Building an Automatic Phenotyping System of Developing Embryos

by

Feng Ning

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Courant Institute of Mathematical Sciences

New York University

September 2006

Yann LeCun

人书俱老

The publication will age with the author.

Dedicated to my wife, Karen, who delights my life

and

my parents, Yuanlai Ning and Liangfang Zhang, who give me life and strength.

Acknowledgements

I sincerely thank my adviser professor Yann LeCun. I am indebted of courage and support from him in the long journey of exploring the unknown and discovering my inner identity.

I would to thank all the people I am working with. Special thanks to the thesis committee members who spend their valuable time reviewing my thesis.

Last but not the least, the staff in the CS department provide a great environment for my study and work. I appreciate their efforts.

Abstract

This dissertation presents a learning-based system for the detection, identification, localization, and measurement of various sub-cellular structures in microscopic images of developing embryos. The system analyzes sequences of images obtained through DIC microscopy and detects cell nuclei, cytoplasm, and cell walls automatically. The system described in this dissertation is the key initial component of a fully automated phenotype analysis system.

Our study primarily concerns the early stages of development of *C. Elegans* nematode embryos, from fertilization to the four-cell stage. The method proposed in this dissertation consists in learning the entire processing chain *from end to end*, from raw pixels to ultimate object categories.

The system contains three modules: (1) a convolutional network trained to classify each pixel into five categories: cell wall, cytoplasm, nuclear membrane, nucleus, outside medium; (2) an Energy-Based Model which cleans up the output of the convolutional network by learning local consistency con-

straints that must be satisfied by label images; (3) A set of elastic models of the embryo at various stages of development that are matched to the label images.

When observing normal (*wild type*) embryos it is possible to visualize important cellular functions such as nuclear movements and fusions, cytokinesis and the setting up of crucial cell-cell contacts. These events are highly reproducible from embryo to embryo. The events will deviate from normal behaviors when the function of a specific gene is perturbed, therefore allowing the detection of correlations between genes activities and specific early embryonic events. One important goal of the system is to automatically detect whether the development is normal (and therefore, not particularly interesting), or abnormal and worth investigating. Another important goal is to automatically extract quantitative measurements such as the migration speed of the nuclei and the precise time of cell divisions.

Contents

List of Figures

List of Appendices

Chapter 1

Introduction

1.1 Motivation

One of the major goals of biological research in the post-genomic era is to characterize the function of every gene in the genome. One particularly important subject is the study of genes that control the early development of animal embryos. Such studies often consist in knocking down one or several genes and observing the effect on the developing embryo, a process called *phenotyping*.

As an animal model, the nematode *C.elegans* is one of the most amenable to such genetic analysis because of its short generation time, small genome size, and availability of a rapid gene knock-down approach, RNAi (RNA interface) [FXM+98].

Since the completion of the *C.elegans* genome sequence and identification of its roughly 20,000 protein-coding genes in 1998 [con98], extensive research has been done on analyzing how these genes function *in vivo*. Early embryonic events provide a good model to assess specific roles genes play in a developmental context. Early *C.elegans* embryos are easily observable under a microscope fitted with Nomarski Differential Interference Contrast (DIC) optics. When observing normal (*wild type*) embryos it is possible to visualize important cellular functions such as nuclear movements and fusions, cytokinesis and the setting up of crucial cell-cell contacts. These events are highly reproducible from embryo to embryo and deviate from normal behaviors when the function of a specific gene is depleted [GEO$^+$00, PSM$^+$00, PSM$^+$02, ZFK$^+$01], allowing the association of a gene's activity with specific early embryonic events.

A typical experiment consists in knocking down a gene (or a set of genes), and recording a time-lapse movie of the developing embryo through DIC microscopy. Figure 1.1 shows a few frames extracted from the movie of a normally developing *C. elegans* embryo from the fusion of the pronuclei to the four-cell stage.

Using RNAi, several research groups have gathered a large collection of such movies. Many of these movies depict cellular behaviors in the early embryos that deviate from the wild type, and some show dramatic prob-

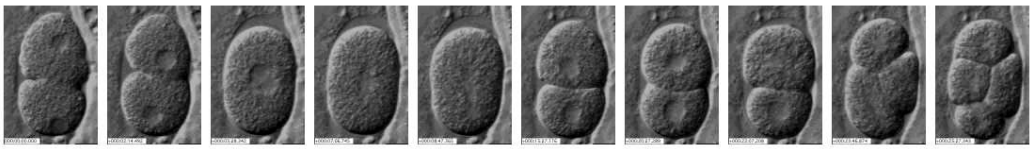

Figure 1.1: Snapshots of the early development stages of a wild type *C.elegans* embryo obtained through DIC microscopy.

lems during embryonic development. Although initial analysis of the movies have been performed by hand, automating the analysis of the cellular behaviors would augment our ability to process the large amounts of data being currently produced, and could reveal more subtle quantitative defects that cannot be detected by manual analysis.

One important classification task is to automatically detect whether the development is normal (and therefore, not particularly interesting), or abnormal and worth investigating. Another important task is to automatically extract quantitative measurements such as the number of cells, the relative positions of the cell nuclei, the time between each cell division, etc... Ultimately, one may want an automated system for classifying the movies into a number of known scenarios of normal or abnormal development.

1.2 Automatic Phenotyping System

We will present our automatic system, which focuses on the detection, identification, and measurement of various objects of interests in the embryo, namely the cell nuclei, cytoplasms, and cell walls, from sequences of images obtained through DIC microscopy. The system described in this thesis is the key component of a fully automated analysis system under development. The present study primarily concerns the early stages of development, from fertilization to the four-cell stage.

Although the development of *C. elegans* embryos is the subject of numerous studies from biologists, there have been very few attempts to automate the task of analyzing DIC image sequences. The most notable exception is the work of Yasuda et al. [YBO+99], which describes a computer vision approach to the detection the nuclei and cell walls. Their method is based on the combination of several types of edge features. Because DIC microscopy images are very noisy and anisotropic, the method produces a large number of false positives (e.g. areas falsely detected as cell nuclei) that must be manually corrected. One conclusion from this work is that DIC images are not easily analyzed with commonly-used feature detectors. In this paper, we propose to rely on machine learning methods to produce a more reliable image segmentation system.

Learning methods have been used for low-level image processing and segmentation with some success over the last few years. A notable example is the object boundary detection system of Martin et al. [MFTM01, MCM04]. Closer to our application is the detection and classification of sub-cellular structures in fluorescence microscopy images. Machine learning and adaptive pattern recognition methods have been widely applied to this problem in a series of influential work [BMM98, HM04]. These systems rely on the time-honored method of extracting a large number of carefully engineered features, while using learning methods to select and exploit these features.

The method proposed in this thesis consists in learning the entire processing chain *from end to end*, from raw pixels to ultimate object categories. The system is composed of three main modules.

The first module is a trainable *Convolutional Network*, which labels each pixel in a frame into one of five categories. The categories are: cell nucleus, nuclear membrane, cytoplasm, cell wall, and outside medium. The main advantage of Convolutional Nets is that they can learn to map raw pixel images into output labels, synthesizing appropriate intermediate features along the way, and eliminating the need for manually engineered features. They have been widely applied to detection and recognition tasks such as handwriting recognition with integrated segmentation (see [LBBH98] for a review), hand tracking [NP95], face recognition [LGTB97], face detec-

tion [VML94, GD04, OML05], and generic object recognition [HLB04]. The main advantages of Convolutional Networks is that they can operate directly on raw images

The architecture of the convolutional network is designed so that each label can be viewed as being produced by a non-linear filter applied to a 41×41 pixel window centered on the pixel of interest in the input image. This convolutional network is trained in supervised mode from a set of manually labeled images. The five categories may appear somewhat redundant: it would be sufficient to label the nucleus, cytoplasm, and external medium to locate the nuclear membrane and the cell wall. However, including the boundaries as explicit categories introduces redundancy in the label images that can be checked for consistency.

Ensuring local consistency is the role of the second module. Since the label of each pixel is produced independently of the labels of neighboring pixels, the predicted label image may indeed contain local inconsistencies. For example, an isolated pixel in the outside medium may be erroneously classified as nucleus. Since nucleus pixels must be surrounded by other nucleus pixels or by nuclear membrane pixels, it would seem possible to clean up the label image by enforcing a set of local consistency constraints. To implement this process, we used an *energy based model* (EBM) [TWOE03, YH05]. EBMs are somewhat similar to Markov Random Fields, and can be seen as

a sort of non-probabilistic Conditional Random Field [LMP01]. The EBM used in the present system can be viewed a scalar-valued "energy" function $E(f(X), Y)$, where $f(X)$ is the label image produced by the convolutional net, and Y is the cleaned-up image. The EBM is trained so that when $f(X)$ is a predicted label image and Y is the corresponding "correct" (cleaned-up) label image, the energy $E(f(X), Y)$ will be smaller than for any other ("incorrect") value of Y. The cleanup process consists in searching for the Y that minimizes $E(f(X), Y)$ for a given $f(X)$. This approach is related to the relaxation labeling method [HZ87]. While learning methods have been used to estimate the coupling coefficients in relaxation labeling systems [PR94], the method used here is based on minimizing a new type of contrastive loss function [YH05].

The third component of the system models the embryos and their internal parts by matching deformable templates to the label images. This module is used to precisely locate and count parts such as cells nuclei, and cell walls. It is also used to determine the stage of development of the embryo in the image. This technique is related to the classical active contour method [KWT87, MT96], and very similar to elastic matching methods based on the Expectation-Maximization algorithm as described in [RWH96, BL94].

A preliminary version of the system presented in this thesis was published

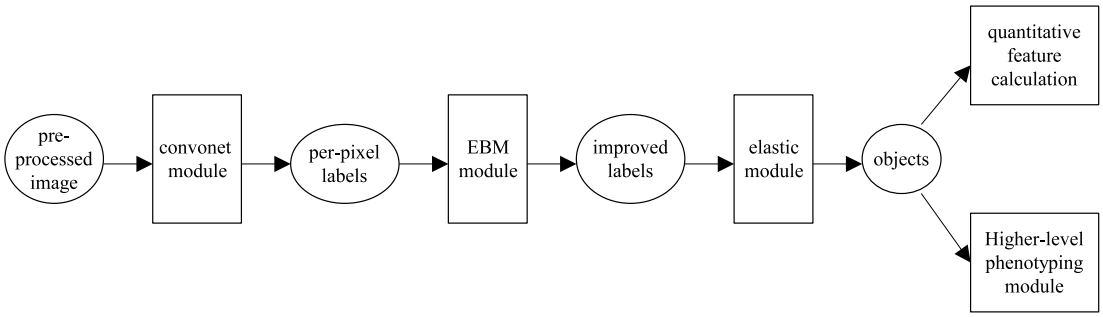

Figure 1.2: The architecture of the entire automatic system, along with a high level description of data flow through the modules.

in [NDL+05]. The system and methods described here contains a number of significant differences with this earlier work:

- Both training data set and testing data set are more than $4x$ larger. The complexity/capacity of the learning system was increased only marginally. The additional training data suppressed the potential effects of increased complexity on overfitting. Our system shows better reliability on testing data with great variability of image statistics.

- The learning/inference algorithms for the energy-based machine module were revised for clarification and efficiency.

- The elastic templates were redesigned and the fitting algorithm was updated accordingly.

1.3 Structure of the Thesis

The thesis is organized as described below. Chapter 1 gives the introduction. Chapter 2, chapter 4, chapter 5 describe in depth the three modules of the system: the convolutional network module, the energy-based machine module, and the elastic fitting module, respectively. Both design and empirical studies are presented in each of the chapters. Chapter 3 is a review of related works. We put our research problem in a generic setting, where we consider the inference of true labels from noisy observations. A series of frameworks are represented, in a way that highlights their motivation and the relationships between them. Chapter 6 gives an outlook for future work and outlines several interesting direction where our work may be extended.

1.4 Conventions

Throughout this thesis, we use a, b, c to denote scalar variables and $\mathbf{a}, \mathbf{b}, \mathbf{c}$ to denote multi-dimensional variables. For most parts of our discussion, we focus on image data which have inherently a 2D matrix representation. For example, for image $\mathbf{x}$, we have a matrix $x_{i,j} : i = 1, .., m, j = 1, .., n$. If we assume natural ordering on the index, we can create a unique 1D representation $x_{1,1}, x_{1,2}, .., x_{1,n}, x_{2,1}, x_{2,2}, .., x_{m,n}$. We do not explicitly state which representation (1D or 2D) used in a particular formula, if self-evident

from the context.

ROI is a shorthand for *region of interest*. In the thesis, we refer to any cell/nucleus or other cellular or cubcellular structure in an image as an ROI.

Chapter 2

The Convolutional Network Module

We present the first module of our system, the convolutional network module. This chapter is divided into two parts. The first part is an introduction of the architectural design of the module. The second part is focused on the empirical study, including the data preparation, the experiments and the analysis of the results.

2.1 Convolutional Network

A Convolutional Network is a trainable system whose architecture is specifically designed to handle images or other 1D or 2D signals with strong local

correlations. A Convolutional Network can be seen as a cascade of multiple non-linear local filters whose coefficients are learned to optimize an overall performance measure. Convolutional Networks have been applied with success to a wide range of applications [LBBH98, NP95, LGTB97, VML94, GD04, OML05, HLB04].

Convolutional Networks are specifically designed to handle the variability of 2D shapes. They use a succession of layers of trainable convolutions and spatial subsampling interspersed with sigmoidal non-linearities to extract features with increasingly large receptive fields, increasing complexity, and increasing robustness to irrelevant variabilities of the inputs. The convolutional net used for the experiments described in this paper is shown in figure 2.1.

Each convolutional layer is composed of a set of planes called *feature maps*. The value at position (x, y) in the j-th feature map of layer i is denoted c_{ijxy}. This value is computed by applying a series of convolution kernels w_{ijk} to feature maps in the previous layer (with index $i-1$), and passing the result through a sigmoid function. The width and height of the convolution kernels in layer i are denoted P_i and Q_i respectively. In our network, the kernel sizes are between 2 and 7. More formally, c_{ijxy} is computed as:

$$c_{ijxy} = \tanh\left(b_{ij} + \sum_{k} \sum_{p=0}^{P_i-1} \sum_{q=0}^{Q_i-1} w_{ijkpq} c_{(i-1),k,(x+p),(y+q)}\right) \qquad (2.1)$$

where p, q index elements of the kernel w_{ijk}, tanh is the hyperbolic tangent function, i is the layer index, j is the index of the feature map within the layer, k indexes feature maps in the previous layer, and b_{ij} is a bias. Each feature map is therefore the result of a sum of discrete convolutions of the previous layer maps with small-size kernels, followed by a point-wise squashing function. The parameters w_{ijkpq} and b_{ij} are all subject to learning.

Subsampling layers have the same number of feature maps as the convolutional layer that precedes them. Each value in a subsampling map is the average of the values in a 2×2 neighborhood in the corresponding feature map in the previous layer. That average is added to a trainable bias, multiplied by a trainable coefficient, and the result is passed through the tanh function. The 2×2 windows are stepped without overlap. Therefore the maps of a subsampling layer are one half the resolution of the maps in the previous layer. The role of the subsampling layers is to make the system robust to small variations of the location of distinctive features.

Figure 2.1 only shows a portion of the network: the smallest portion necessary to produce a single output label. Each output is influenced by a 41×41 pixel window on the input. The full network can be seen as multiple replicas of this network applied to all 41×41 windows stepped every 4 pixels on the input image (more on this later). The window size was chosen so that the system would have enough context information to make an informed

decision about the category of a pixel. For example, the local texture in the nucleus region is often indistinguishable from that of the external medium. Therefore, distinguishing nucleus pixels from external medium pixels can only be performed by checking if the pixel is within a roughly circular region surrounded by cytoplasm. Since the nuclei are typically less than 41 pixels in diameter, we set the window size to 41×41 to ensure that at least some of the nuclear membrane and the cytoplasm will be present in every window containing nucleus pixels. Once the input window size is chosen, the choice of the kernel size and subsampling ratio for each layer is quite constrained. The first layer (marked C1) contains 6 feature maps with 7×7 pixel convolution kernel. The second layer (S2), is a subsampling layer with 2×2 subsampling ratios. The third layer (C3) uses 6×6 convolution kernels. Each of the 16 maps in C3 combines data from several maps in S2 by applying a separate convolution kernel to each map, adding the results, and applying the sigmoid. Each feature variable in C3 is influenced by an 18×18 pixel window on the input. Each C3 map combines input from a different subset of of S2 maps, with a total of 61 individual kernels. Layer S4 is similar to S2 and subsamples C3 by a factor of 2. Layer C5 comprises 40 feature maps that use 6×6 convolution kernels. There is one kernel for each pair of feature map in S4 and C5. The output layer contains five units, one for each category.

One key advantage of convolutional nets is that they can be applied to

images of variable size. Applying the network to a large image is equivalent (but considerably cheaper computationally) to applying a copy of the single-output network to every 41×41 window in the input stepped every 4 pixel. More precisely, increasing the input size by 4 pixels in one direction will increase the size C1 by 4 pixels, S2 and C3 by 2 pixel, and S4, C5, and the output by 1 pixel. The size of the output in any dimension is therefore $(N - 36)/4$, where N is the size of the input image in that dimension. Consequently, the convolutional net produces a labeling for every 4×4 block of pixels in the input, taking information from a 41×41 window centered on that block of pixels. Figure 2.1 shows the size of each layer when a 41×41 pixel input is used and a single output vector is produced. Figure 2.8 shows the result of applying the convolutional network to an image, which produces a label image with 1/4 the resolution of the input. It would be straightforward to modify the method to produce a label image with the same resolution as the input. However, we determined that the current application did not require pixel-level accuracy.

Our task is to assign a label for each pixel on the image. Ideally, we should build a Convolutional Network for each label produced, whose reception field covers the entire image to utilize all information available. However, it is not a good design in practice for several reasons: (1) the typical size of an image is several hundred pixels per each dimension, it is inefficient to train

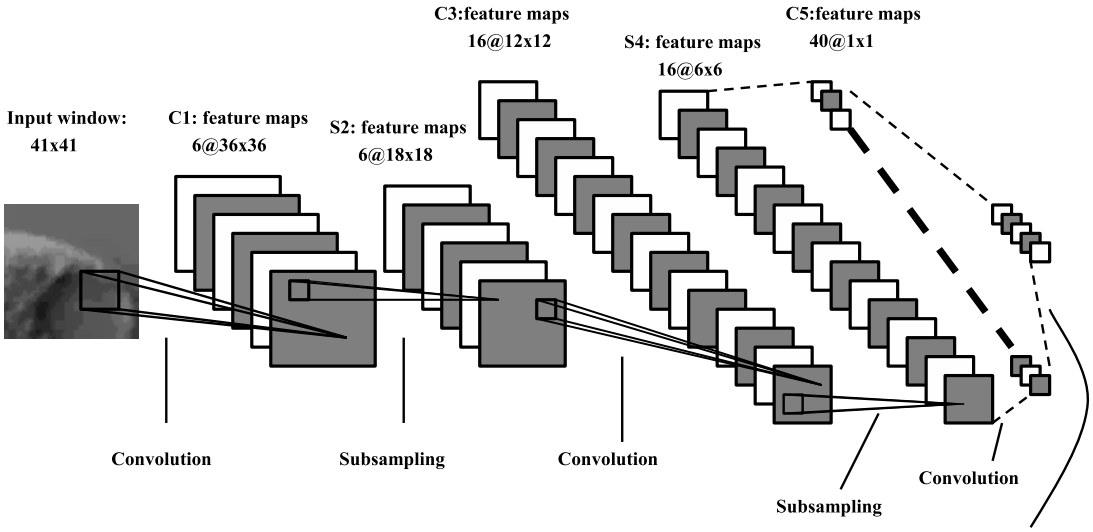

Figure 2.1: The Convolutional Network architecture. The feature map sizes indicated here correspond to a 41×41 pixel input image, which produces a $1 \times \ 1$ pixel output with 5 components each. Applying the network to an $N_x \times N_y$ pixel image will result in output maps of size $(N_x - 40) \times (N_y - 40)$.

a Convolution Network with so large an input size; (2) To assign a label to a pixel, nearby pixels are much more informative than remote pixels. It is more economical to constrain ourselves to local neighborhoods, instead of the entire image.

Our design is based on a Convolutional Network operating on *image windows*. The typical size is 41×41 in our implementation, centered at the pixel whose label is of interest. Our module also provides a scheme to obtain all the windows from the images. For inference, we swipe the images one pixel a time, so that we can infer labels for all pixels (excluding boundary). For training, we add some policy for class balancing (see below).

2.2 Convolutional Network Module

2.2.1 Datasets and Training

Training images were extracted from 10 different movies of *C. elegans* embryos. 20 frames were extracted from each movie, every 10 frames, for a total of 200 frames. Testing images were extracted from another set of 5 movies. Similarly, 20 frames were extracted (separated by 10 frames) from each testing movie, for a total of 30 frames. The sample frames were picked every 10 frames in the movies so as to have a representative set covering the

various stages of embryonic development. See the following fig 2.2.1, where we randomly (not necessarily ordered image frames by timestamp) show four images for each movie in training and in testing. Image size varies and many development stages are observed. Here we show the raw pixels instead of the pre-processed (isotropic) pixels since the raw pixels are visually more vivid to the human eye.

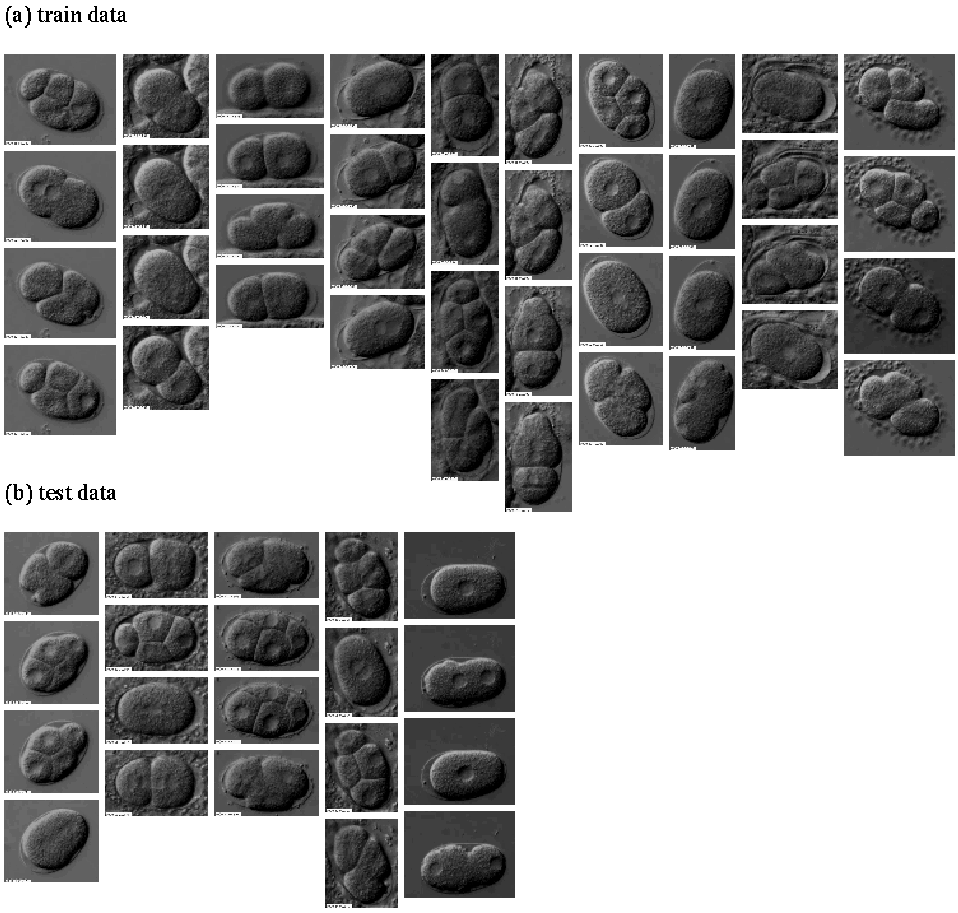

Figure 2.2: Sample input images.

Frames from different movies had different sizes, but were typically around several hundred pixels. All images were 8-bit gray-scale. The movies were stored in Apple Quicktime format, whose compression method introduces some quantization and blocking artifacts in the frames. Working with compressed video make the problem more difficult, but it will allow us to tap into a larger pool of movies produced by various groups around the world, and distributed in compressed formats.

Preprocessing

DIC images are not only very noisy, but also very anisotropic. The DIC process creates an embossed "bas relief" look that, while pleasing to the human eye, makes processing the images quite challenging. For example the cell wall in the upper left region of the raw image in figure 2.8 looks quite different from the cell wall in the lower right region. We decided to design a linear filter that would make the images more isotropic, while preserving the texture information. The linear filter used was equivalent to computing the difference between the image and a suitably shifted version of it. A typical resulting image is shown in figure 2.2.1. The pixel intensities were then centered so that each image had zero mean, and scaled so that the standard deviation was 1.0. An unfortunate side effect of this pre-processing is that it makes the quantization artifacts of the video compression more

apparent. Better preprocessing will be considered for future embodiments of the system. It should be emphasized that the purpose of this preprocessing is merely to make the image features more isotropic. The purpose is not to recover the optical pathlength, as several authors working with DIC images have done [vvA97].

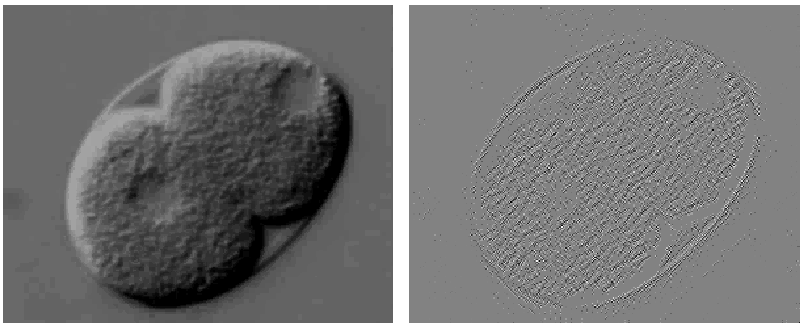

Figure 2.3: Input preprocessing. Left: raw image; right: preprocessed image.

Labels

Each training and testing frame was manually labeled with a simple graphical tool (*CellSmartGui*, see A) by a single person. Labeling the images in a consistent manner is very difficult and tedious. Therefore, we could not expect the manually produced labels to be perfectly consistent. In particular, it is very common for the position nucleus boundary or the cell wall to vary by several pixels from one image to the next.

Consequently, it appeared necessary to use images of desired labels that

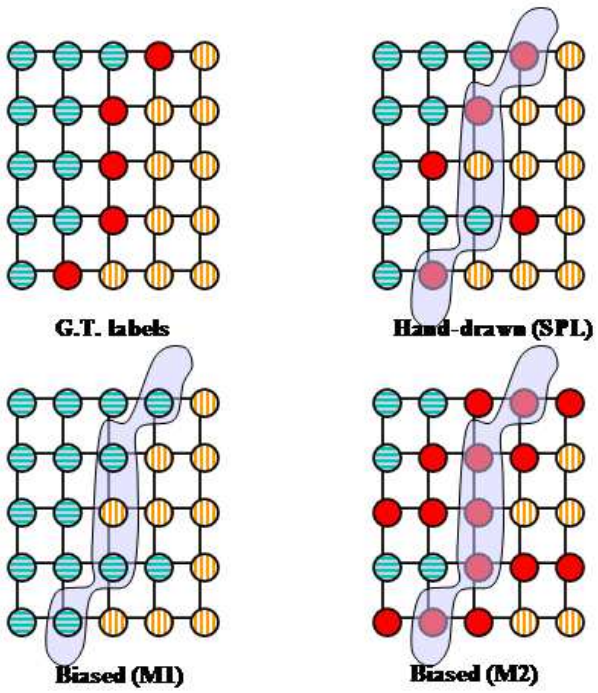

Figure 2.4: Generation of M1 and M2 labels. The ground truth label image (GT) is assumed unknown. The human-produced labels may contain error and inconsistencies. The M1 label image is produced by making the boundary 3 pixel wide so as to encompass the ground truth, whereas the M2 label image is derived from the human produced labels by removing all boundary pixels.

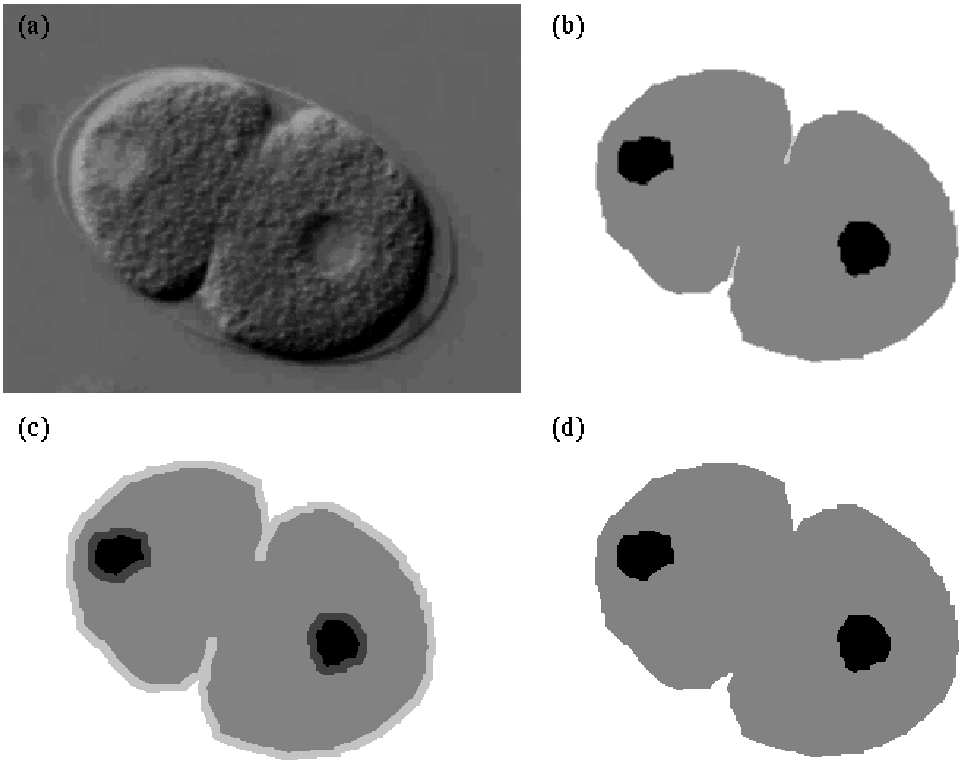

Figure 2.5: Comparison of raw images and different labels. (a) raw image; (b) human labels; (c) M1 labels; (d) M2 labels.

could incorporate a bit of slack in the position of the boundaries. We used a very simple method which consists in deriving two label images from each human-produced label image. The process is described in figure 2.2.1 and what really happen on a real image can be seen in figure 2.2.1. The first image, called M1, is obtained by dilating the boundaries by one pixel on each side, thereby producing a 3-pixel wide boundary. The second label image, M2, contains no boundary labels. It is obtained by turning all the nuclear membrane pixels into either nucleus or cytoplasm using a simple nearest neighbor rule.

Training Set and Test Set

The simplest way to train the system would be to simply feed a whole image to the system, compare the full predicted label image to the ground truth, and adjust all the network coefficients to reduce the error.

This "whole-image" approach has two deficiencies. First, there are considerable differences between the numbers of pixels belonging to each category. This may cause the infrequent categories to be simply ignored by the learning process. Second, processing a whole image at once can be seen as being equivalent to processing a large number of 41×41 pixel windows in a batch. Previous studies have shown that performing a weight update after each sample leads to faster convergence than updating the weights after accumulating

gradients over a batch of samples [LBBH98]. Therefore, we chose to break up the training images into a series of overlapping 41×41 windows that can be processed individually. Overall, from the 200 frames in the training set, $9,931,780$ windows (note: this is not the number of windows the trainer see for each epoch, due to the frequency equalization, as described below) of size 41×41 pixels were extracted. To each such window was associated the desired labels (for M1 and M2) of the central pixel in the window. Each pair of window and label was used as a separate training sample for the convolutional network, which therefore produced a single output vector (a 1 pixel output map). There were wide variations in the number of training samples for each category. A quick estimate shows that there are about 10 times more windows labeled as external medium than those labeled as nucleus. If we train the module with data of such wide variations, the module will generate too many false positives of external medium and false negatives of nucleus for un-equalized data. However, we are more concerned about the precision of locating nucleus than external medium. To correct these wide variations, a class frequency equalization method was used. During one epoch, each sample labeled "external medium" was seen once, while samples from the other categories were repeated c times, where c is constant within each category. c is chosen to be inversely related to ratio of the number of windows of the category and the number of windows of external medium. And we also cap

the maximum value of c to be 10.0 to avoid excessive long training time. It is worthy to point out that if we have two schemes to label the same images (e.g. use M1 and M2), c values will be different. As result, the corresponding training time for one epoch can be different.

The network was trained to minimize the mean squared error between its output vector and the target vector for the desired category. The target vectors were $[+1, -1, -1, -1, -1]$ for nucleus, $[-1, +1, -1, -1, -1]$ for nucleus membrane, $[-1, -1, +1, -1, -1]$ for cytoplasm, $[-1, -1, -1, +1, -1]$ for cell wall, and $[-1, -1, -1, -1, +1]$ for external medium. The training procedure used a variation of the back-propagation algorithm to compute the gradients of the loss with respect to all the adjustable parameters, and used an "on-line" version of the Levenberg-Marquardt algorithm with a diagonal approximation of the Hessian matrix to update those parameters (details of the procedure can be found in [LBBH98]).

2.2.2 Results

The network was trained for 5 epochs on the frequency-equalized dataset for two times, one with the M1 labels and another with the M2 labels. Each epoch takes 45 and 30 hours on a 2.0 GHz *Opteron*-based server respectively. We measured the following pixel-wise error rates, see figure 2.6. The training

error is measured on the *frequency equalized* training set. The test error is measured on the *non frequency equalized* training or testing data set.

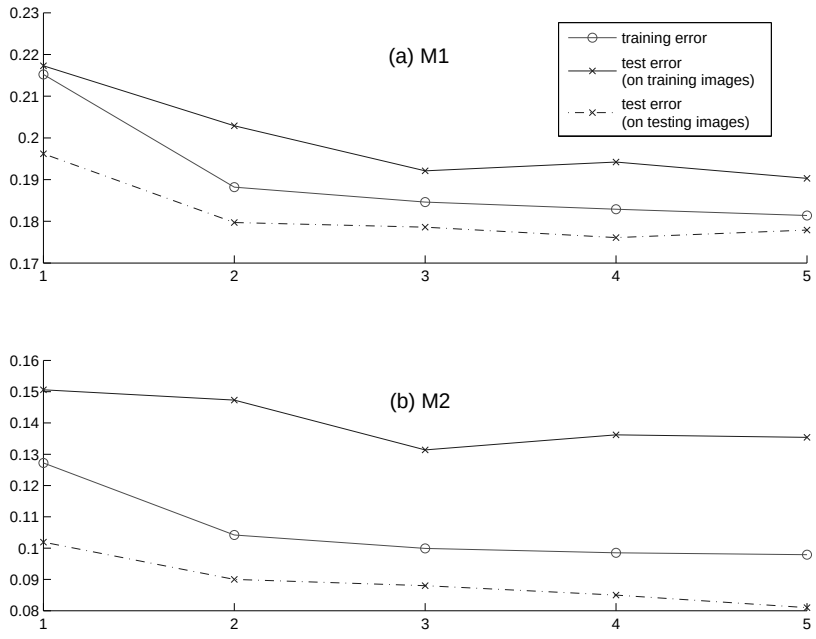

Figure 2.6: The pixelwise error rates for (a) M1 and (b) M2. x-axis: number of epochs of training; y-axis: pixelwise error rates (in percentage).

Some remarks for fig 2.6:

- The error rates are lower in M2 than in M1, since M2 deals with 3-category classification while M1 deals with 5-category.

- The training errors are the error rates we obtain from the training pro-

cess, by feeding the *frequency-equalized* windows to the machine. That is why the error rates are different from the testing error on the training images, where we do not adjust the frequency of the windows.

- The testing error rates are slightly *lower* on the testing images than on the training images.

  This may look suspicious at first sight. But overall, all the error rates are quite similar and, stabilize after decreasing nicely. The lower error rates can be the consequence of how we chose the training and testing images. Currently we chose 10 movies randomly for training and the remaining 5 for testing. However, the mechanical conditions of the DIC microscope and the phenotyping process under observations are not considered for the choice. So our arbitrary choice may not guarantee the randomness. A more rigorous error analysis could be applied, such as leave-one-out cross-validation. But it is very expensive for our problem and, as we point out, the absolute error rate on individual windows is somewhat removed from the ultimate system-level performance.

We also analyzed the per-category training error rates along with the overall training error rates, see figure 2.7, We conclude that the training not only decreases the overall training error rates, but also decreases the variance

of per-category error rates. This property is essential in the testing phase since the data is highly imbalanced. It will be high undesirable if the trainer minimizes only minimize some but not all per-category error rates.

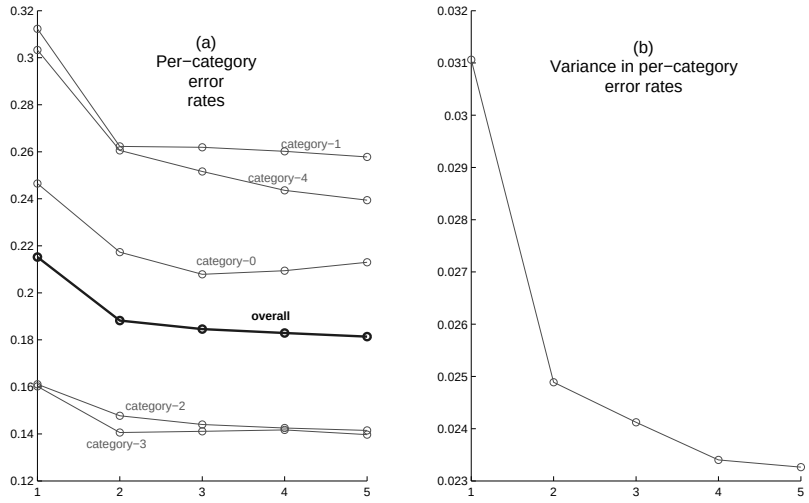

Figure 2.7: The per-category training error analysis. x-axis: number of epochs of training. The error rate for category-i is defined as $1 - \frac{\text{all windows predicted as label } i \text{ with ground truth label } i}{\text{all windows with ground truth label } i}$

It must be emphasized again that pixel-wise error rate is *not* a good indicator of the overall system performance. First of all, many errors are isolated points that can easily be cleaned up by post-processing. Second, it is unclear how many of the errors can be attributed to inconsistencies in the human-produced labels, and how many can be attributed to truly inaccurate classifications. Third, and more importantly, the usefulness of the overall

system will be determined by how well the cells and nuclei can be detected, located, counted, and measured.

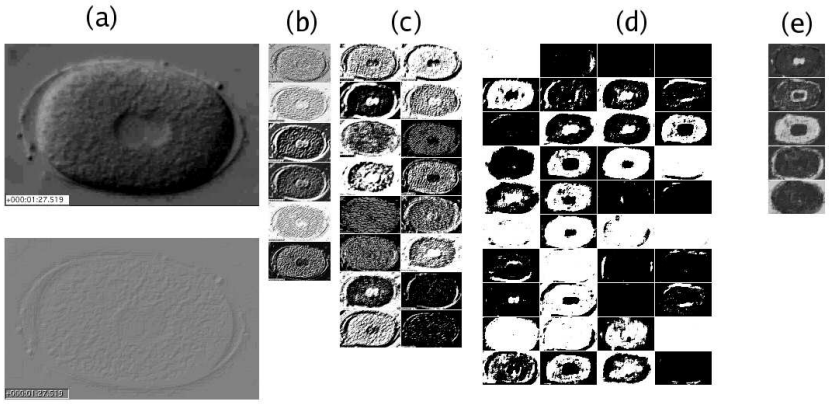

Figure 2.8: The convolutional network module in action. (a) top: raw input image; bottom: pre-processed image; (b) state of layer C1; (c) layer C3; (d): layer C5; (e): output layer. The five output maps correspond to the five categories, respectively from top to bottom: nucleus, nucleus membrane, cytoplasm, cell wall, external medium. The properly segmented regions are clearly visible on the output maps.

Figure 2.8 shows a sample image (top left), a pre-processed version of the image (bottom left), and the corresponding internal state and output of the convolutional network. Layers C1, C3, C5 and F6 (output) are shown. The segmented regions of the five categories (nucleus, nuclear membrane, cytoplasm, cell wall, external medium) are clearly delineated in the five output maps.

The labeling produced by the network for several sample images, from training and testing data, shown in figure 2.9 and figure 2.10. The essential elements of the embryos are clearly detected. The cell nuclei are correctly labeled before, during, and after the fusion of the pro-nuclei. The cell wall is correctly identified by the M1 network. However, the detection of new cell walls created during cell division (mitosis) seems to be more difficult.

2.3 Summary

The main advantage of the convolutional network approach is that the low-level features are automatically learned. A somewhat more traditional approach to classification consists of selecting a small number of relevant features from a large set. One popular approach is to automatically generate a very large number of simple "kernel" features, and to select them using the Adaboost learning algorithm [VJ01]. Another popular approach is to build the feature set by hand. This approach has been advocated in [CEW+04] for the classification of sub-cellular structures. We believe that these methods are not directly applicable to our problem because the regions are not well characterized by local features, but depend on long-range context (e.g. a nucleus is surrounded by the cytoplasm). This kind of contextual information is not easily encoded into a feature set.

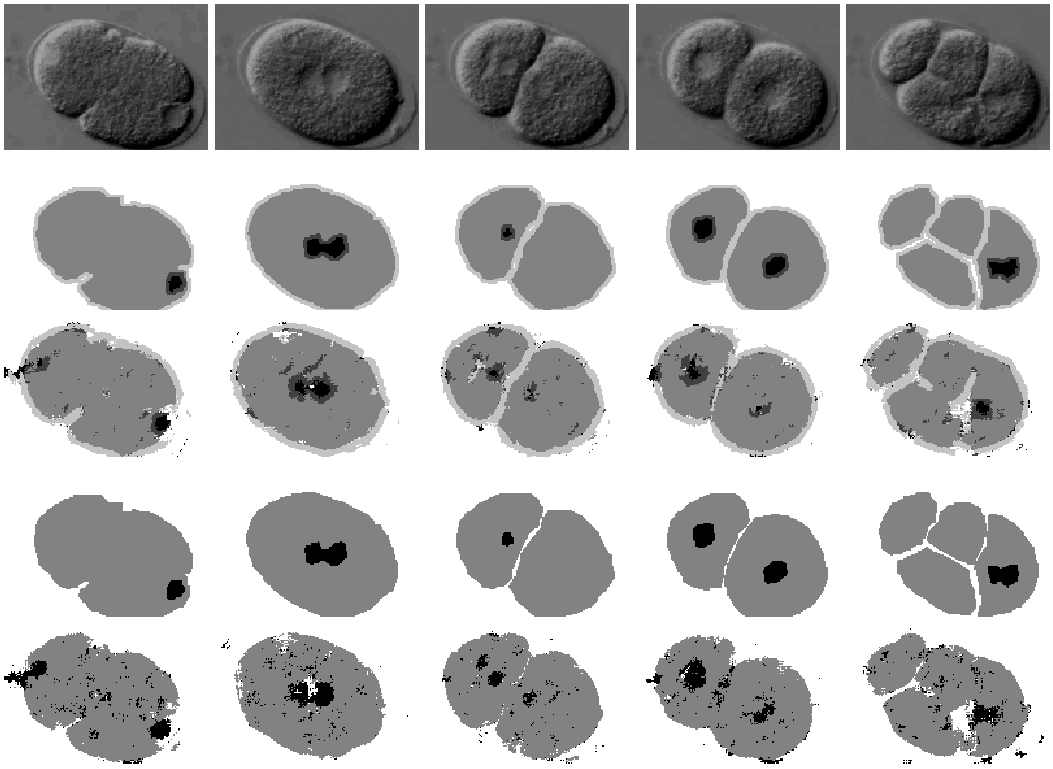

Figure 2.9: Pixel labeling produced by the convolutional network module. Top to bottom: input images; label images produced by the network trained with M1 labels; label images produced by the M2 network Because each output is influenced by a 41×41 window on the input, no labeling can be produced for pixels less than 20 pixels away from the image boundary.

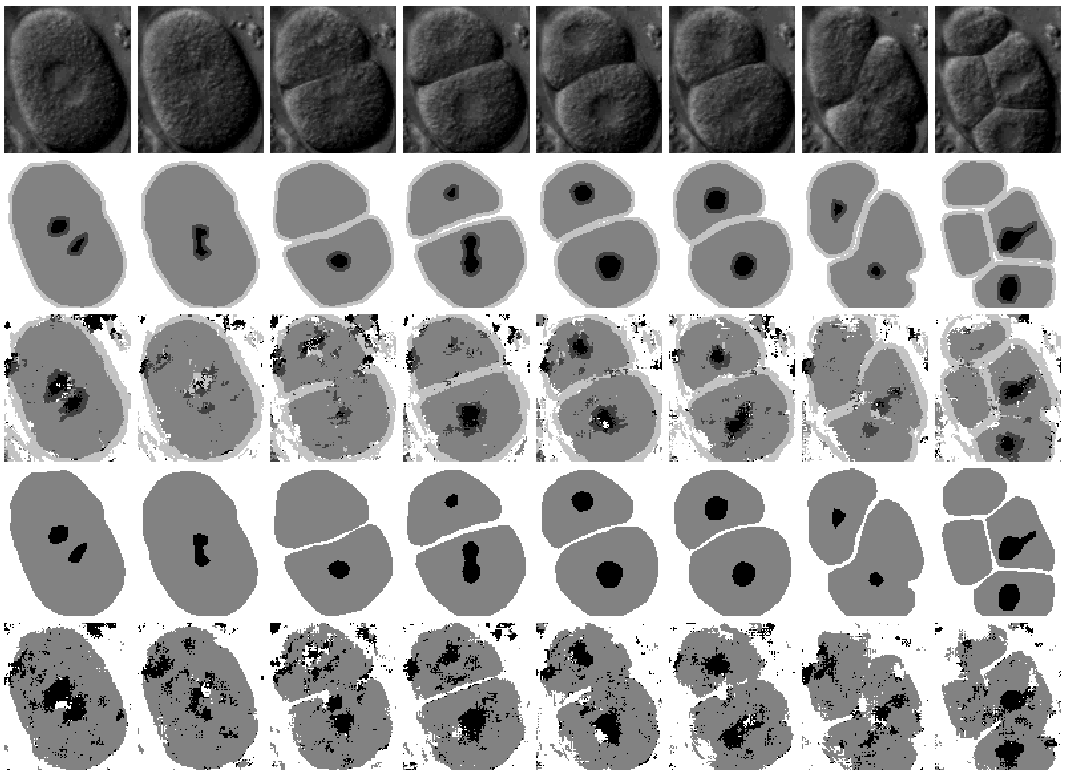

Figure 2.10: The M1, M2 predictions on testing images

Chapter 3

Related Work

The task of inferring states $\mathbf{Y}$ from a set of observations $\mathbf{X}$ arises in many fields, including computational linguistics (1D data) and image labeling/segmentation (2D data). Such inference tasks are often a useful preprocessing step for higher level processing tasks. Our labeling problem can be regarded as a special case, where labels are one-to-one correspondent to the states.

This chapter serves as a review of related works and is organized as follows. Section 3.1 is an introduction to probabilistic approaches. Then we discuss the discriminative framework in section 3.1.3. Practical implementations involve advanced sampling techniques and Contrastive Divergence learning, an approximate learning scheme, which are the topics in section 3.2, 3.3. Factor graph, a non-probabilistic framework. is discussed next in section 3.4,

in the context of graphical models.

3.1 Probabilistic Approaches

Probabilistic approaches model the probability distribution over $\mathbf{X}$ and $\mathbf{Y}$. The model is called *generative* when it models the joint probability, or the conditional probability $P(\mathbf{X}|\mathbf{Y})$. The model is called *discriminative* if it directly models the conditional distribution $P(\mathbf{Y}|\mathbf{X})$. The two types of models try to address a different aspect of the problem, but as we will see next, they are closely related.

3.1.1 Generative Framework

In the generative framework, the joint probability $P(\mathbf{Y}, \mathbf{X})$ is the target of modeling. Using Bayes rule, the posterior over labels $P(\mathbf{Y}|\mathbf{X}) \propto P(\mathbf{Y}, \mathbf{X}) = P(\mathbf{X}|\mathbf{Y})P(\mathbf{Y})$. $P(\mathbf{Y})$ is commonly modeled as Markov Random Field (MRF) [Li95] to incorporate local contextual information. For the sake of tractability, the conditional probability $P(\mathbf{X}|\mathbf{Y})$ is often assumed factorized $P(\mathbf{X}|\mathbf{Y}) = \Pi_i P(\mathbf{X}_i|\mathbf{Y}_i)$. Unfortunately, this assumption is generally seen as too restrictive for modeling natural images [CB01, WL03].

It is worth noting that some authors have attempted to relax the assumption that $P(\mathbf{Y})$ is an MRF [PT00]. Instead, the joint $(\mathbf{Y}, \mathbf{X})$ is assumed to

be an MRF. However, the underlying difficulty is not resolved but merely displaced.

3.1.2 Discriminative framework

The central idea of discriminative framework is to model $P(\mathbf{Y}|\mathbf{X})$ directly, without modeling the joint probability. There are two immediate advantages to the discriminative framework: (1) no efforts are made to model the states $P(\mathbf{Y})$; (2) no unwarranted assumptions of independence are made to factorize the conditional distribution $P(\mathbf{X}|\mathbf{Y})$.

Modeling conditional distributions over complex structured objects has a long history, which arguably started in the field of speech and handwriting recognition (see [LBBH98, YH05] and references therein). The topic has seen a resurgence of interest in the machine learning and natural language processing communities recently [LMP01, Wal04] with the concept of Conditional Random Field (CRF). An extension of CRF for image segmentation called Discriminative Random Field (DRF) were introduced in [SM03]. It uses local discriminative models to capture the class associations at individual sites as well as the interactions in the neighboring sites on 2-D grid lattices.

In DRF, the conditional distribution is modeled as

$$p(\mathbf{Y}|\mathbf{X}) = \frac{1}{Z} exp(\sum_{i \in S} A_i(x_i, \mathbf{Y}) + \sum_{i \in S} \sum_{j \in \mathcal{N}_i} I_{ij}(x_i, x_j, \mathbf{Y}))$$

where A_i and I_{ij} are called *association potential* and *interaction potential* respectively.

3.1.3 Training a Discriminative Models

In this section, we give the derivation of a general class of learning algorithm for training a model that estimates $P(\mathbf{Y}|\mathbf{X}, \mathbf{W})$. We place ourselves in a more general setting than either CRFs or DRFs since we make minimal assumptions about the form of the conditional probability distribution, and its parametrization.

The energy function $E(\mathbf{W}, \mathbf{Y}, \mathbf{X})$ is parameterized by a parameter vector $\mathbf{W}$. Without loss of generality, we pose that the conditional distribution is the normalized exponential of an energy function (Gibbs distribution):

$$P(\mathbf{Y}|\mathbf{X}, \mathbf{W}) = \frac{e^{-\beta E(\mathbf{W},\mathbf{Y},\mathbf{X})}}{\sum_{\mathbf{Y}'} e^{-\beta E(\mathbf{W},\mathbf{Y}',\mathbf{X})}}. \tag{3.1}$$

where $\beta > 0$ is user-chosen constant, akin to an inverse temperature.

The learning problem consists of finding the value of $\mathbf{W}$ that maximizes the (conditional) likelihood of the training data under the model. This process is called Maximum Likelihood Estimation (MLE). Assuming that the training samples $(\mathbf{X}^i, \mathbf{Y}^i)$ are drawn independently, the likelihood of the training data is the product of individual likelihoods over samples $\prod_i P(\mathbf{Y}^i|\mathbf{X}^i, \mathbf{W})$.

Equivalently, the maximum likelihood estimate for $\mathbf{W}$ can be obtained by minimizing the following function which is the negative *log* of the likelihood:

$$f(\mathbf{W}) = -\frac{1}{m\beta}\sum_{i=1}^{m} logP(\mathbf{Y}^i|\mathbf{X}^i,\mathbf{W}) = \frac{1}{m}\sum_i\{E(\mathbf{W},\mathbf{Y}^i,\mathbf{X}^i)+\frac{1}{\beta}log\sum_{\mathbf{Y}} e^{-\beta E(\mathbf{W},\mathbf{Y},\mathbf{X}^i)}\}.$$

To minimize this loss function, we can use a gradient-based method. The gradient of the loss with respect to the parameter vector is:

$$\begin{aligned} f'(\mathbf{W}) &= \frac{1}{m}\sum_i\{\frac{\partial E}{\partial \mathbf{W}}(\mathbf{W},\mathbf{Y}^i,\mathbf{X}^i) + \frac{1}{\beta}\frac{\partial}{\partial \mathbf{W}}log\sum_{\mathbf{Y}} e^{-\beta E(\mathbf{W},\mathbf{Y},\mathbf{X}^i)}\} \\ &= \frac{1}{m}\sum_i\{\frac{\partial E}{\partial \mathbf{W}}(\mathbf{W},\mathbf{Y}^i,\mathbf{X}^i) + \frac{1}{\beta}\frac{-\beta\sum_{\mathbf{Y}} e^{-\beta E(\mathbf{W},\mathbf{Y},\mathbf{X}^i)}\frac{\partial E}{\partial \mathbf{W}}(\mathbf{W},\mathbf{Y},\mathbf{X}^i)}{\sum_{\mathbf{Y}'} e^{-\beta E(\mathbf{W},\mathbf{Y}',\mathbf{X}^i)}}\} \\ &= \frac{1}{m}\sum_i\{\frac{\partial E}{\partial \mathbf{W}}(\mathbf{W},\mathbf{Y}^i,\mathbf{X}^i) - \sum_{\mathbf{Y}}\frac{e^{-\beta E(\mathbf{W},\mathbf{Y},\mathbf{X}^i)}}{\sum_{\mathbf{Y}'} e^{-\beta E(\mathbf{W},\mathbf{Y}',\mathbf{X}^i)}}\frac{\partial E}{\partial \mathbf{W}}(\mathbf{W},\mathbf{Y},\mathbf{X}^i)\}. \end{aligned}$$

Plug in our definition 3.1 of the conditional probability:

$$f'(\mathbf{W}) = \frac{1}{m}\sum_i\{\frac{\partial E}{\partial \mathbf{W}}(\mathbf{W},\mathbf{Y}^i,\mathbf{X}^i) - \sum_{\mathbf{Y}} P(\mathbf{Y}|\mathbf{X}^i,\mathbf{W})\frac{\partial E}{\partial \mathbf{W}}(\mathbf{W},\mathbf{Y},\mathbf{X}^i)\}.$$

We introduce the empirical probability defined by the training data:

$$P^0(\mathbf{X},\mathbf{Y}) = \frac{1}{m}\sum_{i=1}^{m}\delta(\mathbf{X}-\mathbf{X}^i)\cdot\delta(\mathbf{Y}-\mathbf{Y}^i).$$

where $\delta(\cdot)$ is the Delta function. Now we see that $P^0(\mathbf{Y}|\mathbf{X}^i) = \delta(\mathbf{Y}-\mathbf{Y}^i)$ for every i.

Plug in the definition of the P^0 and P^∞, we can write:

$$\begin{aligned} f'(\mathbf{W}) &= \frac{1}{m}\sum_{i=1}^{m}\{P^0(\mathbf{Y}^i|\mathbf{X}^i)\frac{\partial E}{\partial \mathbf{W}}(\mathbf{W},\mathbf{Y}^i,\mathbf{X}^i) - \sum_{\mathbf{Y}} P(\mathbf{Y}|\mathbf{X}^i,\mathbf{W})\frac{\partial E}{\partial \mathbf{W}}(\mathbf{W},\mathbf{Y},\mathbf{X}^i)\} \\ &= \frac{1}{m}\sum_{i=1}^{m}\{\sum_{\mathbf{Y}} P^0(\mathbf{Y}|\mathbf{X}^i)\frac{\partial E}{\partial \mathbf{W}}(\mathbf{W},\mathbf{Y},\mathbf{X}^i) - \sum_{\mathbf{Y}} P(\mathbf{Y}|\mathbf{X}^i,\mathbf{W})\frac{\partial E}{\partial \mathbf{W}}(\mathbf{W},\mathbf{Y},\mathbf{X}^i)\}. \end{aligned}$$

We arrive at a very insightful formula.

$$f'(\mathbf{W}) = \sum_{i=1}^{m} \{ < \frac{\partial E}{\partial \mathbf{W}}(\mathbf{W}, \cdot, \mathbf{X}^i) >_{P^0(\mathbf{Y}|\mathbf{X})} - < \frac{\partial E}{\partial \mathbf{W}}(\mathbf{W}, \cdot, \mathbf{X}^i) >_{P^\infty(\mathbf{Y}|\mathbf{X})} \}.$$

i.e the parameter updates should be proportional to the difference of the expectation of $\frac{\partial E}{\partial \mathbf{W}}$ in two probabilities: The first one $P^0(\mathbf{Y}|\mathbf{X})$ is the empirical probability for the training data, which in our case consist of a delta function around the output for each training sample $\mathbf{Y}^i$. The second one $P^\infty(\mathbf{Y}|\mathbf{X})$ is the distribution given by out model.

Estimating the second term is often intractable, because it involves a sum (or an integral) over a all possible configurations of $\mathbf{Y}$. Many approaches to the problem use Markov Chain Monte Carlo methods to estimate the second term of the gradient. Hence, we call the distribution the *equilibrium distribution*, and use the notation P^∞.

More specifically, there are two general approaches to estimating $f'(\mathbf{W})$, which are the next topics in this chapter.

- Sample from P^∞ approximately, so we have an estimate of $< \frac{\partial E}{\partial \mathbf{W}} >_{P^\infty}$. A common way to do so is to construct a Markov chain and utilize Markov chain Monte Carlo methods (MCMC). One popular approach, when the space of $\mathbf{Y}$ is continuous, is Hybrid Monte Carlo methods.

- Replace the intractable distribution P^{∞} by some more amenable approximation. A recent proposal called *Contrastive Divergence*, consists simply running MCMC for a fixed number of steps, instead of running it until convergence. A key issue is that what we estimate is no longer the gradient of f. Therefore, there is no guarantee that the parameter updates will increase the likelihood.

3.2 Sampling Methods

This section presents various sampling methods that are useful to estimate the gradient of the negative log likelihood.

for the sake of completeness, we give a short introduction to Markov Chain methods at the end of this chapter.

One of the central goals of Monte Carlo (MC) method [G.S95, Mac03, R.M93] is to approximate the expected value of some function $h(\mathbf{Y})$ over a probability distribution $P(\mathbf{Y})$. We denote this expected value $< h(\mathbf{Y}) >_{P(\mathbf{Y})}$. Monte Carlo methods approximate this expectation by an average over a collection of well-chosen discrete samples $\sum_i h(\mathbf{Y}_i)P(\mathbf{Y}_i)$, where $\mathbf{Y}_1, \mathbf{Y}_2, ..., \mathbf{Y}_m$ is a sequence of samples drawn based on the knowledge of P. The unique fact of Monte Carlo approximation is that

The absolute error decreases as $\frac{1}{\sqrt{m}}$, independently of the problem size.

This error bound is valid without any specific knowledge of a particular problem. This property is extremely attractive for research of image processing, where very high dimensional problem space is common. For example, an 256×256 image is in 65536 dimensional space of pixels.

3.2.1 Markov Chain Monte Carlo

In high dimensional spaces, the high probability region is very small. (Such claim can be quantified using the concept of typical set and proved in a general setting, see [Mac03].) So it is wasteful to search for high probability region from scratch every time. It is more economic to explore the space in some systematic way so that the sampler will not leave the high probability region once it is inside. Note that this has the consequence that the samples are co-related instead of independent. But it does not matter since we only use the samples to estimate the expectation. Markov-chain Monte-Carlo (MCMC) utilizes a Markov chain to find the next state $\mathbf{Y}^{i+1}$ from current state $\mathbf{Y}^i$ via a transition matrix. The Hybrid Monte Carlo method moves the sampler along the *log*-probability increasing direction (this will be discussed

in 3.2.2)

We assume the sampling space is finite with all states enumerated as $\mathbf{Y}_1, \mathbf{Y}_2, ...$ If we are given an initial distribution $P^{(0)}$ and a *transition distribution* $T(\cdot, \cdot)$, we can define a sequence of distributions P^t by:

$$P^{(t+1)}(\mathbf{y}_i) = \sum_k T(\mathbf{y}_i, \mathbf{y}_k) P^{(t)}(\mathbf{y}_j), \text{ for all } t = 0, 1, 2..., \forall i, j.$$

The MCMC Algorithm

So far, we have introduced the necessary theoretical framework for MCMC algorithms. For P to be sampled, we should construct a ergodic Markov chain such that P will be the limit probability. A sampling from the limit distribution can be obtained from a sample from initial distribution $P^{(0)}$ (which can be arbitrary, based on the above theorem), then repeat transition for a large number of times. If we repeat n transitions and obtain y, then y is actually sampled from $P^{(n)}$. The inequality in the fundamental theorem gives us an estimate of error of y and a sample from true distribution P.

The practical problem is how to construct the Markov chain, or how to construct the transition distribution.

Metropolis-Hasting Algorithm

The Metropolis method [Mac03] is useful to get samples for P, where P is hard to sample directly, but an approximate Q is easy to sample. It can be a building block for the MCMC and HMC methods.

With current sample $\mathbf{Y}$, The Metropolis method can generate the next sample $\mathbf{Y}'$ from P by:

Sample $\mathbf{Y}*$ from $Q(\mathbf{Y}*;\mathbf{Y})$ and compute $a = \frac{P(\mathbf{Y}*)Q(\mathbf{Y};\mathbf{Y}*)}{P(\mathbf{Y})Q(\mathbf{Y}*;\mathbf{Y})}$.
If $a \geq 1$, we accept the new state: $\mathbf{Y}' = \mathbf{Y}*$. Otherwise, we accept with probability a.
If rejected, we set $\mathbf{Y}' = \mathbf{Y}$.

The efficiency of the Metropolis method depends on a, or, how close P and Q are. If Q is similar to P, a will be closed to 1 most of time, so we obtain a new state. Otherwise, we may need many runs of Metropolis method to obtain one new sample, which leads to inefficiencies.

If the conditional probability of each component is known, *Gibbs sampling* [Mac03] can be used to implement Metropolis-Hasting algorithm. Assume y is K-dimensional, Gibbs sampling is a scheme to generate $\mathbf{Y}'$ from old sample $\mathbf{Y}$ by the following K steps:

$$y_1' \sim P(y_1|y_2, y_3, y_4, ..., y_K)$$

$$y_2' \sim P(y_2|y_1', y_3, y_4, ..., y_K)$$

$$y_3' \sim P(y_3|y_1', y_2', y_4, ..., y_K)$$

$$...$$

$$y_K' \sim P(y_K|y_1', y_2', y_3', ..., y_{K-1}').$$

3.2.2 Hybrid Monte Carlo

The Hybrid Monte Carlo (HMC) method [Mac03, R.M93] is based on a physical analogy. It is sometimes called the dynamical method because of this analogy. The gradient of the potential energy of a physical system, with respect to its spatial coordinates, defines the *force* that acts to change the momentum hence the configuration of the system. When a physical system is in contact with a heat source, it is also influenced by the source in certain random fashion. These dynamical and stochastic effects together result in the system visiting states with a frequency given by its canonical distribution. Therefore, simulating the dynamics of such a physical system provides a way of sampling from the canonical distribution.

When Y is a continuous variable, the HMC method is often more efficient than simple Metropolis algorithms, mainly due to the fact that HMC avoids

the random walk behavior inherent in simple MCMC algorithm. We are free to construct an artificial physical system for our problem (e.g. image labeling) without any relation with a real physical system. We will continue to use some terminology from physical only to keep the analogy.

We need one extra assumption on P so that HMC can be applied. Namely, E should be differentiable with respect to $\mathbf{Y}$.

HMC starts with augmenting state variable $\mathbf{Y}$ with momentum variable $\mathbf{V}$. The Hamilton is defined as $H(\mathbf{Y}, \mathbf{V}) = E(\mathbf{Y}) + K(\mathbf{V})$, where $K(\mathbf{V})$ is the *kinetic energy* as $K(\mathbf{V}) = \frac{1}{2}\mathbf{V}^T\mathbf{V}$. We update the states by alternating two types of proposals. The first one randomizes $\mathbf{V}$ and keeps $\mathbf{Y}$ unchanged. The second proposal changes both $\mathbf{V}$ and $\mathbf{Y}$ by simulating the Hamilton system with the dynamics equations:

$$\begin{aligned} \frac{d\mathbf{Y}}{dt} &= \mathbf{V} \\ \frac{d\mathbf{V}}{dt} &= -\frac{\partial E}{\partial \mathbf{Y}} \end{aligned}$$

We must discretize the above differential equations for computer implementation. A popular scheme is called *leapfrog* discretization, which preserves the phase space volume and is also time reversible. The nice properties ensure high precision and low rejection rates, which improve the overall performance of HMC method. To be precise, the leapfrog scheme to advance the state from time t to $t + \Delta t$ is:

$$
\begin{aligned}
\hat{\mathbf{V}}(t+\frac{\Delta t}{2}) &= \hat{\mathbf{V}}(t) - \tfrac{\Delta t}{2}\tfrac{\partial E}{\partial \mathbf{Y}}(\hat{\mathbf{Y}}(t)), \\
\hat{\mathbf{Y}}(t+\Delta t) &= \hat{\mathbf{Y}}(t) + \Delta t \hat{\mathbf{V}}(t+\tfrac{\Delta t}{2}), \\
\hat{\mathbf{V}}(t+\Delta t) &= \hat{\mathbf{V}}(t+\tfrac{\Delta t}{2}) - \tfrac{\Delta t}{2}\tfrac{\partial E}{\partial \mathbf{Y}}(\hat{\mathbf{Y}}(t+\Delta t)).
\end{aligned}
$$

3.3 Contrastive Divergence Learning

Contrastive divergence learning [Hin02] is an *approximate* Maximum Likelihood (MLE) learning method. To understand the approximation of learning object, we need the concept of Kullback-Leibler (KL) divergence. KL divergence measures the difference of two probabilities on the same data

$$KL(P||Q) = \sum_{\mathbf{Y}} P(\mathbf{Y}) log \frac{P(\mathbf{Y})}{Q(\mathbf{Y})}.$$

It is easy to show that maximizing the likelihood (MLE) is equivalent to minimizing the KL divergence between the data distribution and the model distribution $KL(P^0||P^\infty)$. This can be easily verified by the following fact:

$$\frac{\partial}{\partial \mathbf{W}} KL(P^0||P^\infty) = < \frac{\partial E}{\partial \mathbf{W}} >_{P^0} - < \frac{\partial E}{\partial \mathbf{W}} >_{P^\infty}$$

which is equal to the gradient of the negative log likelihood loss.

CD learning (for a finite integer m) uses a different objective

$$CD_m = KL(P^0||P^\infty) - KL(P^m||P^\infty).$$

The distribution P^m is defined by running a Markov chain for m steps, starting from data distribution P^0.

We expand the formula for CD_m:

$$CD_m = \sum_{\mathbf{Y}} P^0 log P^0 + \sum_{\mathbf{Y}} (P^m - P^0) log P^\infty - \sum_{\mathbf{Y}} P^m log P^m.$$

Observing the first term is independent of W, we can calculate the derivative

$$\frac{\partial CD_m}{\partial \mathbf{W}} = < \frac{\partial E}{\partial \mathbf{W}} >_{P^0} - < \frac{\partial E}{\partial \mathbf{W}} >_{P^m} - \sum_{\mathbf{Y}} \frac{\partial P^m}{\partial \mathbf{W}} (log P^\infty - log P^m).$$

The third term is hard to evaluate, but we can safely ignore in practice. The weight update rule for CD learning therefore is

$$\Delta \mathbf{W} \propto < \frac{\partial E}{\partial \mathbf{W}} >_{P^0} - < \frac{\partial E}{\partial \mathbf{W}} >_{P^m} .$$

Obviously, if $m \to \infty$, we are back to MLE. However, the really interesting CD learning comes with small m. In fact, [Hin02] suggests to use $m = 1$. Along with reduced time to generate samples, samples from P^m tend out to have lower variation than samples from P^∞.

In appendix C, we present our un-published using contrastive divergence to train an EBM to do image denoising. We can clearly see how the energy profile changes.

3.4 Factor Graph

Factor graphs [KFL01] are a popular way to represent dependence and independence relationships between variables. Factor graphs subsume Bayesian nets and probabilistic graphical models. Our introduction below highlights the relations between factor graphs, graphical models, and energy-based models.

Graphical models [JGJS98] have witnessed rapid advances in the past decade. By definition, they are graphs associated with probabilistic semantics. The nodes in the graph represent variables while the (lack of) edges describe conditional (in-)dependences between variables. Visible nodes represent observations and the hidden nodes represent the unobserved causes or effects. Training a graphical model consists in assigning high probability to observations (i.e. observed configurations of visible variables); inference consists in estimating the posterior distribution over variables to be predicted given the value of observed variables. Depending on whether the edges are directional or not, graphical models can be classified into two categories.

(1) Directed graphical models. [1] In such models, an edge $y' \rightarrow y$ means that y' is a direct cause of y. The joint distribution over all the nodes

[1] We only consider the directed graphical models without loops. They are the most simple but most popular ones. If a directed graphical model contains loops, the formula 3.2 is not necessarily valid.

y is given as

$$P(\mathbf{Y}) = \Pi_i P(y_i|pa(y_i)). \tag{3.2}$$

where $pa(y_i)$ are the parents of the node y_i. In real world problems, it may not be easy or necessary to identify the "causal" relationships of all the nodes. The only situation in which we rely on causality is when we define the joint distribution via conditional distributions. Bayes rule establishes that $P(\mathbf{Y}|\mathbf{Y}')$ and $P(\mathbf{Y}'|\mathbf{Y})$ can be converted into one another. Conceptually, there is no reason to favor one over the other.

(2) Undirected graphical models. A *clique* is defined as a totally connected subset of nodes. Assume for any clique $\mathbf{Y}_c$ in the graph, we have a potential $\psi_c(\mathbf{Y}_c)$. The joint distribution is

$$P(\mathbf{Y}) = \frac{1}{Z}\Pi_c \psi_c(\mathbf{Y}_c).$$

where $Z = \sum_c \Pi_c \psi_c(\mathbf{Y}_c)$ is the normalization constant, called *the partition function*.

Directed graphical models can be converted into undirected ones by *moralizing*.

(3) Factor graphs. The graph is bi-partite. The nodes are categorized as *variable nodes* and *function nodes*. A "global" function of many variables factors into a product of "local" functions. Compared to directed and undirected graphical models, factor graphs decompose the interactions between

the variables more clearly. As result, a factor graph is more succinct to express models where many relationships between variables are *overlapping*.

In the undirected graphical models, the joint distribution is defined via products of un-normalized, local potentials. There is a deeper connection of undirected graphical models and factor graphics. The (loopy) belief propagation algorithm [YFW01, YFW02, Teh03], which operates on graphical models by "message passing", can be translated directly to the "sum-product" algorithm operating on factor graphs, since they ultimately express the same factorization.

Factor graphs can expressed in the *log*-probability (the "energy") space where the global function is a sum of local functions. The two expressions are equivalent and share same computational complexity. More specifically, the "sum-product" algorithm is replaced by "max-sum" algorithm, by litterally changing the words "sum"/"product" by "max"/"sum".

The energy-based model (EBM, more details in next chapter) also works in the "energy" space. EBM expresses many overlapping interactions between the variables directly. For example, our system uses an EBM to express many constraints on the same/overlapping neighborhood in an image. However, EBMs are very different from factor graphs. EBMs are non-probabilistic and do not involve any normalization. As consequence, we need a different learning paradigm to train EBMs.

It is interesting to see how the models will express the (in-)dependence for a very simple scenario, see figure 3.1, where the evolution from one model to another is also suggested. We consider a scenario with only three variables A, B and C. B depends on A and C depends on both A and B. Such dependency is explicitly expressed as directed edges in the directed graphical model. In contrast, it is not explicit in the undirected graphical model, where we have a clique of size three. A factor is associated to each edge in the factor graph, but no causal information available due to lack of direction of the edges.

In the EBM, the local potential functions are associated to each pair of nodes just as the factors in the factor graph, whose sum is the global potential function, no normalization involved. The edges in the EBM graphical representation indicate the data flow (or, the order of calculation), not dependency.

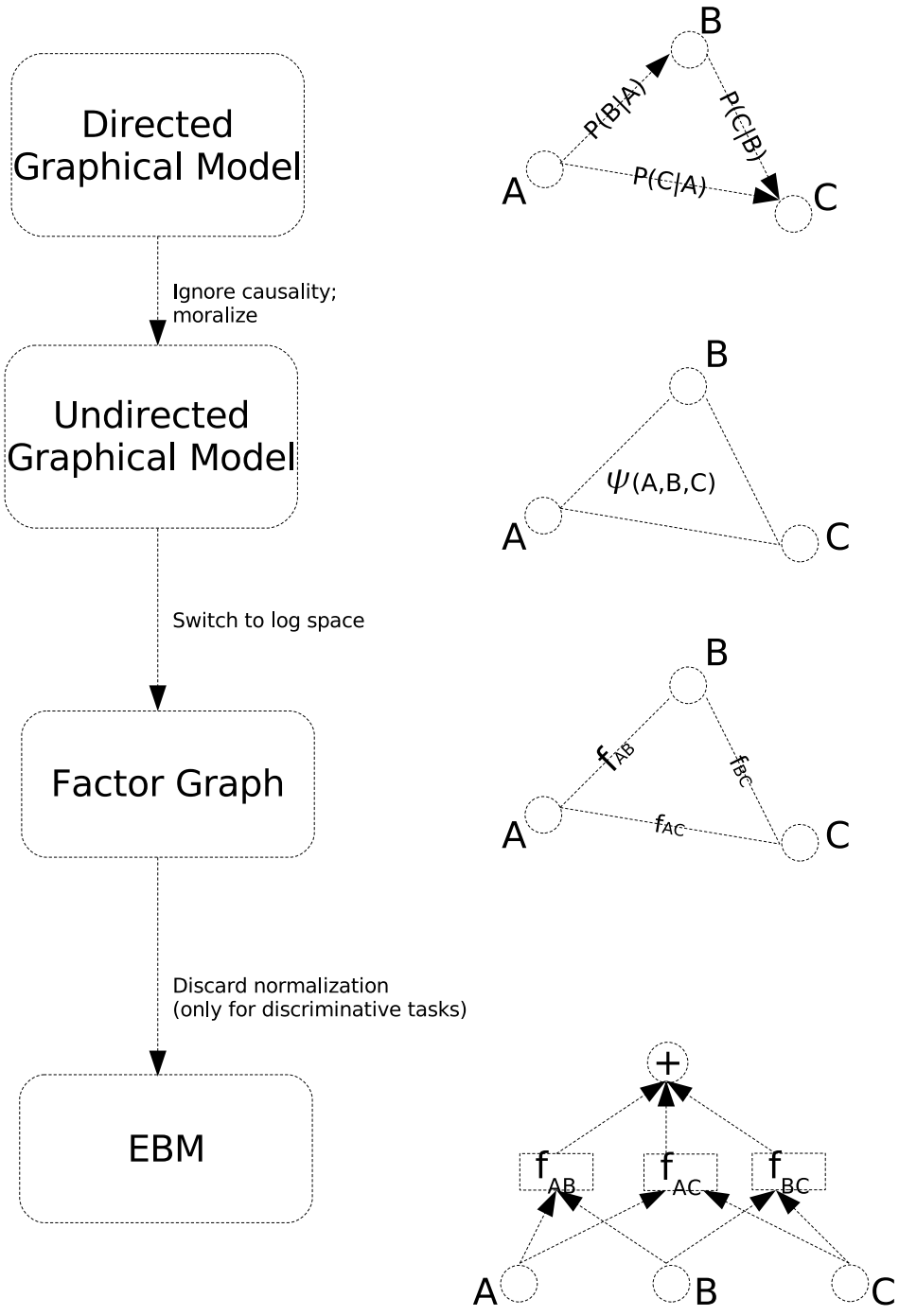

Figure 3.1: Comparison of several modeling methods on a simple, 3-variable scenario.

3.5 Summary

In this chapter, we have discussed a lot of models, including both probabilistic and non-probabilistic models. We do not pretend to provide an encyclopedic review of all related work. Instead, we try to select and organize our material to motivate the energy-based machine (EBM) approach, which we build our second module upon.

3.6 Appendix: A Mini Introduction to Markov Chain

In brief, a Markov Chain is a sequence of random variables $\mathbf{Y}_1, \mathbf{Y}_2, ...$, where the past is irrelevant to predict the future, given the knowledge of the present (Markov property). Formally, $P(\mathbf{Y}_{i+1}|\mathbf{Y}_i) = P(\mathbf{Y}_{i+1}|\mathbf{Y}_i, \mathbf{Y}_{i-1}, .., \mathbf{Y}_1)$. Certain Markov Chains can be described as finite state machine, where we can define the transition matrix T, $T(i, j) = P(\mathbf{Y}_k = i|\mathbf{Y}_{k-1} = j)$. A distribution $\pi(\mathbf{y})$ is called *invariant distribution* if

$$\pi(\mathbf{y}_i) = \sum_k T(\mathbf{y}_i, \mathbf{y}_k)\pi(\mathbf{y}_j), \forall i, j$$

A distribution $\pi(\mathbf{Y})$ is called *ergodic distribution* if

$$P^{(t)}(\mathbf{Y}) \to \pi(\mathbf{Y}), \text{ as } t \to \infty, \text{ for any } P^{(0)}(\mathbf{Y})$$

Note that it is quite strong condition for a distribution to be ergodic, because the above identity is required to hold for all $P^{(0)}$. In practice, there are some necessary conditions for ergodic chains easy to verify. For example, a Markov chain with reducible transition matrix cannot be ergodic. Also, if a Markov chain has a periodic set, it cannot be ergodic.

The fundamental theorem ([R.M93], chapter 3) gives a sufficient condition for ergodic chain:

Theorem: If a homogeneous Markov chain on a finite space with transition probabilities $T(\mathbf{Y}, \mathbf{Y}')$ has π as an invariant distribution and

$$\nu = min_{\mathbf{Y}} min_{\mathbf{Y}':\pi(\mathbf{Y}')>0} T(\mathbf{Y}, \mathbf{Y}')/\pi(\mathbf{Y}') > 0.$$

then the Markov chain is ergodic. Moreover, we have quantitative bounds for convergence speed of the distribution and the expections: A bound on the rate of convergence is given by

$$|\pi(\mathbf{Y}) - P^{(n)}(\mathbf{Y})| \leq (1-\nu)^n$$

For any real valued function f,

$$| < f >_{\pi} - < f >_{P^{(n)}} | \leq (1-\nu)^n max_{\mathbf{Y},\mathbf{Y}'} |a(\mathbf{Y}) - a(\mathbf{Y}')|.$$

The significance of the fundamental theorem is apparent in the following scenario. We are required to sample from $\pi(\mathbf{Y})$, which may be complicated.

What we can do in practice is to construct a Markov chain and sample from $P^{(n)}$ for certain n. The key issue is whether we know the samples from $P^{(n)}$ are good enough to approximate the samples from $\pi(\mathbf{Y})$. The fundamental theorem sheds on on this issue.

Chapter 4

EBM Module

4.1 Overview

The previous chapter reviews related work on modeling labels on images. The probabilistic models will inevitably require evaluate a partition function, the normalization term which is a sum over all possible input configurations. The computational inefficiency motivates the adoption of non-probabilistic methods. This chapter presents the Energy-Based Model (EBM) we implement.

EBM [TWOE03, YH05] associates a scalar *energy* to each configuration of the input variables. Making an inference with an EBM consists in searching for a configuration of the variables to be predicted that minimizes the energy. EBMs have considerable advantages over traditional probabilistic models: (1) there is no need to compute the partition functions that may be intractable;

(2) because there is no requirement for normalizability, the repertoire of possible model architectures that can be used is considerably richer than with probabilistic models.

Some of those local constraints are easy to formulate. For example, a nuclear membrane pixel must be connected to other nuclear membrane pixels, and must have a nucleus pixel on one side, and a cytoplasm pixel on the other side. Traditionally, the interactions terms between the variables in the model are encoded by hand. While some of the rules for our application could be encoded by hand, we chose to learn them from data using the Energy-Based Model framework.

Training an EBM consists in finding values of the trainable parameters that associate *low energies* to "desired" configurations of variables (e.g. observed on a training set), and *high energies* to "undesired" configurations. With properly normalized probabilistic models, increasing the likelihood of a "desired" configuration of variables will automatically decrease the likelihoods of other configurations. With EBMs, this is not the case: making the energy of desired configurations low may not necessarily make the energies of other configurations high. Therefore, one must be very careful when designing loss functions for EBMs. We must make sure that the loss function we pick will effectively drive our machine to approach the desired behavior.

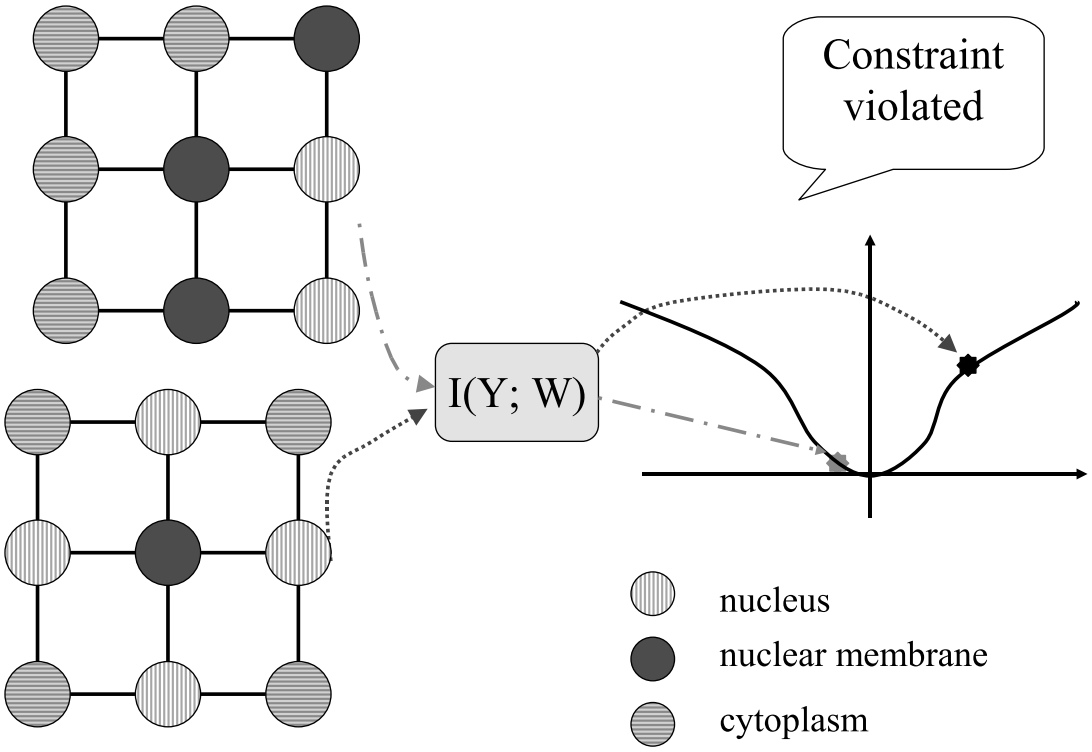

Figure 4.1: Modeling the local constraints. Top-left: consistent configuration, low energy assigned; bottom-left: non-consistent configuration, high energy assigned.

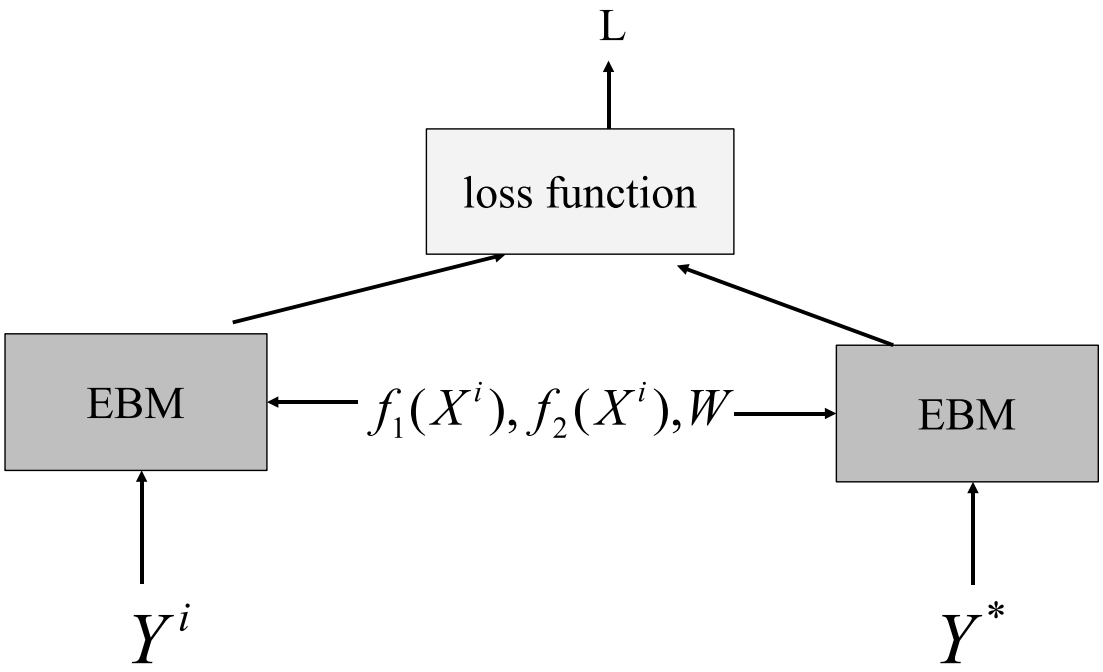

Figure 4.2: The loss function.

4.1.1 Loss Function

The energy function for EBM is a scalar $E(\mathbf{W}, Y, f_1(\mathbf{X}), f_2(\mathbf{X})))$ where $\mathbf{W}$ is the parameter vector to be learned, $\mathbf{Y}$ is the label image to be predicted using the EBM, $f_1(\mathbf{X})$ is the label image produced by the M1 convolutional network, and $f_2(\mathbf{X})$ the label image produced by the M2 convolutional network. Each of the variables $\mathbf{Y}, f_1(\mathbf{X}), f_2(\mathbf{X})$ are 3-D arrays of size $5 \times N_x \times N_y$, where N_x and N_y are the dimensions of the label images.

Before we write down any formula for the energy term and the loss function, we consider what they should look like.

First of all, the energy of desired configuration should be a stable minima. [YH05] captures this idea as:

Necessary Condition on the energy:

$$E(\mathbf{W}, \mathbf{Y}^i, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i)) < E(\mathbf{W}, \mathbf{Y}^*, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i)) - m,$$

where $\mathbf{Y}^* = argmin_{\mathbf{Y} \neq \mathbf{Y}^i} E(\mathbf{W}, \mathbf{Y}, \mathbf{X}^i)$.

We focus on a class of loss functions whose dependency on $\mathbf{X}^i$ only comes through E. [YH05] states the sufficient condition that minimizing such loss will drive the machine to find a solution that satisfies the above necessary condition for energy, if such a solution exists.

Sufficient condition on the loss function:

The minima of $Q_{[E_y]}(E(\mathbf{W}, \mathbf{Y}^i, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i)), E(\mathbf{W}, \mathbf{Y}^*, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i)))$ are in the half plane

$E(\mathbf{W}, \mathbf{Y}, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i)) < E(\mathbf{W}, \mathbf{Y}^i, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i)) + m$,

where the parameter $[E_y]$ contains the vector of energies for all values of $\mathbf{Y}$ except $\mathbf{Y}^i$ and $\mathbf{Y}^*$.

We use the following loss function, a version of generalized margin loss [YH05]. Similar loss functions have recently been used in the somewhat different contexts of face detection and pose estimation [OML05], pose and illumination-invariant generic object recognition [YH05], and face verification using trainable similarity metrics [CHL05].

$$L(\mathbf{W}, \mathbf{Y}^i, \mathbf{X}^i) = E(\mathbf{W}, \mathbf{Y}^i, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i)) + c_1 e^{-c_2 \min_{\mathbf{Y}, \mathbf{Y} \neq \mathbf{Y}^i} E(\mathbf{W}, \mathbf{Y}, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i))} \tag{4.1}$$

where c_1 and c_2 are user-specified positive constants, given training example $(\mathbf{X}^i, \mathbf{Y}^i)$, where $\mathbf{X}^i$ is an input image and $\mathbf{Y}^i$ a human-produced label image. The overall loss function is the average of the above function over the training set. The first term is the energy associated with the desired input configurations $(\mathbf{Y}^i, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i))$. Minimizing the loss will make this energy low on average. The second term is a monotonically decreasing function

of $\min_{\mathbf{Y},\mathbf{Y}\neq\mathbf{Y}^i} E(\mathbf{W},\mathbf{Y},f_1(\mathbf{X}^i),f_2(\mathbf{X}^i))$, which can be seen as the energy of the "best wrong answer", i.e. the lowest energy associated with a $\mathbf{Y}$ that is different from the desired answer $\mathbf{Y}^i$. Minimizing the loss will make this energy large. Minimizing this loss function makes the machine approach the desired behavior by making the energy of desired configurations low, and the energy of wrong configurations found by our inference algorithm high.

4.2 The Submodules of the EBM

Operating the EBM consists in running the input image through the M1 and M2 convolutional networks, and clamping the $f_1(\mathbf{X})$ and $f_2(\mathbf{X})$ inputs of the EBM to the values thereby produced. Then the $\mathbf{Y}$ input is initialized with the value $f_2(\mathbf{X})$, and an optimization algorithm is run to find a value of $\mathbf{Y}$ that locally minimizes $E(\mathbf{W},\mathbf{Y},f_1(\mathbf{X}),f_2(\mathbf{X})))$. The quantity at each pixel location of $\mathbf{Y}$, $f_1(\mathbf{X})$, and $f_2(\mathbf{X})$ is a discrete variable with 5 possible values: nucleus, nuclear membrane, cytoplasm, cell wall, external medium. A number of Markov-Chain Monte-Carlo (MCMC) methods were tested for minimizing the energy, including simulated annealing with Gibbs sampling. In the end, a simple "greedy" descent was used due to its nice property of efficiency with effectiveness. The distinct feature of our problem is that the labels are discrete, so continuous-variable based calculus cannot apply

straightforward. There are several workarounds, including (1) embed the discrete space in a continuous space; (2) use real-valued confidence scores from the convolutional layer to train EBM, instead of discrete-valued labels. In short, the sites are updated sequentially and set to the configuration that minimizes the energy, keeping the other sites constant.

The EBM is composed of two submodules or *factors*, as shown in figure 4.3. The overall energy is the sum of the energies produced by the two factors.

The first factor is the association module $A(\mathbf{Y}, f_1(\mathbf{X}), f_2(\mathbf{X}))$. The association module is fixed (it has no trainable parameter), and produces a high energy if $\mathbf{Y}$ gets very different from $f_1(\mathbf{X})$ or $f_2(\mathbf{X})$. The output energy of the association module is simply the average of those energies over all pixel locations. A particular way we choose to design an association module is to encode the function $A(\mathbf{Y}, f_1(\mathbf{X}), f_2(\mathbf{X}))$ in the form of a $5 \times 5 \times 5$ table which contains the energies associated with each possible combination of values of the variables $\mathbf{Y}$, $f_1(\mathbf{X})$, and $f_2(\mathbf{X})$ at any particular pixel location. The items in the table are statistics gathered from training data.

The idea is: if the M1 prediction is i, M2 prediction is j and the ground truth label is k, then $T(i, j, k)$ should be small. In this way, we should incur less resistance to correct the label to k from M1/M2 predictions. In fact,

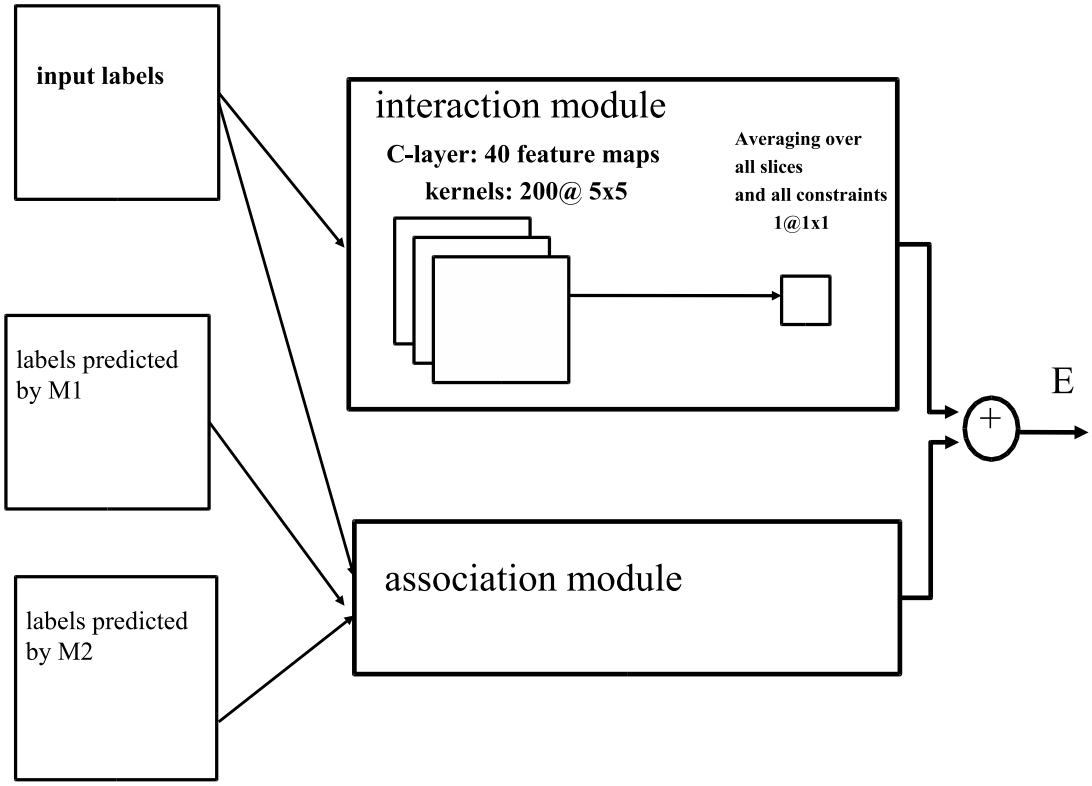

Figure 4.3: The architecture the Energy-Based Model. The image marked "input labeling" is the variable to be predicted by the EBM. The first layer of the interaction module is a convolutional layer with 40 feature maps and 5×5 kernels operating on the 5 feature maps from the output label image. The second layer simply computes the average value of the first layer.

we use a $5 \times 5 \times 5$ table C to collect the counts how M1, M2 and true labels are distributed on the training data. Then we build the table T by $T(i,j,k) = -logC(i,j,k)$, with the rationale that large value $C(i,j,k)$ indicates the true label is very likely k if M1 predicts i and M2 predicts j, so we should assign a low penalty (to correct the label from i to k). Finally, we scale and shift the values in T to meet some technical criteria: (1) the magnitude of values in table T should be smaller than the magnitude of fluctuation of output of interaction submodule. Otherwise, the effect of EBM will be dominantly comes from the effect of the association module, which is undesirable since T is still built from pixel-wise information without neighborhood information. (2) $T(i,i,i) = 0$, which is the interpretation that there should be no penalty if M1 and M2 both predict correct labels.

The second factor is the *interaction module* $I(\mathbf{W}, \mathbf{Y})$. The interaction module implements the local consistency constraints and is trained from data. The first layer of the interaction module is a convolutional layer with 40 feature maps and 5×5 kernels that operate on the 5 feature maps of $\mathbf{Y}$. The non-linear activation function of the units is of the form $g(u) = \frac{u^2}{1+u^2}$. This function and its derivatives are shown in figure 4.4. The idea behind this activation function is that each unit implements a *linear constraint* on the neighborhood of variables from which it takes inputs [TWOE03]. When

the input vector is near orthogonal to the weight vector, the output is near zero, indicating that the constraint is satisfied. When the input vector has a non-zero projection on the weight vector, the output is non zero. If the constraint is strongly violated, the output will be near 1.0 (the asymptote). The saturating asymptote ensures that only a "nominal price" will be paid (in terms of energy) for violating a constraint [TWOE03]. The total output energy of the interaction module is the average of the outputs of all the 40 feature maps over all positions.

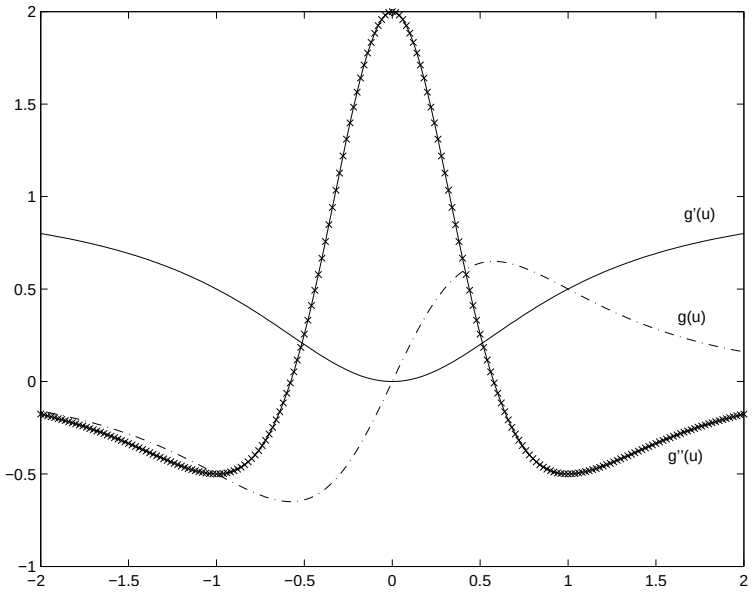

Figure 4.4: The activation function $g(u) = 1 - \frac{1}{1+u^2}$ used in the EBM feature maps.

4.3 The Approximate Coordinate Descent Algorithm

The straightforward method to minimize the loss function

$$L(\mathbf{W}, \mathbf{Y}^i, \mathbf{X}^i) = E(\mathbf{W}, \mathbf{Y}^i, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i)) + c_1 e^{-c_2 \min_{\mathbf{Y}, \mathbf{Y} \neq \mathbf{Y}^i} E(\mathbf{W}, \mathbf{Y}, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i))}$$

is via gradient descent:

$$\mathbf{W}' \leftarrow \mathbf{W} - \eta(\quad \frac{\partial E(\mathbf{W}, \mathbf{Y}^i, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i))}{\partial \mathbf{W}} \\ -c_1 c_2 e^{-c_2 E(\mathbf{W}, \mathbf{Y}^*, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i))} \quad \frac{\partial E(\mathbf{W}, \mathbf{Y}^*, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i))}{\partial \mathbf{W}}).$$

where $\mathbf{Y}^* = argmin_{\mathbf{Y}, \mathbf{Y} \neq \mathbf{Y}^i} E(\mathbf{W}, y, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i))$. Global minima is intractable to search in practice, so we will replace it by a local minima $\mathbf{Y}^* = argmin_{\mathbf{Y}, \mathbf{Y} \text{ near } \mathbf{Y}^i} E(\mathbf{W}, \mathbf{Y}, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i))$.

The inference (from state $\mathbf{Y}[0]$) is to find the minima:

$$\mathbf{Y}^* = argmin_{\mathbf{Y}, \mathbf{Y} \neq \mathbf{Y}[0]} E(\mathbf{W}, \mathbf{Y}, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i)).$$

In practice, we iterate the following local search, starting from $\mathbf{Y}[0] = f_1(\mathbf{X}^i)$:

$$\mathbf{Y}[t+1] = argmin_{\mathbf{Y}, \mathbf{Y} \text{ near } \mathbf{Y}[t]} E(\mathbf{W}, \mathbf{Y}, f_1(\mathbf{X}^i), f_2(\mathbf{X}^i)).$$

When t is large enough, we end the iteration and set $\mathbf{Y}^* = \mathbf{Y}[t]$.

We now give the detailed strategy of searching the local minima of E *near* some $\mathbf{Y}$, which is the core routine for both training and inference. If

$\mathbf{Y}$ is a continue variable, the local minima can be defined as minima in a neighborhood and gradient descent gives us an effective iterative method to approach the local minima. But in our case, $\mathbf{Y}$ is a discrete variable so that gradient does not make sense. Exhaustive search can find the exact minima in the discrete domain, but the time cost grows exponentially so practically it can only work for very low dimensions. To meet the challenge, we design a method to approximate the minima via a greedy algorithm of optimizing one coordinate at a time. So the time cost will be *linear* with respect to the dimension of the problem. We work on label images generally several hundred pixels in each dimension, so the label image is about 0.1 to 1.0 million dimension in the pixel space. The linear cost still proves to be too expensive in such high dimensional space. So we make a further approximation to significantly reduce the dimension of space, hence make our local search algorithm reasonably fast.

The key observation for speed up is that

> If $f_1(\mathbf{X}^i)[j,k] = f_2(\mathbf{X}^i)[j,k]$, then $f(\mathbf{X}^i)[j,k] = f_1(\mathbf{X}^i)[j,k]$, for any site (j,k) on the image, holds for both training and test data *approximately*.

In fact, the violation was below 1% for training data, below 5% for test data.

Since our goal is to correct wrong labels, we should focus on the sites where the labels are wrong. With above observation, we can only focus on the sites where M1 and M2 give different predictions. Empirical study confirms that M1 and M2 agree on more than 90% sites, so we effectively reduce the dimension of our search domain to sub-10% and it means a $10X+$ speedup. In our implementation, we randomize the order of the coordinates being searched.

4.4 Empirical Study

The training sets and test sets were the same as for the convolutional network module training. Namely, we took the same 10 movies used in convolutional net training, used the M1 and M2 predictions on them as input for EBM training. We ran EBM inference on M1 predictions of both the 10 training movies and the other 5 test movies. We trained EBM for 100 iterations, which took about 30 hours on a 2.0 GHz Opteron workstation. The inference was pretty efficient. For each label image, we ran the inference for 500 iterations, which only took about one minute.

See figure 4.5, 4.6. The method does a good job at eliminating isolated

points that were erroneously labeled. The nuclei are clearly identifiable in the resulting images. However, the method is a bit overzealous in cleaning up cell wall pixels. Several legitimate cell wall pixels that were correctly identified by the convolutional net were eliminated by the EBM. Our EBM uses kernel size of 5×5 so it is not designed to correct large region of wrong labels.

4.5 Summary

In this chapter, we present the detailed design the energy-based module, for the task of labeling clean-up. We start with some discussion on design of the energy function and the loss function. Next we present the training and inference algorithms and elaborate our efforts on effective local minimization in discrete domain. We conclude this chapter by the empirical study results to demonstrate the effectiveness of our design.

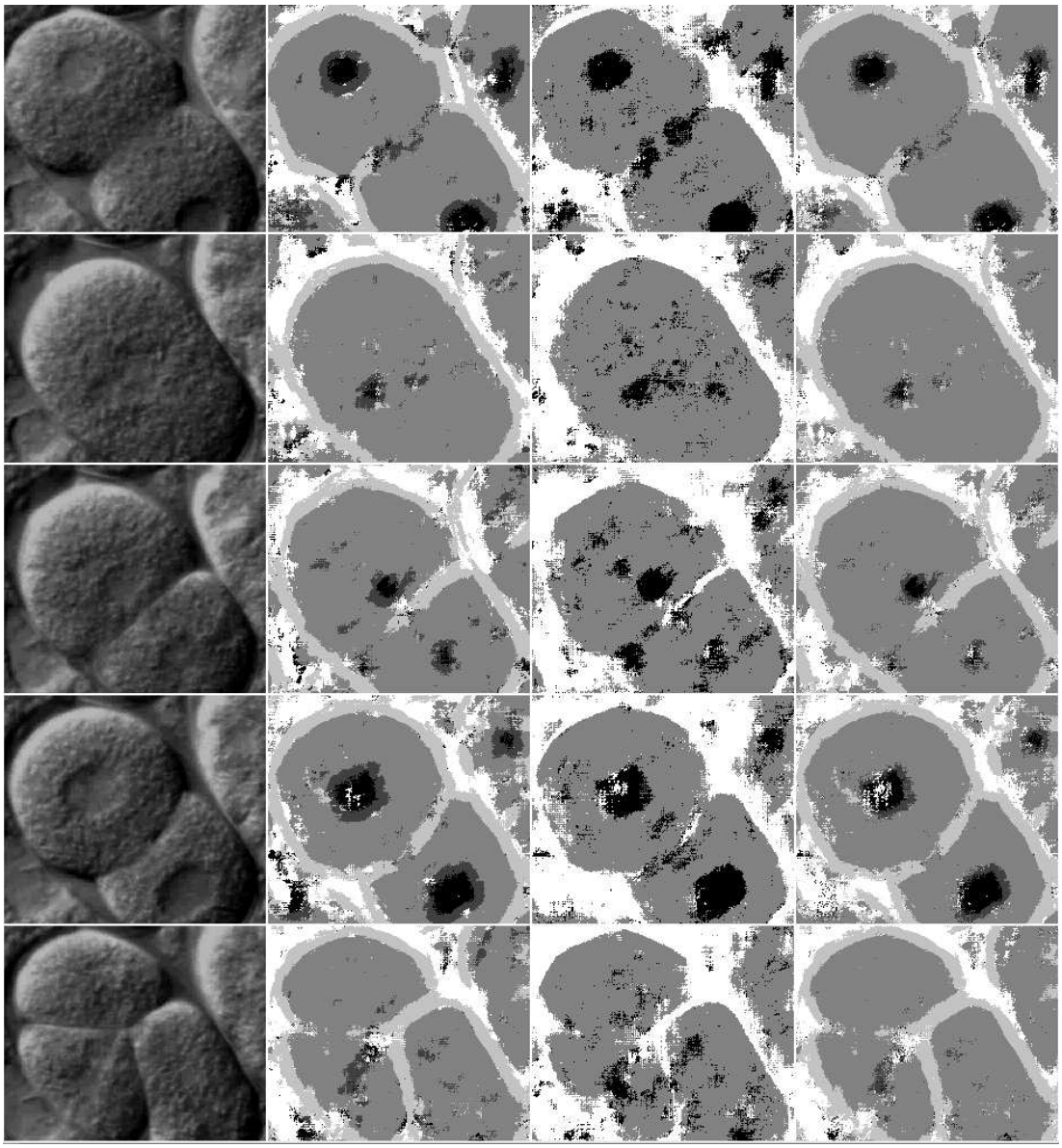

Figure 4.5: Results of EBM-based clean-up of label images on 5 training images. From left to right: input image; output of M2 convolutional network; output of M1 convolutional network; cleaned-up image by EBM.

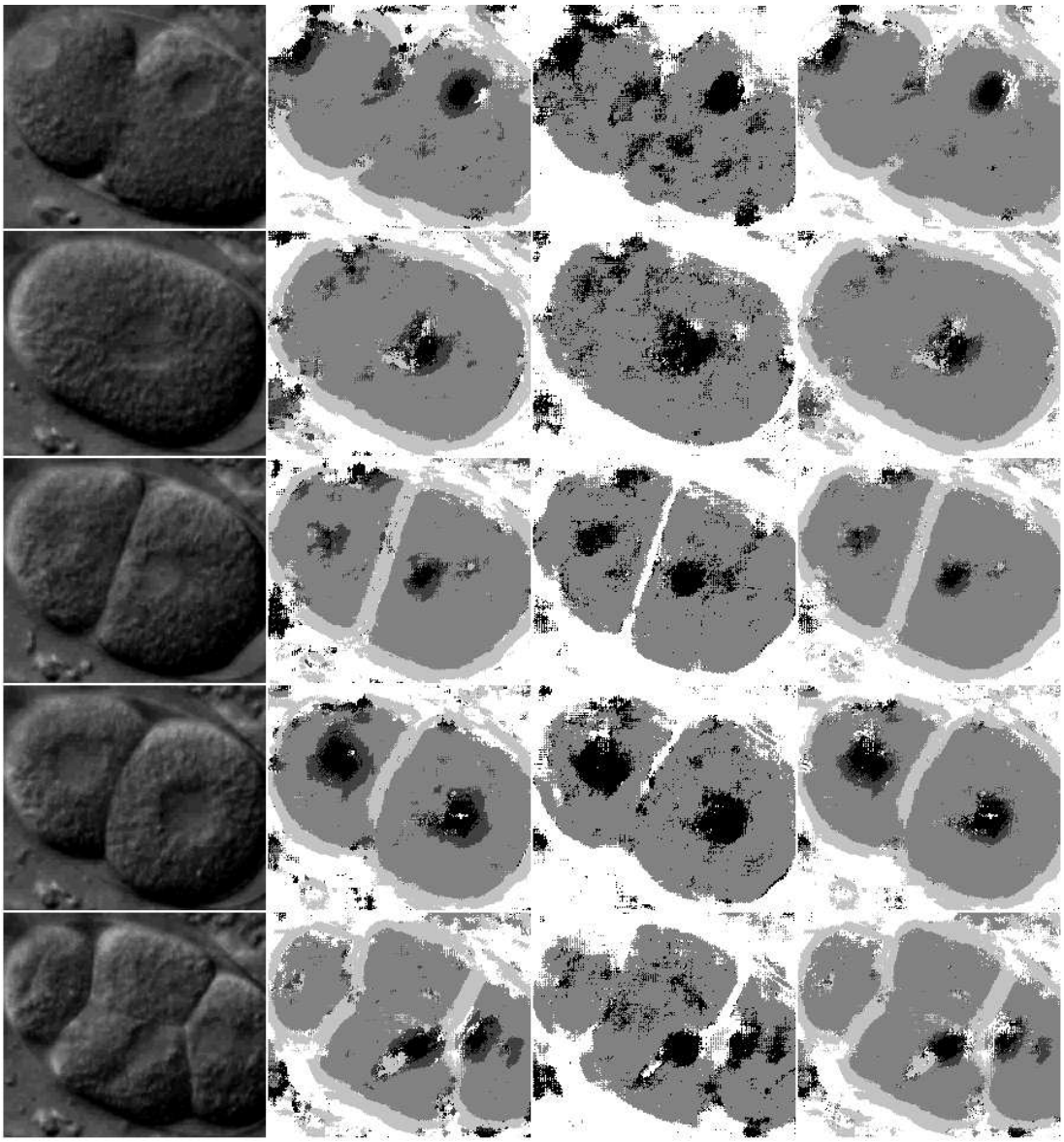

Figure 4.6: Results of EBM-based clean-up of label images on 5 testing images. The format is the same as in figure 4.5.

Chapter 5

Elastic Fitting Module

5.1 Introduction

The previous chapters discussed image analysis techniques to accurately segment images into cellular elements such as cell nucleus, nuclear membrane, etc. Further processing steps are needed to transform the *pixel-wise information* into the more *global characteristics* that are relevant to automatic phenotyping. This chapter first discusses various methods we considered for extracting information at such higher levels of abstraction and our specific implementation based on one approach. Then, we represent the design of our elastic fitting module.

5.2 Overview of Deformable Models for Template Matching

When experts visually inspect the images, they observe the emerging organization of the multi-cellular organism. This information is often expressed by drawing a sketch of the cell representing the relative shapes and positions of the cells and their nuclei. When inspecting a movie, sequences of specific events occurring over multiple frames are identified. Events can be global (for instance, which cells divide and in what order) or very specific (for instance, a visible feature that appears between two specific cells at a specific time during the embryo development).

One possible approach consists in collecting sketches representing various normal and abnormal stages of organism development, and using them as a dictionary of deformable templates that can be aligned and matched with each frame of the movies. Identifying the stage of development comes down to finding the deformable template in the dictionary that best matches the image under consideration. By aligning and fitting the template to the images, we can extract the relative shapes and positions of the cells, the orientation of the organism, and the identity of each individual cell. Deformable templates also enforce global constraints that are not easily implemented by local analysis. For example, the nucleus must lie roughly in the center of the

cell, the cell boundary must be a closed curve, etc.

Deformable template models have been used successfully in many problems, such as medical image processing [KWT87] and object matching in video sequences [AZL96]. For a survey of active contour methods, see [MT96].

In principle, there are two different methods for matching label images to deformable templates. The first method uses "sparse" elastic templates that are matched to the label images by minimizing an energy function using an efficient method reminiscent of the Expectation-Minimization algorithm (EM), see [NDL+05]. The second method uses "dense" elastic templates that are matched to the label images using an algorithm reminiscent of Kohonen's Self-Organizing Map algorithm (SOM, [Koh84]). Such dense templates may contain finer levels of global information such as the precise position of the cell boundaries. Fitting such models with the EM algorithm is prohibitively expensive, and subject to local minima. Xu and Prince [XP97] claim that complex contours may be identified more efficiently using dynamic algorithms that do not derive from the optimization of an energy function. Interesting active contour algorithms [AM93] are derived from the SOM method.

Figure 5.1 shows preliminary results obtained with a dense model using Colored SOMs. Each deformable template is specified by assigning labels to the nodes of a regular lattice [VKO97]. The lattice is then aligned with the label images associated with each frame using a variant of Kohonen's SOM

algorithm. Each iteration of the alignment algorithm picks a random image pixel and locates the closest lattice node with the same label. This node is then moved towards the location of the image pixel. Neighboring nodes in the lattice are also moved in the same direction with an amplitude that depends on their lattice distance to the node being considered. Step sizes and neighborhood sizes are decreased slightly after each iteration.

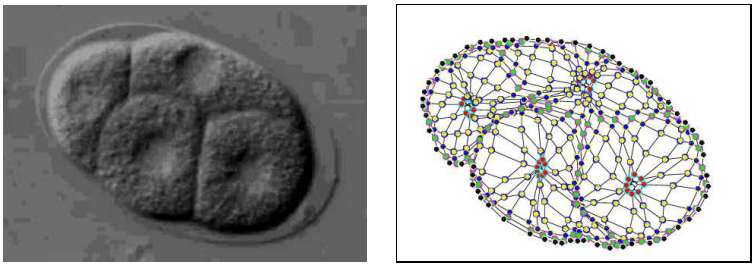

Figure 5.1: Matching deformable templates with Colored Self-Organizing Maps. Each deformable template is specified by coloring a regular lattice of nodes. The lattice is then aligned with the cell component labels derived from the image.

5.3 The Elastic Model

The dense model we discussed in the previous section does not incorporate the a priori knowledge about the global structure. As a consequence, it is not straightforward to interpret the fitting results to obtain useful knowledge of the objects of interest. On the other hand, the sparse model, by definition,

is not capable of fitting the curved contours of objects accurately.

The model we implemented, which we refer to as the "elastic model" from now on, borrows ideas from both dense and sparse models, to benefit from both without suffering their drawbacks. In our model, a template is composed of several sub-templates, in one-to-one correspondence with the objects under observation. For example, if there are two cells in the image and one of them contains a nucleus, we will have three objects and the template will have three corresponding sub-templates. Each sub-template is dense and is constructed by refining a sparse sub-skeleton. We refer to the collection of all sub-skeletons from all sub-templates as the skeleton for a template. The overall construction can be described in a more intuitive way: the skeleton describes the *topological* information of the model and each dense sub-template allows rich *geometric* information to be stored.

- **Sparse sub-skeleton:** a sub-skeleton is very similar to the template we built in [NDL+05], which defines the nodes as well as the elastic links among them to avoid collapse.

- **The refinement:** a sub-template is obtained by refinements to the skeleton. More nodes are added using B-spline subdivision scheme, if we view the nodes in the skeleton as the control points of a closed curve. More links are added to associate the new nodes to the skeleton and

resist self-collapse when fitting. See figure 5.2 for a complete list of all the templates we build.

5.3.1 Meta-templates

After fitting, a template will carry the information of the underlying label image. In our model, all templates are obtained from a few *meta-templates* by geometric transformation.

We use the special term *meta-* to indicate that they exist before any fitting process. We provide 5 meta-templates in the current implementation, which are built based on our knowledge of possible different phenotyping phases. See figure 5.2 for all the meta-templates. We can find that the meta-templates are like sketches of the correspondent images, but all metric information ignored (size, orientation). The metric information will be recovered in the fitting process. Figure 5.3 illustrates the procedure for building a meta-template from scratch, assuming we create a meta-template for the phenotype stage where there is one cell (C1) and one nucleus (K1).

5.3.2 Template Fitting

Matching images to the deformable templates can be achieved using the Expectation-Maximization algorithm (EM).

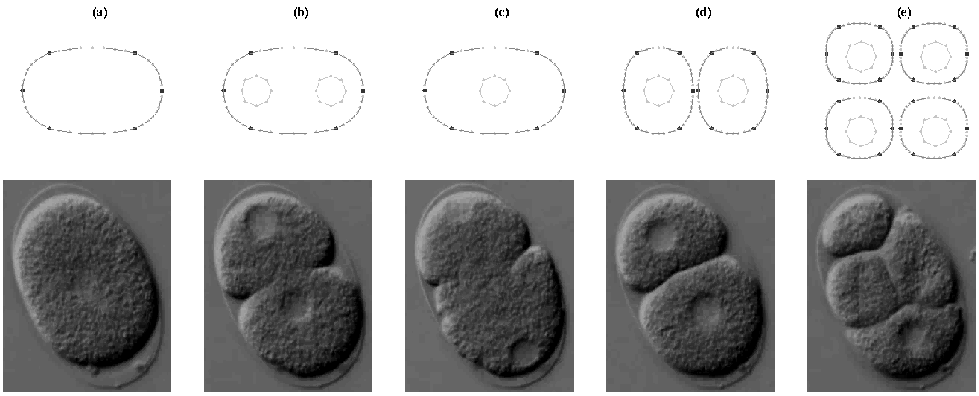

Figure 5.2: Meta-templates: (a) Fertilization has just occurred. (b) The maternal pronucleus migrates to the posterior area and a pseudo-cleavage furrow forms. (c) The pronuclei fuse. (d) The cell divides unequally to produce two cells. (e) The two cells further split into four cells.

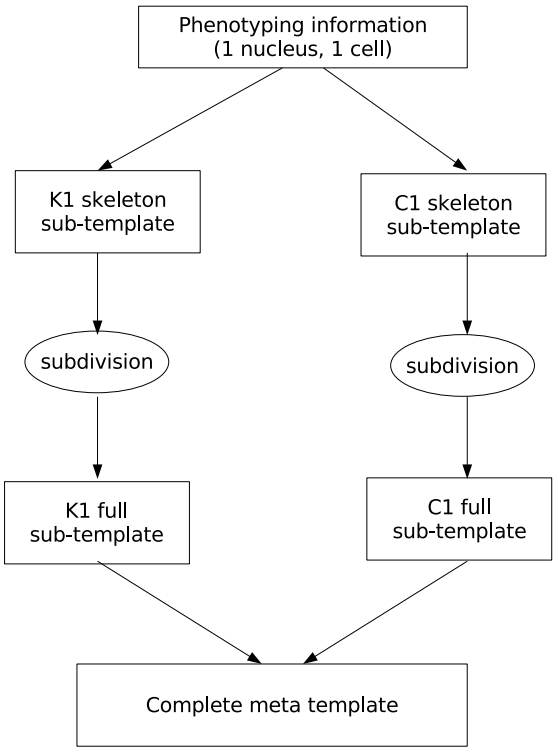

Figure 5.3: Constructing a meta-template using phenotype information.

Previous work [GCR91] has applied spline-based models to hand-written character recognition, using deformable splines whose control point positions were optimized with the EM. Each spline was seen as the mean of a probabilistic Gaussian model that could generate the "ink" of a character. Similarly, [BL94] proposed a normalization method for handwritten words that used EM to fit quadratic lines to key points on the trajectory of a pen writing a word.

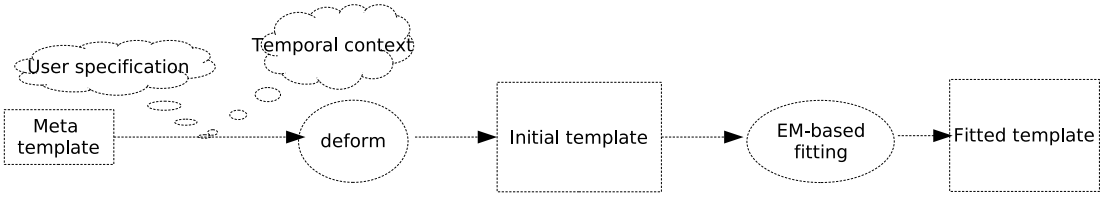

Figure 5.4: Transforming a meta-template and fitting it to a label image

However, an EM-like algorithm alone is not sufficient to provide robust fitting, and is prone to becoming trapped into local minima. Also, the label images from EBM inference cannot be assumed to be very high quality. Our solution is a semi-automatic template fitting scheme, which is shown in figure 5.4.

Our fitting algorithm is composed of two components. The core component of the algorithm is still elastic fitting based on EM. The other component of our algorithm is setting up good starting location. As we explained before, starting EM from arbitrary location can be disastrous. Good starting

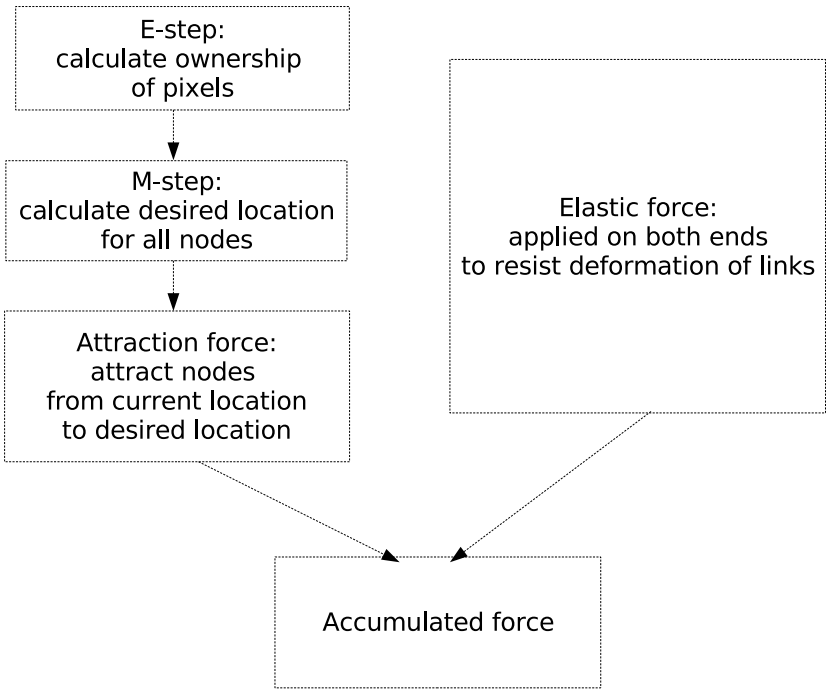

Figure 5.5: Calculating the forces for template deformation

location can be obtained by using certain prior knowledge, such as (1) expert specified initial transformation of a meta-template; (2) history information of fitting a previous frame in the same movie (temporal coherence).

EM is a frequently used method for data clustering and it alternates between performing an E-step (expectation) and a M-step (maximization). Our fitting problem can be phrased as finding "good" (the most "fitted") locations for the template nodes, given the labels for every pixel. This is done by alternating the following two steps. The first step computes the MLE estimates of the parameters in the system. The second step calculates the "desired" location of the nodes. Unlike plain EM algorithm, we do not update the location of the nodes to be exactly their "desired" locations. Instead, we design forces dragging all the nodes to the desired locations, and also introduce elastic forces to resist to any change of distance among the nodes. Figure 5.5 explains how the forces are calculated. We update the locations of the nodes by numerically simulating the system driven by the forces.

5.3.3 An Application: Iterative Labeling

Elastic fitting module lays the foundation for global analysis. It can find itself useself in other applications, for example, iterative labeling of large

image set. It can be time consuming and tedious to label a large image set by hand, even with the help of *CellSmart*. In contrast, the *iterative* approach can requires us label a few images and tweak other image labels. It works as below:

- Label a small subset of the images by hand using the *CellSmart* GUI front-end. Train the whole system with the subset of images and their labels.

- Use the system trained on the subset to perform inference for all the available data. *TemplateSmart* will create ROIs for *all the data*. We are now ready to train the system again, but with all the data.

Since our system is first trained with a small dataset, the generated ROIs are only approximate and adjustment by hand may be needed. However, adjustment by hand should cost less time than labeling from scratch.

5.4 Summary

In this chapter, we explain the design of our elastic model, which is motivated from both previous dense models and previous sparse models. We also give an application of the elastic model.

Chapter 6

Conclusion

6.1 Contributions

This thesis describes an end-to-end, learning-based system for automatic *C. elegans* embryo phenotyping. Our contribution is two-fold. First, we implement the system as described and measure the performance using a large dataset created from genetic research. The system we implement is fully functional and achieves satisfactory and robust results. Since the system is highly modular, it is extensible and ready to be incorporated into a more sophisticated phenotyping system. Second, we explore efficient, robust algorithms for learning/inference in energy-based machines. We identify the challenges in training an EBM in discrete, high-dimensional space. We propose the concept of slightly-perturbed labels and use them as guides in our

approximate coordinate descent learning algorithm.

6.2 Future Work

Building a fully automatic phenotyping system is the grand goal for future research. To this end, there are several possible enhancements which can be made to our system. We discuss two areas for future enhancements below, with suggested approaches to make the enhancements.

6.2.1 Encoding Temporal Consistency

Building a fully automatic phenotyping system is the grand goal for future research. To this end, there are several possible enhancements which can be made to our system. Because the configuration of the embryo changes slowly from frame to frame, it should be possible to improve the reliability of the labeling system by processing several successive frames simultaneously. The system described in this thesis could easily be modified so that a window of a few successive frames could be fed to the convolutional network. Similarly, the EBM could take multiple successive frames into account and encode temporal as well as spatial consistency.

Building such a system will require modeling the sequential aspect of the data. One technique being pursued involves the use of Hidden Markov Models

(HMM). Each known development scenario (normal or abnormal) can be associated with an HMM whose states represent the various stages of embryo development. The emission probability model for each state is a mixture model whose components are the deformable templates. Each deformable template can be seen as a probability density model whose log-likelihood is proportional the fitting energy of the deformable model. Classifying a movie into one of the scenarios simply consists in finding the HMM that maximizes the likelihood of the observed data. This can be performed with one of the standard methods for HMM inference (the Viterbi algorithm, or the forward algorithm).

6.2.2 Increasing System Efficiency

Machine learning, in addition to its statistical and theoretical aspects, must also be concerned with efficient algorithms and practical implementations. In our system, training is very computationally intensive. Some may argue that though training is the most time consuming part, it is irrelevant since the machine is trained off-line. However, efficiency issues can be critical if we look ahead to: (1) meeting the demand from functional genetic study for a *on-line* phenotyping system; (2) embedding our current system within a large-scale

system where it will be invoked multiple times, or fed with increased data volume. There are several possible approaches, as suggested below.

Search for a better learning algorithm. Let us analyze the two most significant training performance bottlenecks. For the first module, we currently feed a huge number of windows to the learner. We do this for two reasons: (1) to equalize for various categories; (2) to let the learner see varieties in the input characteristics. However, considering the capability of the kernels (only several thousand free parameters), we should be able to achieve the same goal with reduced number of inputs.

The EBM module requires training in the discrete domain. We provide our approximate coordinate descent algorithm as a solution. Many off-the-shelf algorithms are valid only for training in the continuous domain, which cannot be directly applied to our problem. Therefore our system performance can be boosted if there is breakthrough.

Obviously, any breakthrough in this direction will boost the performance of our system. Another approach would be to convert our EBM training problem to the continuous domain, perhaps using confidence scores instead of discrete labels.

Improve the performance of computing by code parallelization. This is essentially a black box approach, because details of specific algorithms do not matter. State-of-art computer systems are parallel systems (multi-

core, multi-CPU) that support high performance computing. Much machine learning code, especially for "neuron operations", is inherently parallelizable. In particular, the most frequent operation in our system is to apply the same kernel to multiple inputs, which can be done simultaneously without much coordinating overhead. It will be interesting to develop parallel machine learning libraries, esp. in LUSH. Such libraries will not only be beneficial to our system, but for learning research at large.

Appendix A

CellSmart

Training our system requires the knowledge of true labels for the pixels of the training images. The convolutional network uses the labels as targets for supervised learning, while the EBM uses the labels as "ground truth".

We designed and implemented a labeling tool to mark the cell wall/nucleus wall on the embryo images. Labels for all pixels in the training images can be obtained by the following steps. First, ROIs are labeled by hand and stored with the help of the labeling tool. Then Bresenham's line scanning algorithm computes all the pixels which are labeled as cell wall/nucleus wall. Finally, flooding algorithm is performed to determine the labels of all other pixels, which can either nucleus, cytoplasm or exterior area.

In principle, any image processing software with free drawing capability can be used, for example, the popular Java-based *ImageJ* [Ima], which is

open-source and cross-platform.

However, a major drawback of general-purpose drawing software is that it is not designed to be efficient for our labeling purpose. The large number of embryo images to be labeled imposed an efficiency requirement that warrants our efforts to design and implement a custom tool, *CellSmart*, along with a GUI front-end called *CellSmartGUI*. Both the tool and the front-end are implemented in LUSH [lus].

CellSmart can read an image from disk, in any format supported by LUSH, and store the labeled output in ROI file(s). Cell wall/nucleus wall is approximately polygonal and we record the position of all the polyon vertices in the ROI file. One image can correspond to multiple ROIs, hence correspond to multiple ROI files, if there are more than one cell/nucleus in the image.

CellSmartGUI is designed for efficient labeling. Since our images are sequential images, usually decoded from a single movie, we provide buttons to quickly switch to the next/previous images in a sequence of images, without specifying the full file path. In the early development of the embryo, only up to four cells and four nuclei can appear in an image. Therefore we provide four buttons for the cells ($C1 - C4$) and four for the nuclei ($K1 - K4$). In this way, the user can quickly access the predefined ROIs and label images much more quickly than with a general-purpose drawing tool. See figure A.1

for a screenshot.

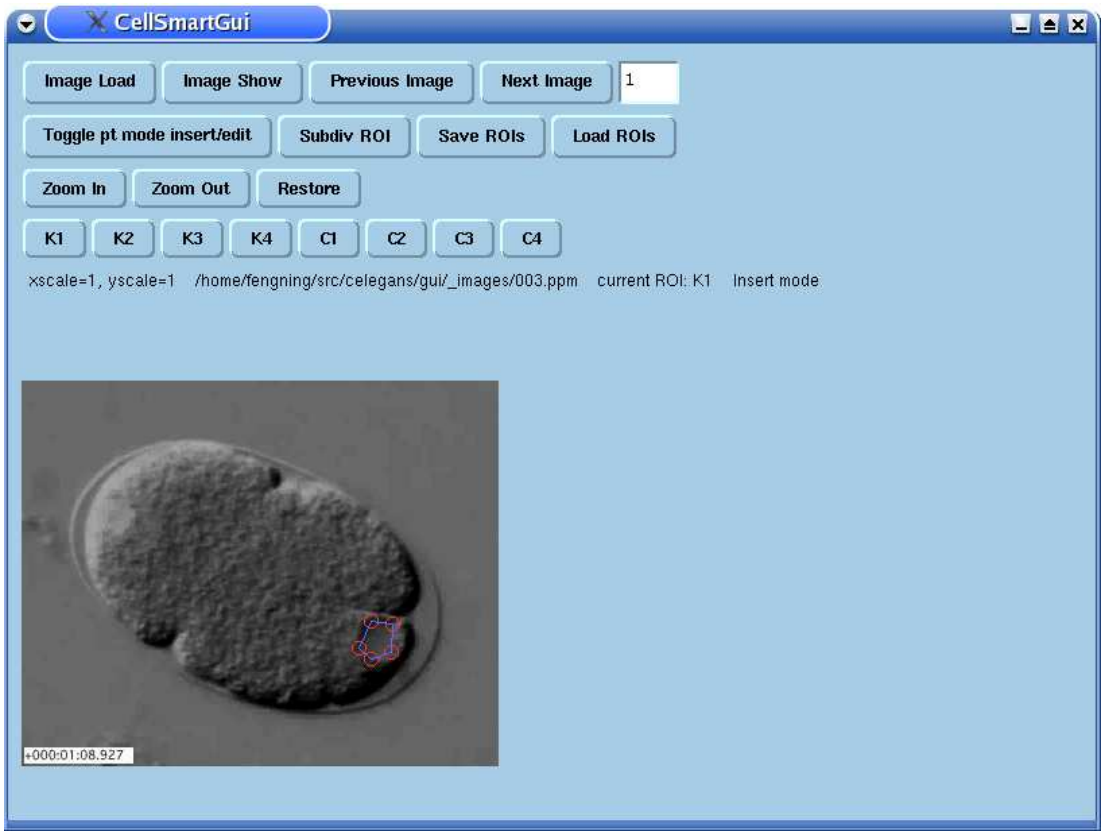

Figure A.1: Screenshot of the *CellSmartGui*, the GUI front-end of *CellSmart* application.

Appendix B

TemplateSmart

Chapter 5 discussed the design of elastic template fitting module. Here we introduce *TemplateSmart*, LUSH application we implemented for the module. It reads all parameters (the meta-template loaded, the initial geometric transformation applied on the meta-template, spring stiffness, number of EM iterations, etc) from its front-end, runs the EM-based algorithm and writes the result of fitting to disk files, in the same ROI format as in *CellSmart*.

There are two front-ends for *TemplateSmart*. One front-end is GUI-based, called *TemplateSmartGUI*. The other is non-GUI, used for batch processing. The relationship of *TemplateSmart* to its front-ends can be seen from figure B.1.

The *TemplateSmartGUI* front-end allows the user to control the fitting process interactively, by tweaking the various system parameters. It will

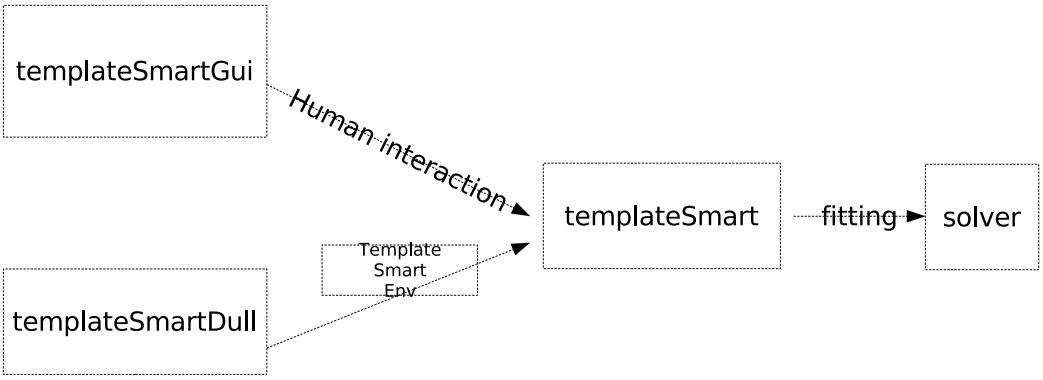

Figure B.1: *TemplateSmart* and its two front-ends.

display the template overlayed on the label image, side-by-side with the input image.

Below we show some sample snapshots of *TemplateSmart* in action. Figures B.2 and B.3 show the interface before and after fitting a T4 template.

The non-GUI front-end works as a driver program to run the fitting process non-interactively. Instead of acquiring input from the user, it reads the parameters from a disk file (the LUSH class *TemplateSmartEnv* wraps all the parameters). A typical use case is that we run the fitting process multiple times on the same image, with different sets of parameters. A higher level of module will then calculate some measure of "fitness" to select the best fitted template.

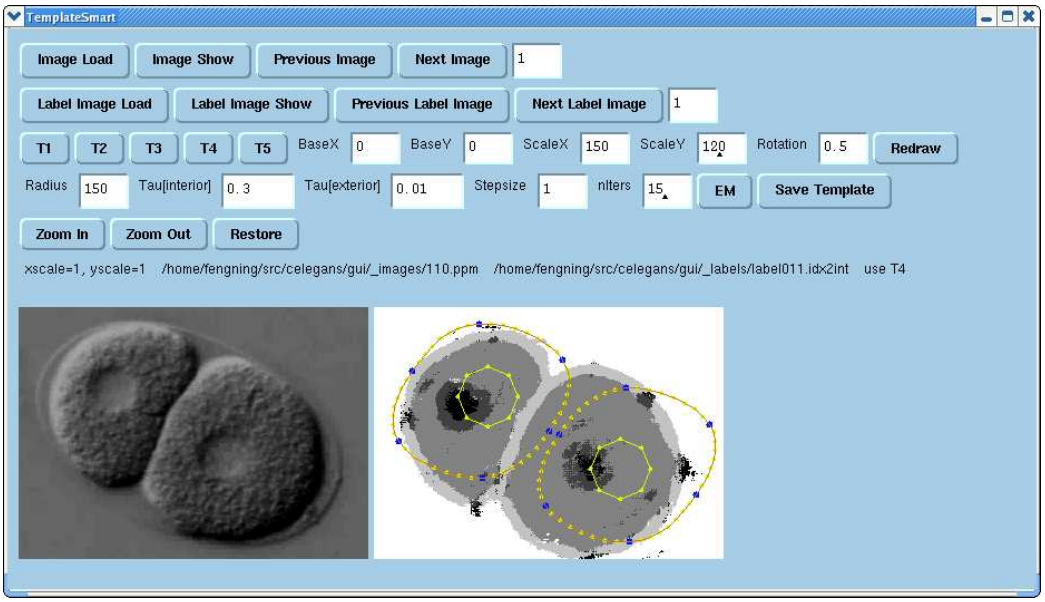

Figure B.2: Screenshot: overlaying a T4 template on a label image.

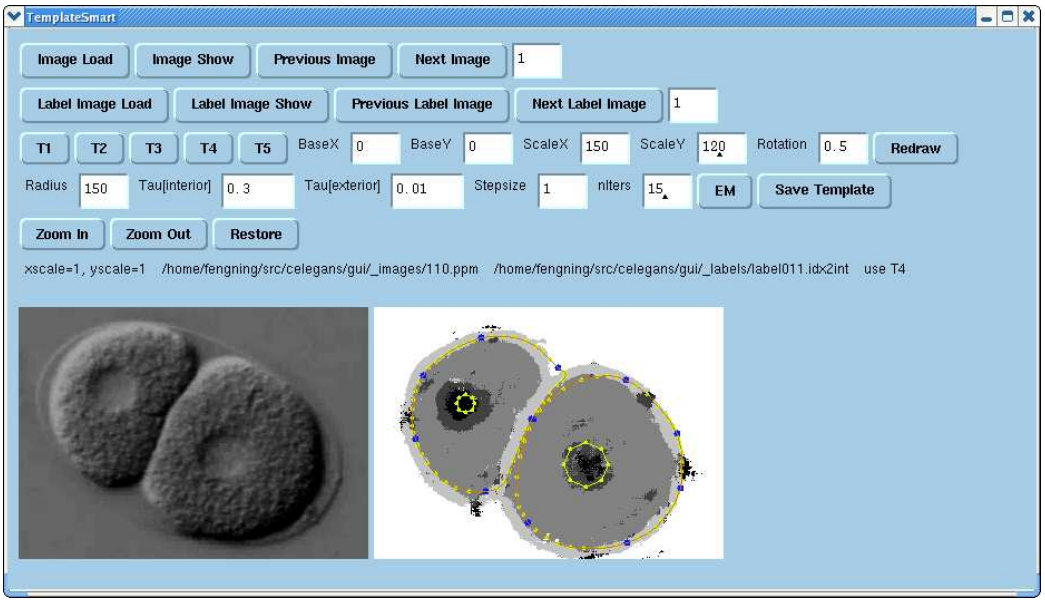

Figure B.3: Screenshot: fitting a T4 template to a label image.

Appendix C

Image denoising using EBM

We present our EBM-based image denoising study in this appendix. It serves as a working example to demonstrate the power of Convergence Divergence learning and the capability of EBM. As result, we feel justified to include the section in the thesis, even it is not incorporated in the phenotyping system we discussed.

As we discussed in chapter 4, training an EBM consists in finding values of the trainable parameters that associate *low energies* to "desired" configurations of variables (e.g. observed on a training set), and *high energies* to "undesired" configurations. This rationale is also valid for training an EBM for image denoising. Being able to denoising rather than discriminating "desired" configurations from "undesired" ones imposes one more requirement: following energy-decreasing path (e.g. the gradient-descent path), the states

should be more and more denoised. This requirement is imposed on the energies of a sequence of states, not only on two states. EBM is trained using Contrastive Divergence method for the sake of efficiency. A related research can be found in the study of Field of Experts (FoE) [RB05]. In fact, the FoE can be viewed as a specially designed EBM. Both the FoE and our EBM are trained using Contrastive Divergence. However, there are still significant difference between the two studies. For example, we use different strides for training and inference.

The data

Our study uses the Berkeley segmentation dataset [BSD]. We build our training dataset as 2000 patches of size 32×32. The location and the image id numbers of these patches are random. These patches are "positive" patches that low energies should be assigned with.

Figure C.1: The sample patches, from Berkeley segmentation dataset.

The network

The energy module is a two-layer convolutional network. The first layer is a convolution layer with 10 feature maps, each with a 8×8 trainable kernel. The nonlinearity function is $g(u) = 1 - 1/(1 + u^2)$. The second layer is a plain summation layer, whose output is the sum of all inputs of all feature maps at all locations. We remark that our network is so far not uniquely decided yet: we have the freedom to choose stride for the convolution layer. In practice, we use $stride = 8$ for training and $stride = 1$ for inference.

C.0.3 Training the EBM

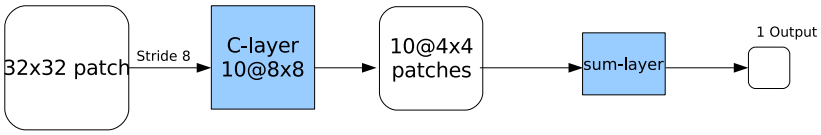

Figure C.2: The network architecture for training. We use $stride = 8$.

We use a simple margin loss:

$$L(\mathbf{W}) = \sum_i E(\mathbf{X}_i) + (m - min_{\|\mathbf{Y}-\mathbf{X}_i\| \geq \delta} E(\mathbf{Y}))^+.$$

The margin is chosen to be $m = 15$. The system is initialized with random parameters. We train the system for 100 epochs via loss minimization using gradient descent. The loss function above is modified by adding a penalty term $\nu \|W\|_{L^1}$ ($\nu = 1e-5$) to encourage the decay of weights. The "negative"

state $\mathbf{Y}$ is obtained by Contrastive Divergence, running one Hybrid Monte Carlo step which is composed of 30 Leapfrog steps. Overall acceptance is above 85% during the training. We can see local patterns after training as in figure C.3.

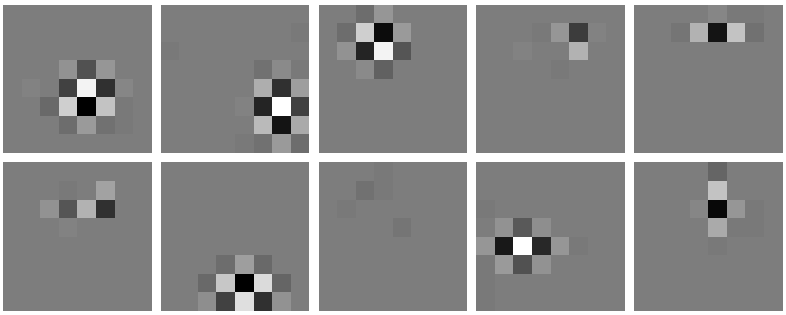

Figure C.3: The kernels learned after 100 epochs.

We examine how the energies associated with the training patches and the noisy patches. We create 2000 noisy patches by adding Gaussian noise to the ground truth image patches (noise level = 200 for pixel range $[0, 255]$). By comparison of figure C.4 and figure C.5, we find that Contrastive Divergence successfully adapt the energy surface to discriminate "positive" samples and negative samples.

Denoising an image patch

Denoising consists of energy minimization from the state of the noisy patch $\mathbf{Y}^0$. We add a penalty term $\eta\|\mathbf{Y}-\mathbf{Y}^0\|^2$ to the energy to discourage any large

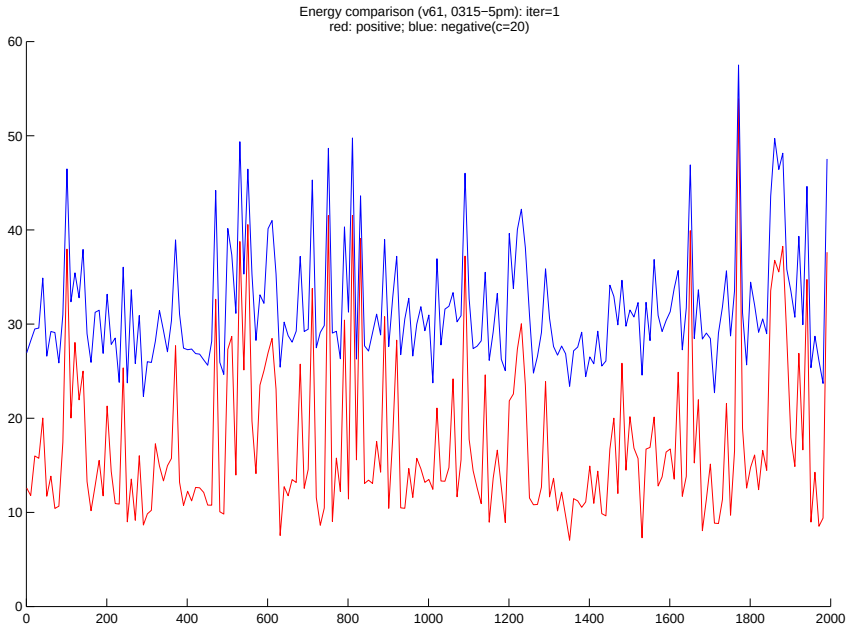

Figure C.4: The energies on training data, epoch 1.

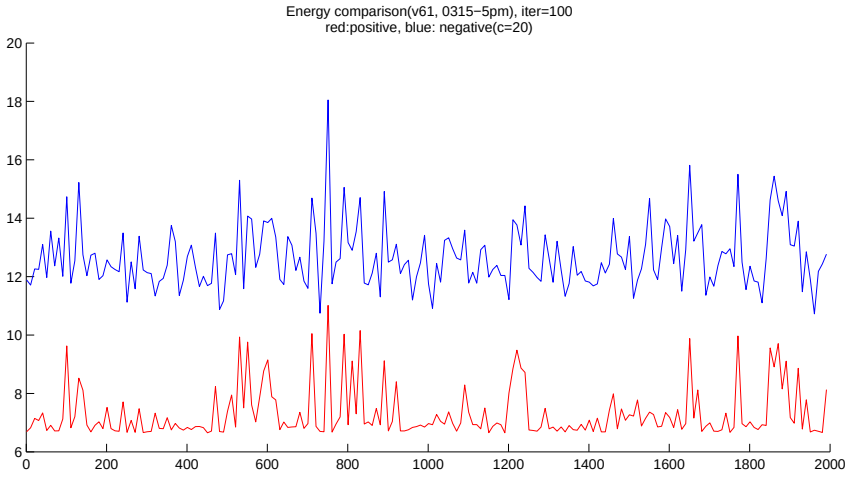

Figure C.5: The energies on training data, epoch 100.

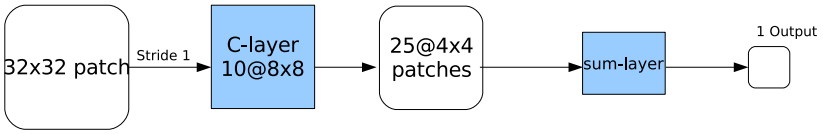

Figure C.6: The network architecture for inference. We use $stride = 1$.

change from the noisy patch. This is necessary to avoid a trivial solution to be found (e.g. a uniform background patch which is perfectly smooth and natural but recover nothing from the original patch). We take $\eta = 0.01$ as shown below. Figure C.7 shows the denoising process working on a particular patch. We show the original noisy patch, then all the intermediate patches for every two steps of denoising, up to 50 steps. We can see that high frequency noisy are effectively abated while the major texture is preserved.

Denoising a whole image

Without changing the network, we can denoise a whole image. For example, we pick out the famous "peppers" image from Berkeley Segmentation Dataset and add Gaussian noise (noise level 20, for pixel range $[0, 255]$) to it, see figure C.8. We run the inference for 500 steps to obtain the denoised image in figure C.9. The original noise image has Peak-to-Noise-Ratio (PSNR) value 22.40 and the denoised image has improved PSNR 27.73.

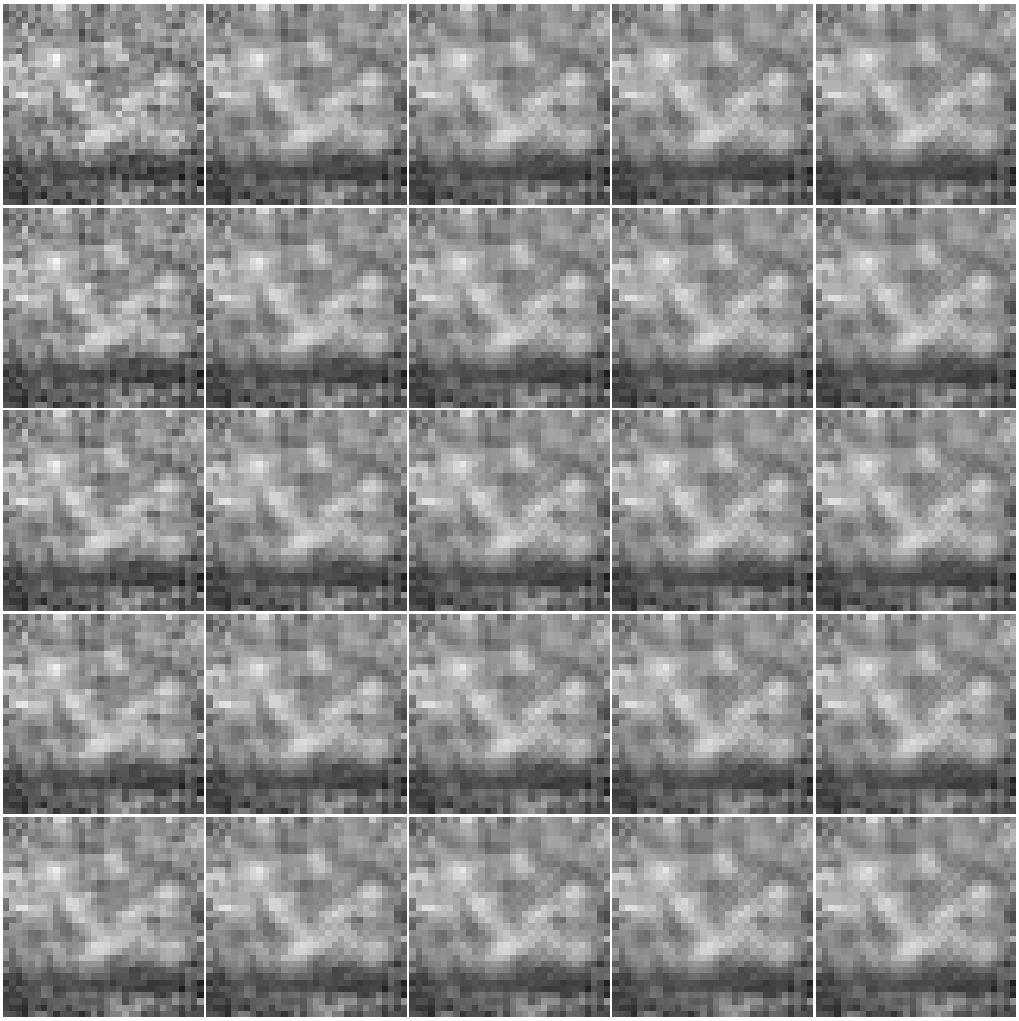

Figure C.7: Denoising in action. Patches are shown every two steps. Order: top to down, then left to right.

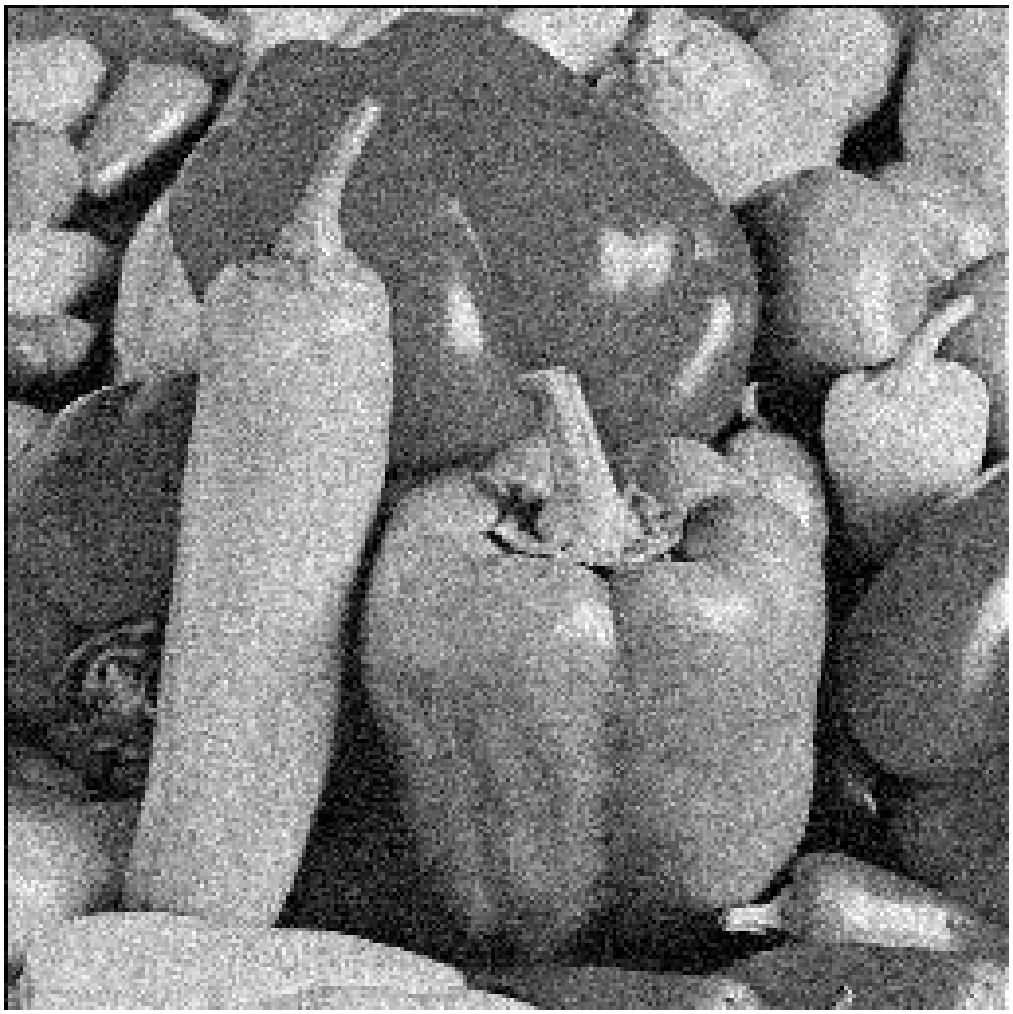

Figure C.8: The pepper image, with $c = 20$ Gaussian noise. $PSNR = 22.40$.

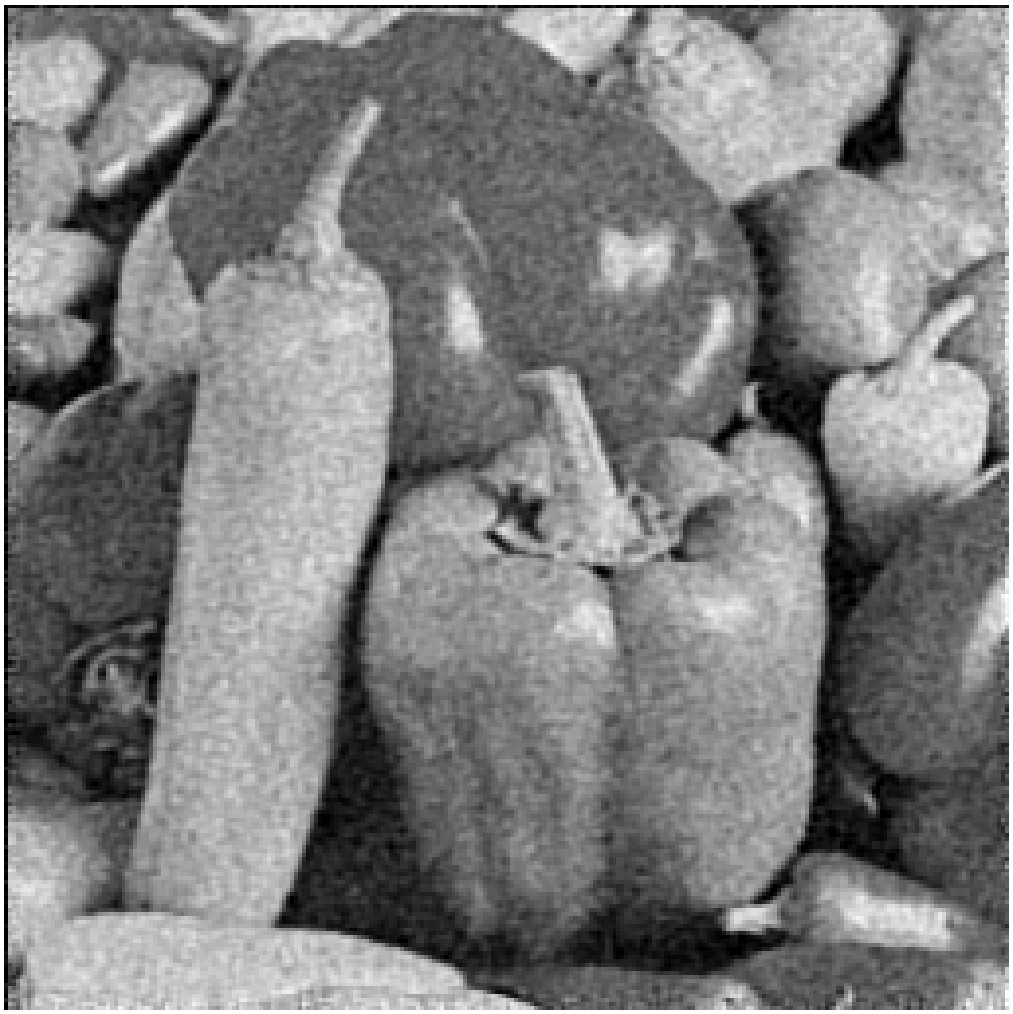

Figure C.9: The pepper image, denoised after 500 steps. $PSNR = 27.73$.

Bibliography

[AM93] Arnaldo J. Abrantes and Jorge S. Marques. A common framework for snakes and Kohonen networks. In *Proceedings of the 1993 IEEE Conference on Neural Networks for Signal Processing*, pages 251–260, 1993.

[AZL96] A.K.Jain, Y. Zhong, and S. Lakshmanan. Object matching using deformable templates. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(3):267–278, Mar 1996.

[BL94] Y. Bengio and Y. LeCun. word normalization for on-line handwritten word recognition. In IAPR, editor, *Proc. of the International Conference on Pattern Recognition*, volume II, pages 409–413, Jerusalem, October 1994. IEEE.

[BMM98] M. V. Boland, M. K. Markey, and R. F. Murphy. Automated recognition of patterns characteristic of subcellular structures in fluorescence microscopy images. *Cytometry*, 33:366–375, 1998.

[BSD] Berkeley segmentation dataset. `http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbe%nch/`.

[CB01] H. Cheng and C. A. Bouman. Multiscale bayesian segmentation using a trainable model. *IEEE Transactions on Image Processing*, 10(4):511–525, 2001.

[CEW+04] C. Conrad, H. Erfle, P. Warnat, N. Daigle, T. Lorch, J. Ellenberg, R. Pepperkok, and R. Eils. Automatic identification of subcellular phenotypes on human cell arrays. *Genome Research*, 14:1130–1136, 2004.

[CHL05] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verificatio. In *Proc. of Computer Vision and Pattern Recognition Conference*. IEEE Press, 2005.

[con98] The *C. elegans* Sequencing consortium. Genome sequence of the nematode *C. elegans*: a platform for investigating biology. *Science*, 11(282):2012–2018, Dec 1998.

[FXM+98] A. Fire, S. Xu, M.K. Montgomery, S.A. Kostas, S.E. Driver, and C.C. Mello. Potent and specific genetic interference by double-

stranded rna in *Caenorhabditis elegans*. *Nature*, 391(6669):806–811, Feb 1998.

[GCR91] G.E.Hinton, C.K.I.Williams, and M. Revow. Adaptive elastic models for hand-printed character recognition. In *Advances in Neural Information Processing Systems (NIPS*1991)*. MIT Press, 1991.

[GD04] Christophe Garcia and Manolis Delakis. Convolutional face finder: A neural architecture for fast and robust face detection. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 26(11):1408–1423, Nov 2004.

[GEO+00] P. Gonczy, C. Eheverri, K. Oegema, A. Coulson, S.J. Jones, R.R. Copley, J. Duperon, M. Brehm, E. Cassin, M. Kirkham, S. Pichler, K. Flohrs, A. Goessen, S. Leidel, A.M. Alleaume, C. Martin, N. Ozlu, P. Bork, and A.A. Hyman. Functional genomic analysis of cell division in *C. elegans* using RNAi of genes on chromosome III. *Nature*, 408(6810):331–336, Nov 2000.

[G.S95] G.S.Fishman. *Monte Carlo Concepts, Algorithms and Applications*. Springer-Verlag, 1995.

[Hin02] G.E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Comput.*, 2002.

[HLB04] Fu-Jie Huang, Yann LeCun, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Proceedings of CVPR'04*. IEEE Press, 2004.

[HM04] K. Huang and R. F. Murphy. From quantitative microscopy to automated image understanding. *J. Biomed. Optics*, 9(5):893–912, 2004.

[HZ87] R. A. Hummel and S. W. Zucker. *On the foundations of relaxation labeling processes*, pages 585–605. Morgan Kaufmann Publishers Inc., 1987.

[Ima] Imagej: Image processing and analysis in java. `http://rsb.info.nih.gov/ij/`.

[JGJS98] Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 1998.

[KFL01] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 2001.

[Koh84] T. Kohonen. *Self-Organization and Associative Memory (2nd edition)*. Springer Verlag, Berlin, New York, 1984.

[KWT87] M. Kass, A. Witkin, and D. Terzopoulos. Snake: Active contour models. *International Journal of Computer Vision*, 1(4):321–331, 1987.

[LBBH98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceddings of the IEEE*, 86(11):2278–2324, Nov 1998.

[LGTB97] Steven. Lawrence, C. Lee Giles, Ah Chung Tsoi, and Andrew Back. Face recognition: A convolutional neural network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, 1997.

[Li95] S. Z. Li. *Markov random field modeling in computer vision*. Springer-Verlag, 1995.

[LMP01] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: probabilistic models for segmenting and labeling sequence data. *International Conference on Machine Learning*, pages 282–289, 2001.

[lus] Lush. `http://lush.sf.net`.

[Mac03] Mackay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[MCM04] D. Martin, C.Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color and texture cues. *Trans. on PAMI*, 26(5):530–549, Jan 2004.

[MFTM01] 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. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.

[MT96] T. McInerney and D. Terzopoulos. Deformable models in medical image analysis: a survey. *Medical Image Analysis*, 1(2):91–108, Mar 1996.

[NDL+05] Feng Ning, Damien Delhomme, Yann LeCun, Fabio Piano, Léon Bottou, and Paolo Emilio Barbano. Toward automatic phenotyping of developing embryos from videos. *IEEE Transactions on Image Processing*, 2005.

[NP95] S. Nowlan and J. Platt. A convolutional neural network hand tracker. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Ad-*

vances in Neural Information Processing Systems 7, pages 901–908, San Mateo, CA, 1995. Morgan Kaufmann.

[OML05] R. Osadchy, M. Miller, and Y. LeCun. Synergistic face detection and pose estimation with energy-based model. In *Advances in Neural Information Processing Systems (NIPS*2004)*. MIT Press, 2005.

[PR94] M. Pelillo and M. Refice. Learning compatibility coefficients for relaxation labeling processes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(9):933–945, 1994.

[PSM+00] F. Piano, A.J. Schetter, M. Mangone, L. Stein, and K.J. Kemphues. RNAi analysis of genes expressed in the ovary of *Caenorhabditis elegans*. *Current Biology*, 10(24):1619–22, Dec 2000.

[PSM+02] F. Piano, A.J. Schetter, D.G. Morton, K.C. Gunsalus, V. Reinke, S.K. Kim, and K.J. Kemphues. Gene clustering based on rnai phenotypes of ovary-enriched genes in *C. elegans*. *Current Biology*, 12(22):1959–64, Nov 2002.

[PT00] W. Pieczynski and A.N. Tebbache. Pairwise markov random fields and its application in textured image segmentation. *Proc.*

of 4th IEEE Southwest Symposium on Image Analysis and Interpretation, pages 106–110, Nov 2000.

[RB05] Stefan Roth and Michael J. Black. Fields of experts: A framework for learning image priors. *Proceedings of Computer Vision and Pattern Recognition*, 2005.

[R.M93] R.M.Neal. Probabilistic inference using markov chain monte carlo methods. Technical report, University of Torento, 1993.

[RWH96] M. Revow, C. K. I. Williams, and G. E. Hinton. Using generative models for handwritten digit recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18, 1996.

[SM03] S.Kumar and M.Hebert. Discriminative fields for modeling spatial dependencies in natural images. *Neural Information Processing System*, 2003.

[Teh03] Yee Whye Teh. *Bethe Free Energy and Contrastive Divergence Approximations for Undirected Graphical Models*. PhD thesis, U.of Torento, 2003.

[TWOE03] Y. W. Teh, M. Welling, S. Osindero, and Hinton G. E. Energy-based models for sparse overcomplete representations. *Journal of Machine Learning Research*, 4:1235–1260, 2003.

[VJ01] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, pages 511–518, 2001.

[VKO97] J. Vleugels, J. Kok, and M. Overmars. Motion planning using a colored Kohonen network. *International journal of neural systems*, 8:613–628, 1997.

[VML94] R. Vaillant, C. Monrocq, and Y. LeCun. Original approach for the localization of objects in images. *IEE Proc. on Vision, Image, and Signal Processing*, 141(4):245–250, August 1994.

[vvA97] E. van Munster, L. van Vliet, and J Aten. Reconstruction of optical pathlength distribution from images obtained by a wide-field differential interference contrast microscope. *Journal of Microscopy*, 188(2):149–157, 1997.

[Wal04] H.M. Wallach. Conditional random fields: An introduction. *University of Penslvania tech report*, 2004.

[WL03] R. Wilson and C. T. Li. A class of discrete multiresolution random fields and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(1):42–56, 2003.

[XP97] C. Xu and J. Prince. Gradient vector flow: A new external force for snakes. In *Proceedings of Computer Vision and Pattern Recognition (CVPR '97)*, pages 66–71, San Juan, Puerto Rico, June 1997. IEEE.

[YBO+99] T. Yasuda, H. Bannai, S. Onami, S. Miyano, and S. Kitano. Towards automatic construction of cell-lineage of *C. elegans* from Nomarski DIC microscope images. *Genome Informatics*, 10:144–154, 1999.

[YFW01] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Generalized belief propagation. *Advanced Neural Information Processing Systems*, 13:689–695, 2001.

[YFW02] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free energy approximations and generalized belief propagation algorithms. Technical report, Mitsubishi Electric Research Laboratories, 2002.

[YH05] Y.LeCun and J. Huangfu. Loss functions for discriminative training of energy-based graphical models. *NIPS*, 2005.

[ZFK+01] P. Zipperlen, A.G. Fraser, R.S. Kamath, M. Martineez-Campos, and J. Ahringer. Roles for 147 embryonic lethal genes on *C.*

elegans chromosom I identified by RNA interference and video microscopy. *EMBO Journal*, 20(15):3984–3992, Aug 2001.