A novel fast training algorithm for adaboost," *J. Fudan University*, vol. 1, 2004.

[112] J. Platt, "Resource-allocating network for function interpolation," *Neural Computation*, vol. 3, no. 2, pp. 213–225, 1991.

[113] K. Zhang and J. T. Kwok, "Density-weighted nystrm method for computing large kernel eigensystems," *Neural Computation*, vol. 21, no. 1, pp. 121–146, 2009.

[114] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*, 2nd ed. Wiley Interscience, 2000.

[115] K. Zhang and J. T. Kwok, "Simplifying mixture models through function approximation," *IEEE Trans. Neural Netw.*, vol. 21, no. 4, pp. 644–658, Apr. 2010.

[116] D. Comaniciu and P. Meer, "Mean shift: a robust approach toward feature space analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 5, pp. 603–619, May 2002.

[117] A. Gersho and R. M. Gray, *Vector Quantization and signal compression.* Kluwer Academic Publishers, 1992.

[118] A. Frank and A. Asuncion, "UCI machine learning repository," 2010. [Online]. Available: http://archive.ics.uci.edu/ml

[119] P. Brodatz, *Textures: A Photographic Album for Artists and Designers.* New York: Dover, 1966.

[120] L. Breiman, "Random forests," *Mach. learning*, vol. 45, no. 1, pp. 5–32, 2001.

[121] F. Schroff, A. Criminisi, and A. Zisserman, "Object class segmentation using random forests," in *Proc. British Mach. Vision Conf.*, 2008.

[122] A. Bosch, A. Zisserman, and X. Muoz, "Image classification using random forests and ferns," in *Proc. Int. Conf. Comput. Vision.* IEEE, 2007, pp. 1–8.

[123] P. Kontschieder, S. R. Bulò, A. Criminisi, P. Kohli, M. Pelillo, and H. Bischof, "Context-sensitive decision forests for object detection," in *Proc. Neural Inform. Process. Syst.*, 2012, pp. 440–448.

[124] A. Criminisi, J. Shotton, D. Robertson, and E. Konukoglu, "Regression forests for efficient anatomy detection and localization in ct studies," in *Medical Computer Vision. Recognition Techniques and Applications in Medical Imaging.* Springer, 2011, pp. 106–117.

[125] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. learning*, vol. 20, no. 3, pp. 273–297, 1995.

[126] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," *Proc. Neural Inform. Process. Syst.*, 1990.

[127] G. Ou and Y. L. Murphey, "Multi-class pattern classification using neural networks," *Pattern Recognit.*, vol. 40, no. 1, pp. 4–18, 2007.

[128] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Trans. Neural Netw.*, vol. 13, no. 2, pp. 415–425, 2002.

[129] A. Torralba, K. Murphy, and W. Freeman, "Sharing visual features for multiclass and multiview object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 5, pp. 854–869, 2007.

[130] A. Criminisi, J. Shotton, and E. Konukoglu, "Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning," *Foundations and Trends® in Computer Graphics and Vision*, vol. 7, no. 2-3, pp. 81–227, 2011.

[131] R. Caruana, N. Karampatziakis, and A. Yessenalina, "An empirical evaluation of supervised learning in high dimensions," in *Proc. Int. Conf. Mach. learning.* ACM, 2008, pp. 96–103.

[132] B. H. Menze, B. M. Kelm, D. N. Splitthoff, U. Koethe, and F. A. Hamprecht, "On oblique random forests," in *Mach. Learning and Knowledge Discovery in Databases.* Springer, 2011, pp. 453–469.

[133] J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso, "Rotation forest: A new classifier ensemble method," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 10, pp. 1619–1630, 2006.

[134] S. Bernard, L. Heutte, and S. Adam, "Forest-rk: A new random forest induction method," in *Advanced Intell. Comput. Theories Applicat. With Aspects of Artificial Intell.* Springer, 2008, pp. 430–437.

[135] E. E. Tripoliti, D. I. Fotiadis, and G. Manis, "Modifications of the construction and voting mechanisms of the random forests algorithm," *Data Knowl. Eng.*, vol. 87, no. 0, pp. 41 – 65, 2013.

[136] K. J. Cios and N. Liu, "A machine learning method for generation of a neural network architecture: A continuous id3 algorithm," *IEEE Trans. Neural Netw.*, vol. 3, no. 2, pp. 280–291, 1992.

[137] I. Ivanova and M. Kubat, "Initialization of neural networks by means of decision trees," *Knowledge-Based Systems*, vol. 8, no. 6, pp. 333–344, 1995.

[138] A. Banerjee, "Initializing neural networks using decision trees," 1997.

[139] R. Setiono and W. K. Leow, "On mapping decision trees and neural networks," *Knowledge-Based Systems*, vol. 12, no. 3, pp. 95–99, 1999.

[140] C. Olaru and L. Wehenkel, "A complete fuzzy decision tree technique," *Fuzzy sets and systems*, vol. 138, no. 2, pp. 221–254, 2003.

[141] O. Irsoy, O. T. Yildiz, and E. Alpaydin, "Soft decision trees," in *Proc. Int. Conf. Pattern Recognition*. IEEE, 2012, pp. 1819–1822.

[142] J. Wang and V. Saligrama, "Local supervised learning through space partitioning," in *Proc. Neural Inform. Process. Syst.*, 2012, pp. 91–99.

[143] M. J. Saberian and N. Vasconcelos, "Multiclass boosting: Theory and algorithms," in *Proc. Neural Inform. Process. Syst.*, 2011, pp. 2124–2132.

[144] G. Ratsch, T. Onoda, and K.-R. Muller, "Soft margins for adaboost," in *Mach. Learning*, 2000, pp. 287–320.

[145] A. Frank and A. Asuncion, "UCI machine learning repository," http://archive.ics.uci.edu/ml, 2010.

[146] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Trans. on Intelligent Syst. and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[147] R. Sznitman, C. J. Becker, F. Fleuret, and P. Fua, "Fast Object Detection with Entropy-Driven Evaluation," in *Proc. Comput. Vision and Pattern Recognition*, 2013.

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

[149] S. Vitaladevuni, Y. Mishchenko, A. Genkin, D. Chklovskii, and K. Harris, "Mitochondria detection in electron microscopy images," *Workshop Microscopic Image Anal. Appl. Biol*, 2008.

[150] K. Smith, A. Carleton, and V. Lepetit, "Fast ray features for learning irregular shapes," *Proc. Int. Conf. Comput. Vision*, 2009.

[151] A. Lucchi, K. Smith, R. Achanta, G. Knott, and P. Fua, "Supervoxel-based segmentation of mitochondria in em image stacks with learned shape features," *IEEE Tran. on Medical Imaging*, vol. 31, no. 2, pp. 474 –486, 2012.

[152] T. Tasdizen, J. P. Tarel, and D. B. Cooper, "Algebraic curves that work better," *Proc. Comput. Vision and Pattern Recognition*, vol. 2, pp. 35–41, 1999.

[153] T. Tasdizen and D. B. Cooper, "Boundary estimation from intensity/color images with algebraic curve models," *Proc. Int. Conf. Pattern Recognition*, vol. 1, pp. 225 –228, 2000.