December 2012

Automatic Foreground Initialization for Binary Image Segmentation

Wei Li
The University of Western Ontario

Supervisor
Olga Veksler
The University of Western Ontario

Graduate Program in Computer Science

A thesis submitted in partial fulfillment of the requirements for the degree in Master of Science

AUTOMATIC FOREGROUND INITIALIZATION FOR BINARY IMAGE SEGMENTATION

(Thesis format: Monograph)

by

Wei Li

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Masters of Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

THE UNIVERSITY OF WESTERN ONTARIO
School of Graduate and Postdoctoral Studies

CERTIFICATE OF EXAMINATION

Examiners:

Supervisor:

.....
Dr. Steven Beauchemin

.....
Dr. Olga Veksler

.....
Dr. Mark Daley

Supervisory Committee:

.....
Dr. Kenneth McIsaac

The thesis by

Wei Li

entitled:

Automatic Foreground Initialization for Binary Image Segmentation

is accepted in partial fulfillment of the
requirements for the degree of
Masters of Science

.....
Date

.....
Chair of the Thesis Examination Board

Abstract

Foreground segmentation is a fundamental problem in computer vision. A popular approach for foreground extraction is through graph cuts in energy minimization framework.

Most existing graph cuts based image segmentation algorithms rely on users initialization. In this work, we aim to find an automatic initialization for graph cuts. Unlike many previous methods, no additional training dataset is needed. Collecting a training set is not only expensive and time consuming, but it also may bias the algorithm to the particular data distribution of the collected dataset.

We assume that the foreground differs significantly from the background in some unknown feature space and try to find the rectangle that is most different from the rest of the image by measuring histograms dissimilarity. We extract multiple features, design a ranking function to select good features, and compute histograms based on integral images.

The standard graph cuts binary segmentation is applied, based on the color models learned from the initial rectangular segmentation. Then the steps of refining the color models and re-segmenting the image iterate in the grabcut manner, until convergence, which is guaranteed.

The foreground detection algorithm performs well and the segmentation is further improved by graph cuts. We evaluate our method on three datasets with manually labelled foreground regions, and show that we reach the similar level of accuracy compared to previous work. Our approach, however, has an advantage over the previous work that we do not require a training dataset.

Keywords: image features, graph cuts, discrete optimization, object detection, automatic segmentation, iterative energy minimization, expectation maximization

Acknowledgements

My deepest gratitude goes first and foremost to Dr. Olga Veksler, my supervisor, for her constant encouragement and guidance. She has walked me through all the stages of this thesis. I learnt a lot from her serious attitude toward research. Meanwhile, I really enjoy the process of discussing with her like old friends. Without her consistent and illuminating instruction, this thesis could not have been possible.

I would like to express my heartfelt thank to Dr. Yuri Boykov, who led me into the world of computer vision and image processing. I was delighted to attend his lectures and I appreciate the various and impressing ways he employed to explain every detail clearly.

Special thanks are given to the members of my examining committee, Dr. Steven Beauchemin, Dr. Mark Daley and Dr. Kenneth McIsaac.

I am also greatly indebted to Dr. Steven Beauchemin, Dr. Roberto Solis-Oba, and Dr. James H. Andrews, who have instructed and helped me in the past year.

I would extend my sincere thanks to students in Vision Group, Paria Mehrani who provided me images for testing and comparing the algorithms. Also Dr. Lena Gorelick, Dr. Yu Liu, Dr. Andrew Delong, Hossam Isack, Junwei Sun and Greg Elfers, for patiently answering my questions and I learnt quite a lot from the discussions with them.

I would also like to thank staff members in the main office and system group, for their help and support.

My thanks would go to my beloved family for their loving considerations and great confidence in me all through these years.

Last but not the least, I especially appreciate the support of my husband, Shenggang Shang, who is always there loving me, helping me, and encouraging me to do what I like.

Contents

List of Figures

List of Tables

List of Appendices

Chapter 1

Introduction

1.1 Overview

The goal of image segmentation is to cluster pixels into meaningful image regions, i.e., regions corresponding to objects, natural parts of objects, or individual surfaces.

As a human, we can finish the tasks of object recognition and segmentation very fast, while developing a computer system that can automatically and in real time detect and segment an object is something that computer vision scientists have been working on for decades. Many researchers tried various segmentation techniques to make computers mimic human vision processing. However, we have to admit that there is still a long way to go before computer performance can compare with a human, even on a relatively simple task of foreground segmentation.

In this thesis, we focus on "binary" automatic segmentation of an input image. Intuitively, given an input image, the task is to separate it into two regions, one corresponding to the foreground, and the other one to the background (see figure 1.1), based on the feature differences in the two parts.

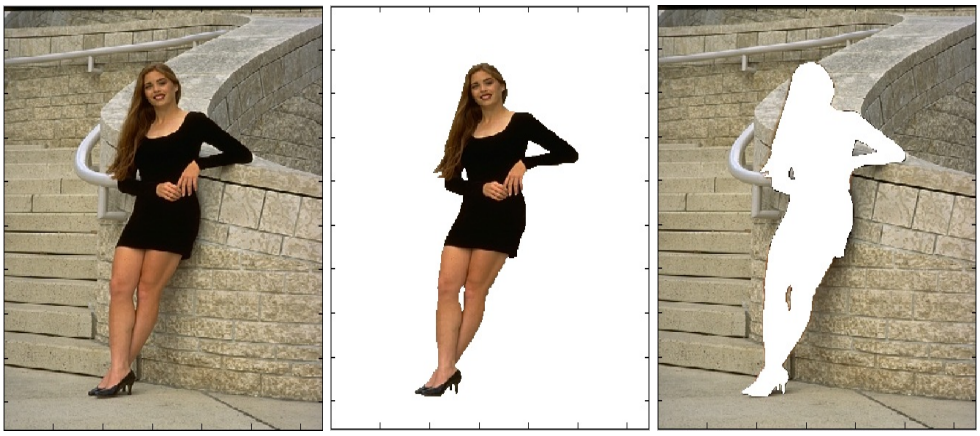


Figure 1.1: Left: original image, middle: foreground region, right: background region

1.1.1 Application Prospect of Automatic Image Segmentation

Automatic image segmentation can be applied in object recognition [14], image compression [70], image editing [45], image searching [48] and other tasks of machine vision.

In industry and daily life, the applications of image segmentation lie in different aspects. Such as disease diagnosis [7, 8, 39, 37], including localization of tumors and other pathologies, measuring tissue volumes, and computer-guided surgery, etc. In remote sensing interpretation [24], image segmentation is being used to locate objects in satellite images (roads, forests, etc.). In order to maintain security, face recognition [40], fingerprint recognition technique can be helpful. On the other hand, traffic control systems, such as brake light detection [13], is another application of automatic image segmentation in practice.

1.1.2 Motivation and Challenge of This Work

Besides the practical needs, our work was motivated by the graph cuts algorithm [8] for object segmentation , which makes accurate and efficient automatic segmentation possible, in principle. Segmentation with graph cuts is attractive because it allows formulation of an objective function that encodes the desired property of segmentation, such as color coherence of the object/background, smoothness of the boundary, etc, as well as a method for globally optimizing this objective function.

Graph cuts, as well as the following publications that directly build upon it, such as grabcut [62], lazy snapping [49], star shape prior [74], paint selection [51], etc. are effective interactive segmentation approaches. However, there are some drawbacks in those methods. The segmentation performance is very dependent on user-specified seeds or other form of initialization. Often, additional interactions are necessary when the initialization is not precise enough. Moreover, the user interactions are time-consuming and therefore infeasible in certain applications [35]. The pros and cons of interactive segmentation with graph cuts are the direct inspiration of this work.

Figure 1.2: Left: an ambiguous example, right: an unambiguous example

The challenges of this work mainly rise from two aspects. First, image segmentation, including binary image segmentation, is a highly ambiguous problem. Consider the left image in figure 1.2. Should the birds or the nest or all of them be in the foreground? Different viewers will have different opinions on this subject. Therefore a plethora of interactive, that is user-guided, segmentation methods were designed [8, 62, 49, 36, 58, 21]. However, there are many cases that are unambiguous, or at least where most of the viewers would agree on a single foreground object (see the right image in figure 1.2). The goal or our thesis is automatic foreground segmentation in cases that are unambiguous.

The second challenge of proposed method lies in the fact that natural images are usually quite complex both in foreground and background appearance. The complexity comes from the object and the surrounding, such as wide range of color, texture, as well as imaging conditions, viewing angles, inter-reflections in the scene, etc. The input of the task is usually a single image, or several closely-related images, for example, a stereo pair [12], a common foreground object with varying backgrounds [63], or an image sequence [46]. Independent of the input type, the objects blend into the background seamlessly, which due to the loss of 3D information. There are also cases of animal camouflage, where an object (an animal) has developed appearance similar to its background in order to fool the visual system of its predators by blending in with the background, see crocodile example in figure 1.3. Therefore, it is quite difficult to segmentation natural images automatically.

Figure 1.3: Camouflage example

1.2 Our Approach

1.2.1 Methodology

In this thesis, we propose an automatic foreground region detection method, which will be a starting point for a more accurate binary segmentation with graph cuts. The foreground detecton is based on texture and color features, in other words, our feature space is the combination of texture and color space. Our features might be redundant and our feature space is not orthogonal, but over-complete representations are frequently more successful in vision compared

to non-redundant representations.

Different from graph cuts [8] or grabcut [62] methods, this work employs an automatic searching strategy to detect a rough foreground location instead of asking for foreground/background seeds or foreground rectangle from a user.

Our approach is also different from supervised machine learning approach, that trained on a large collection of pre-labeled images to learn to detect the foreground object [5, 29]. A downside of these approaches is that they require a large training set and the results are sensitive to the particular dataset used for training [71]. Our approach is to find a feature space in which the object is quite different from the background, which can be accomplished easier.

To a large extent, our method is close in spirit to saliency detection methods [32, 25]. The difference between them is that instead of combining different features with different weights, like they do, we are selecting features to make some rectangle stand out from the background.

The aim of our work is to achieve an accurate automatic object segmentation method. Figure 1.4 shows the flow chart of our method, which can be roughly divided into three steps.

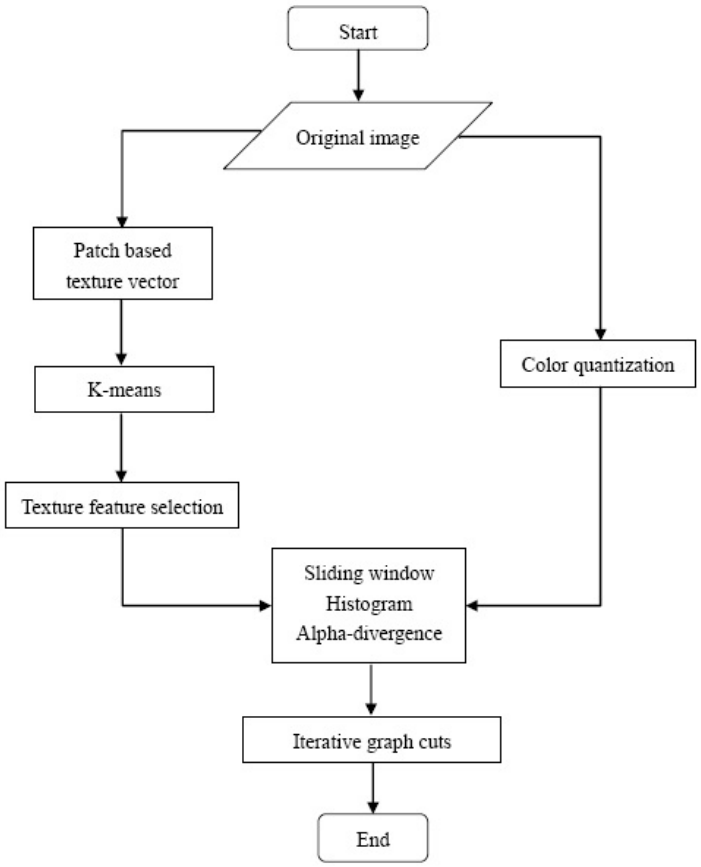


Figure 1.4: The flow chart of the thesis

First, we extract and select points of interest in feature space. In particular, an image patch [72] is employed to capture texture characteristics and image quantization algorithm is applied to quantize color characteristics. Since a feature is only useful if it helps to separate the object from the background, a ranking function is designed with the goal to rank each feature as being useful or not. The ranking function looks at the feature distribution throughout the image. If a feature spread more or less uniformly throughout the image, such feature is judged as useless. If a feature is highly concentrated in some spatial area of the image, it is likely to be useful and is ranked higher. After feature ranking, lower-scoring features are filtered out because they are unhelpful for foreground/background discrimination.

Our second step is to search for a rectangular region where the foreground is probably located by measuring the divergence of feature histograms inside and outside this region. The greater the divergence, the more likely the rectangle contains the foreground. The reason we choose rectangular shape is that histograms can be computed efficiently for rectangles based on integral images [76].

The last step is to model the foreground/background regions from the rectangular initialization and perform iterative segmentation in the binary graph cuts [8, 62] style.

1.2.2 The Advantages of Our Method

We evaluate our algorithm on three datasets: Berkeley (BSD)[1] [54], Grabcut (GSD)[2] and Achanta et al.(ASD) [1]. Our approach performs in the similar level or even better compared with the results from previous work. For example, in figure 1.5, we segment the same image as well as the result from grabcut [62] algorithm, that user initialization and editing are needed. In figure 1.6, we get more precise segmentation for koala image than the work in [56], which initialize graph cuts based on saliency map trained on manually labeled dataset. Our algorithm reaches the similar average accuracy compared to the method [56]. For more results and comparison, see chapter 7.

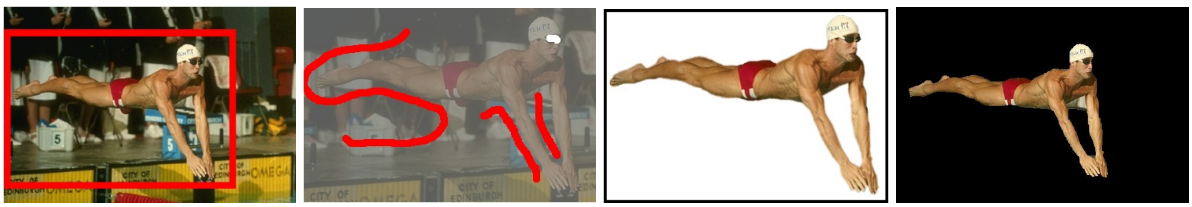


Figure 1.5: From left to right: grabcut initialization rectangle, grabcut user editing, grabcut segmentation result and our result

[1] http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/

[2] http://research.microsoft.com/en-us/um/cambridge/projects/visionimagevideoediting/segmentation/grabcut.htm

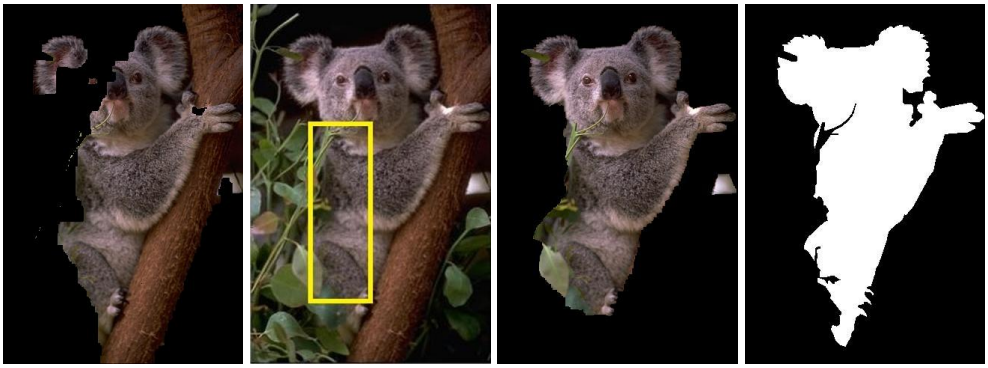


Figure 1.6: From left to right: segmentation result from previous work [56], our foreground detection result, our segmentation result and ground truth

Meanwhile, our algorithm shows significant advantages in many aspects, which are listed as follows:

- **Automatic segmentation** Although the existing interactive segmentation methods have made an impressive progress in the last decade, enabling the computer to segment images automatically has obvious advantages, such as applicability in domains where user interaction is not possible or desirable. Automatic segmentation methods have a wider application base than the interactive ones. Our work targets to find an effective way to implement automatic segmentation in energy minimization framework.

- **Foreground detection without dataset bias** There are automatic foreground segmentation techniques based on learning from a labelled dataset [56], however, training on a particular dataset can give a method that is tuned (biased) to the particular data distribution in that dataset [71]. If the object of interest is very different from those appearing in the training dataset, such learning-based methods are more likely to fail. On the contrary, our proposed method mines the input image itself, which makes it applicable to any object class, as long as we can find features that distinguish it from the background.

- **No training dataset required** We work on single input image and only mine the input image itself, making use of low-level visual cues, such as intensity, color and texture. Hence, we do not need to collect and label a large training set, which is usually very labor intensive and time consuming.

- **Multi-object segmentation compatibility** Most existing region based segmentation algorithms can segment out a single blob with the homogeneous features, especially the early methods like region growing [27], watershed [75] and splitting-and-merging [30] methods, etc. In other words, they only face to single object segmentation. Although the first stage of our algorithm also targets to find a single rectangle region to mark the foreground location, it only acts as the initialization of graph based segmentation. At the later stage, segmentation with graph cuts is able to find several disconnected objects, as long as long as they have similar appearance.

1.3 Outline of the Thesis

This thesis is organized as follows. Chapter 2 is a brief introduction to image features. Then chapter 3 introduces binary image segmentation with graph cuts in energy minimization framework. Chapter 4 reviews and analyses the previous works on automatic and interactive segmentation. Chapter 5 focuses on the first phase of the proposed method which is foreground included rectangle detection. In chapter 6, details of foreground/background segmentation are explained. Some experimental results are provided in chapter 7. Finally, conclusion and future work is in chapter 8.

Chapter 2

Image Features

The first stage of this thesis is based on image feature analysis to detect the potential object location. In particular, we choose the combination of texture and color as our feature space, and detect the rectangle where the features inside and outside of the rectangle show the largest dissimilarity. This chapter gives a brief overview of image features.

2.1 Feature, Feature Vector and Feature Space

In computer vision, a feature is a term that is usually used to denote a piece of information which is useful for solving a specific task. Understandable, there is no universal or exact definition of what constitutes a feature, since the exact definition that makes sense often depends on the problem or the type of application. Informally, a feature is understood as an "interesting" part of an image.

In practice, the set of features of a given data instance is often grouped into a feature vector. This is because no single feature is usually enough to describe sufficient amount information. Instead, two or more different features are extracted, resulting in two or more feature descriptors at each image point. A common practice is to organize the information provided by all feature vectors as the elements of one single vector, referred to as a feature vector.

The set of all possible feature vectors constitutes a feature space. Within this thesis, there is a basic assumption that the foreground and background of the input image can be well separated in certain feature space.

2.2 Feature Types

Many computer vision algorithms use feature extraction as the initial step, as a result, all kinds of feature detectors have been developed. There is a large variability in the ways features are detected, the computational complexity and the repeatability. At an overview level, these features can be divided into different groups (with some overlap), for example, feature at each pixel (texture, intensity, color, gradient, motion, and stereo depth cue) and feature of a group

of pixels, such as edge, corner, blob and ridge. Here, a short introduction is given about texture and color features tested in this work, with explanations about how we make the choice.

2.2.1 Texture Descriptors

In this section we describe a few popular features for capturing texture information in an image.

- **SIFT**

  In Scale Invariant Feature Transform (SIFT) [53] algorithm, features are detected through a staged filtering approach that identifies stable points in scale space. Image keys are created that allow for local geometric deformations by representing blurred image gradients in multiple orientation planes and at multiple scales. SIFT feature is invariant to image scaling, translation, and rotation, and partially invariant to illumination changes and affine or 3D projection.

  Intuitively, SIFT features capture how edges are oriented in the neighborhood of a given pixel, thus capturing texture information. In particular, SIFT first transforms an image into a large collection of local feature vectors, then maxima and minima of a difference of Gaussian function are applied in scale space to select key locations. The pixels that fall in a circle of radius 8 pixels around the key location are inserted into the orientation planes. The orientation is measured relative to that of the key by subtracting the keys orientation. Usually, 8 orientation planes are used, each sampled over a 4×4 grid of locations, with a sample spacing 4 times that of the pixel spacing used for gradient detection. The blurring is achieved by allocating the gradient of each pixel among its 8 closest neighbors in the sample grid, using linear interpolationin orientation and the two spatial dimensions. The resulting SIFT descriptor has length of 128. Figure 2.1 shows how SIFT feature is extracted.

  In our work, SIFT feature is used to present the texture characteristic around every pixel, which is relatively computationally expensive.

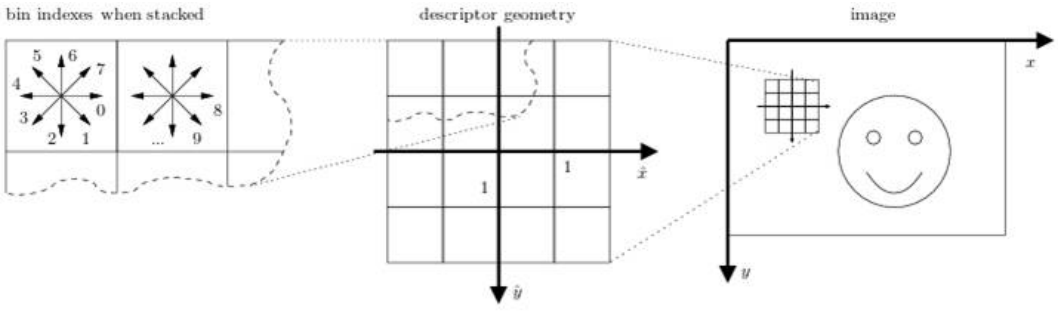


Figure 2.1: SIFT feature[1]

[1] figure cited from `http://www.vlfeat.org/overview/sift.html`

- **The Leung-Malik (LM) Filter Bank**

  Another popular way to capture texture is filter banks response. The LM [47] set is a multi scale, multi orientation filter bank with 48 filters. It consists of first and second derivatives of Gaussians at 6 orientations and 3 scales making a total of 36, 8 Laplacian of Gaussian (LOG) filters, and 4 Gaussians.

  In LM Small (LMS), the filters occur at basic scales $\sigma = \left\{1, \sqrt{2}, 2, 2\sqrt{2}\right\}$. The first and second derivative filters occur at the first three scales with an elongation factor of 3 (i.e. $\sigma_x = \sigma$ and $\sigma_y = 3\sigma_x$). The Gaussians occur at the four basic scales while the 8 LOG filters occur at σ and 3σ. For LM Large (LML), the filters occur at the basic scales $\sigma = \left\{\sqrt{2}, 2, 2\sqrt{2}, 4\right\}$. The illustration of filter bank is shown in figure 2.2.

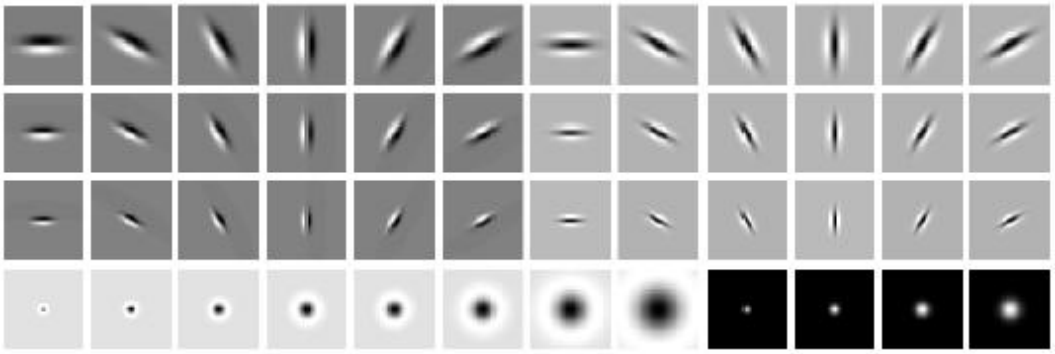


Figure 2.2: The Leung-Malik (LM) Filter Bank[2]

  In our work, the filter response vectors are use as descriptor of image texture, which are clustered later in the proposed algorithm.

- **Patch Representation**

  Though there has been ample evidence to suggest that filter banks can lead to good performance, however, many empirical results have shown that a multi-scale, multi-orientation large support filter bank is not necessary. Small image patches [72] can also lead to successful classification.

  Also, the supremacy of filter banks for texture synthesis was brought into question by the approach of Efros and Leung [20]. They demonstrated that superior synthesis results could be obtained using local pixel neighbourhoods directly, without resorting to large scale filter banks.

  Usually, the central pixel is discarded and only the neighborhood is used, feature vectors drawn from the set of N: i.e. the set of $N \times N$ image patches with the central pixel left

[2]figure cited from `http://www.robots.ox.ac.uk/~vgg/research/texclass/filters.html`

out. For example, in the case of a 3×3 image patch, only the 8 neighbors of every central pixel are used to form feature vectors. Figure 2.3 illustrates how to get a 3×3 image patch vector.

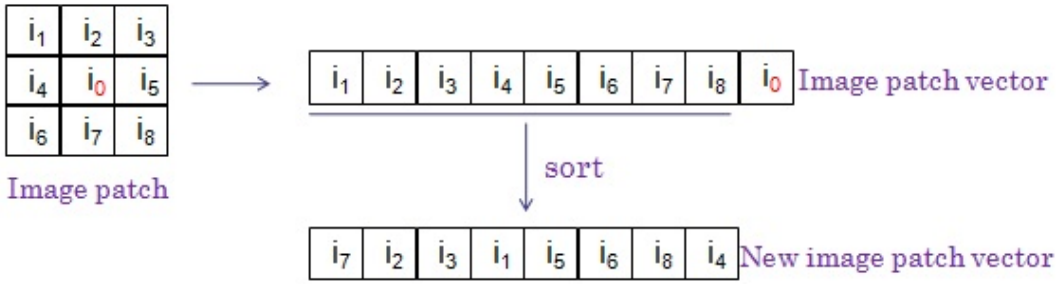


Figure 2.3: 3×3 image patch vector

In this thesis, these patch vectors are quantized to get "textons" that describe image texture.

Filter banks have a number of disadvantages compared to smaller image patches: first, the large support they require means that far fewer samples of a texture can be learnt from training images (there are many more 3×3 neighborhoods than 50×50 in an 100×100 image). Second, the large support is also detrimental in texture segmentation, where boundaries are localized less precisely due to filter support straddling region boundaries; A third disadvantage is that the blurring (e.g. Gaussian smoothing) in many filters means that fine local detail can be lost [73].

2.2.2 Color Features

- **RGB Space**

The RGB color model is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors. The name of the model comes from the initials of the three additive primary colors, red, green, and blue. The range of each color channel is [0, 255], so the total number of color in RGB space is 256^3. RGB color model is the most commonly used in computer vision and image processing.

Figure 2.4 shows the RGB color space, intuitively.

- **CIELAB Space**

[3] figure cited from `http://www.clear.rice.edu/elec301/Projects02/artSpy/color.html`

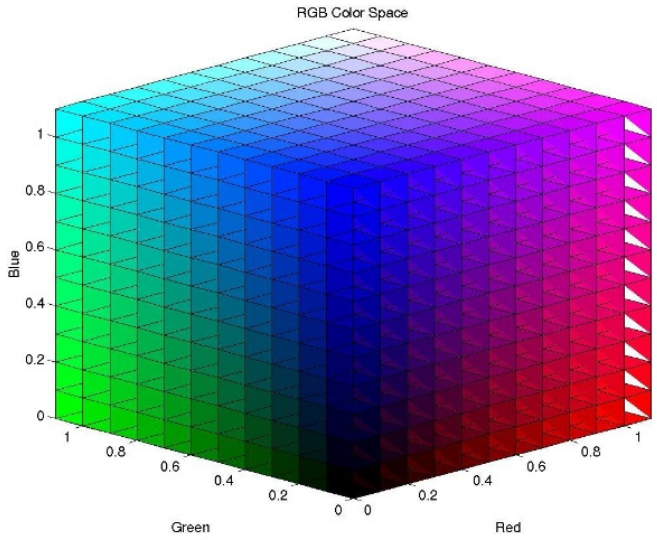


Figure 2.4: The RGB color[3]

CIELAB describes all the colors visible to the human eye and was created to serve as a device-independent model to be used as a reference. The three coordinates of CIELAB represent the lightness of the color ($L* = 0$ yields black and $L* = 100$ indicates diffuse white; specular white may be higher), its position between red/magenta and green ($a*$, negative values indicate green while positive values indicate magenta) and its position between yellow and blue ($b*$, negative values indicate blue and positive values indicate yellow).

CIELAB is a nonlinear transformation of RGB where the Euclidean distance between two colors is equal to their perceptual distances. Algorithms that process color images often produce better results in CIELAB[4], but it dosen't show significant advantage in our task.

2.3 Feature Selection

Feature selection (also known as subset selection) is a process commonly used in machine learning. It aim to select the best subset contains the least number of dimensions that most contribute to accuracy, therefore, reduce overfitting facilitate data visualization and data understanding, reduce the measurement and storage requirements, and training and utilization times.

Feature selection is also an important stage of image preprocessing. Many feature selection algorithms include ranking function as a principal or auxiliary selection mechanism because of its simplicity, scalability, and good empirical success.

General speaking, there are two approaches of feature selection [66]:

[4] http://www.mathworks.com/matlabcentral/fileexchange/24010

Forward selection Start with no variables and add them one by one, at each step adding the one that decreases the error the most, until any further addition does not significantly decrease the error.
Backward selection Start with all the variables and remove them one by one, at each step removing the one that decreases the error the most (or increases it only slightly), until any further removal increases the error significantly.

Figure 2.5 and 2.6 show common feature selection models and feature selection algorithms, respectively.

We develop our own feature selection approach based on how useful a feature is in separating a rectangular region from the rest of the image.

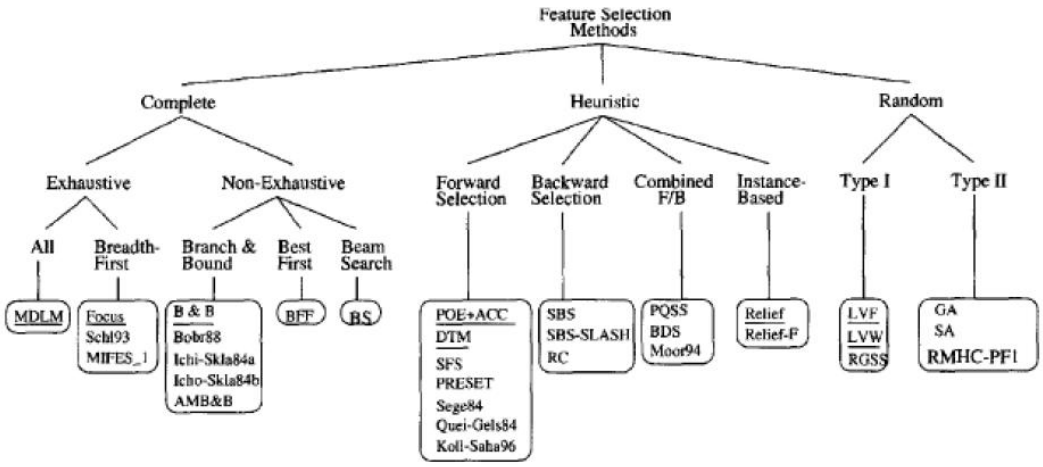


Figure 2.5: Feature selection models[5]

[5]figure cited from paper [66]

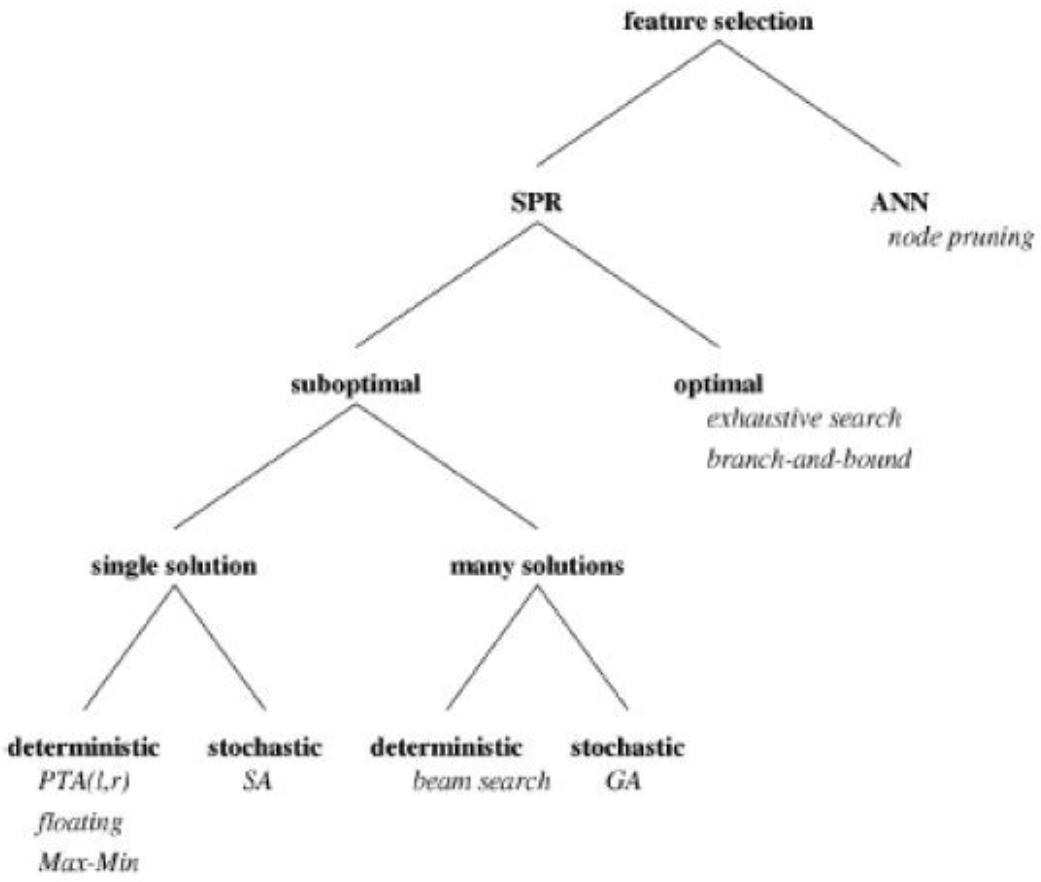


Figure 2.6: Feature selection algorithms[6]

[6]figure cited from paper [66]

Chapter 3

Overview of Energy Minimization Framework

The proposed algorithm implements automatic segmentation using the tool of graph cuts, which works in the energy optimization framework. The task of binary image segmentation is posed as a binary labeling problem, and solved by minimization a energy function. The resulting energy function can be globally and optimally solved with the max-flow/min-cut algorithm. Figure 3.1 below shows the way to solve image segmentation problem in the energy optimization framework.

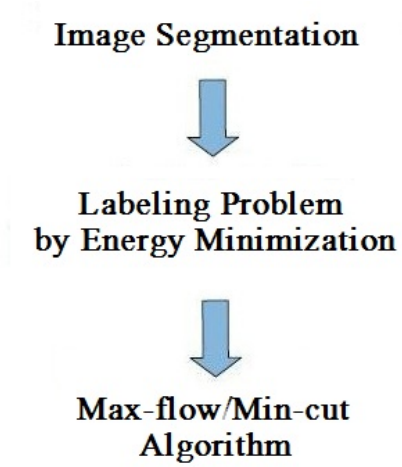


Figure 3.1: Solving image segmentation problem in energy minimization framework

In the optimization framework, there are two major steps, the formulation of energy function and the optimization of it. There are many advantages to the optimization based approach, though it is difficult to formulate appropriate energy function, and to optimize them are not trivial tasks, either. First, it provides a common framework which abstracts useful constraints from details of each particular problem. Once an energy function is formulated, the standard optimization approaches can be applied to solve it. Secondly, it enables us to apply our prior knowledge to solve the problem by encoding it in the energy function. Thus we can expect the

desired solution to have some nice global properties, such as the overall smoothness. Finally, the value of the energy function provides an effective way to evaluate the solution and can be used as a guide in the optimization algorithm [52].

Graph cuts is an optimization algorithm in energy minimization framework, which has been successfully used for a wide variety of vision problems, including image restoration [11, 12, 28, 34], stereo and motion [6, 11, 12, 33, 38, 41, 65, 50, 64], image synthesis [45], image segmentation [9], voxel occupancy [42], multicamera scene reconstruction [69], and medical imaging [7, 8, 39, 37]. The output of graph cuts algorithms is generally a solution with some interesting theoretical quality guarantee. In some cases, the solutions are the global minimum, even though in some case [11, 28, 33, 34, 50], it is a local minimum, but still within a known factor of the global minimum [12]. The experimental results produced by graph cuts algorithm are also quite good [43].

In this chapter, we will go over the energy optimization framework and introduce graph cuts algorithm.

3.1 From Segmentation to Labeling Problem

The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze [67]. Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. In computer vision, segmentation is the process of partitioning an image into multiple segments (sets of pixels, also known as superpixels). More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain visual characteristics.

A labeling problem is, roughly speaking, the task of assigning an explanatory "label" to each element in a set of observations [17]. Many classical clustering problems are also labeling problems because each data point is assigned a cluster label. Obviously, we can pose the task of image segmentation as a labeling problem, too.

To describe a labeling problem, one needs a set of observations (the data) and a set of possible explanations (the labels). A discrete labeling problem associates one discrete variable with each datum, and the goal is to find the best overall assignment to these variables (a "labeling") according to some criteria. In computer vision, the observations can be things like pixels in an image, salient points within an image, depth measurements from a range-scanner, or intensity measurements from CT/MRI. The labels are typically either semantic (car, pedestrian, street) or related to scene geometry (depth, orientation, shape, texture) [17].

The goal in a labeling problem is to construct a map $f : \mathcal{P} \mapsto \mathcal{L}$ that assigns to each element $p \in \mathcal{P}$ a corresponding label f_p. The set $\mathcal{P}$ indexes the observations, and the label set $\mathcal{L}$ indexes the explanations. In general, we have $|\mathcal{L}|^{|\mathcal{P}|}$ possible labelings (configurations of f), and we prefer one labeling over another based on some application-specific criteria [17].

In image segmentation, the map f is to assign labels f_p to image pixels such that $\forall p \in \mathcal{P}, f_p \in \mathcal{L}$, where f_p is the label assigned to pixel p, $\mathcal{L}$ is the set of possible labels, and $\mathcal{P} = \{1, 2, \ldots, P\}$ is the set of image pixels. If the label set for all pixels is the same, i.e. $\mathcal{L}$, then the set of possible labelings is $\mathcal{L} = L \times L \times \cdots \times L$. That is, the total number of different labelings is as huge as L^p.

For binary image segmentation, the label set $\mathcal{L} = \{0, 1\}$, where 0 and 1 stand for the background and the foreground, respectively. In this thesis, we will focus on binary labeling problem.

3.2 Energy Function Construction

3.2.1 Objective Function

The first step of optimization problem is to formulate an objective function, which maps any solution to a real number. In this way, we get a quantity measurement of how good the solution is.

An objective function $f : \mathcal{X} \mapsto \mathcal{Y}$ with $\mathcal{Y} \subseteq \mathbb{R}$ is a function which subjects to optimization [78]. The codomain $\mathcal{Y}$ of an objective function as well as its range must be a subset of the real numbers ($\mathcal{Y} \subseteq \mathbb{R}$).

Generally speaking, the objective function should be incorporated with the constraints that an acceptable solution must satisfy. A good objective function should assign high goodness score to solutions that match the constraints well. In computer vision, the objective function is usually referred to as energy function.

In particular, when designing an energy function, we should make sure it maps a good solution to low energy, and a bad solution will get a high energy. For example, in figure 3.2, the right image is the reshuffled image of the one on the left. Although the histograms of the left and right images are the same, the left image is more spatially coherent in its color distribution, so the energy of the solution on the left should be much lower than the energy of the solution on the right.

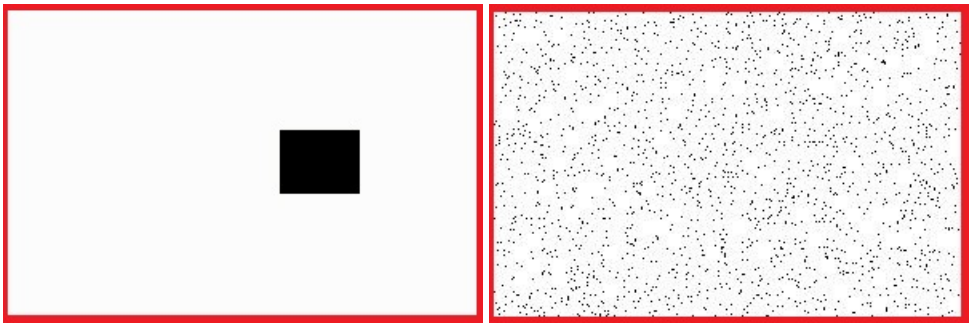


Figure 3.2: Left: original image, right: reshuffled image

There are two commonly used constraints in designing an energy function in image segmentation, the data constraint and the smoothness constraint. The data constrain comes from the observed data. It requires the solution to be close to the observed data.

Figure 3.3: Swan

For example, in swan image (figure 3.3), it is easy to come up with the constraint that the pixels that belong to the swan should have lighter colors, and the background pixels should have darker colors. Otherwise, they are violating the data constraints provided by the color information of the image [52].

The smoothness constraint usually encodes our preference for spatial coherence of the labels. Intuitively, most physical world objects are coherent in space. If a pixel in an image belongs to an object, the nearby pixels are very likely to belong to the same object. In other words, the smoothness knowledge tells us that both the background and the object should be spatially coherent. At the same time, we can encode more prior constraints in the energy functions, such as a preference for a particular shape [74].

3.3 Optimization Approaches

The second step of the energy minimization framework is to minimize the energy. In general, this is a very hard problem. The computational task of minimizing the energy is usually quite difficult as it usually requires minimizing a nonconvex function in a space with thousands of dimensions. If the functions have a very restricted form, they can be solved efficiently using dynamic programming [3]. However, researchers typically have needed to rely on general purpose optimization techniques such as simulated annealing [4], but it is very slow in practice [43].

Great effort has been made towards developing effective energy optimization algorithms. Among these approaches, graph cuts algorithm [8] is an effective tool for image segmentation. Kol-

mogorov and Zabih [43] describe the conditions on binary energy functions that can be optimized exactly with graph cuts, and the energy function satisfing the conditions can be solve in polynomial time.

In our case, we can optimize an energy function exactly and efficiently, using the results from [8] and [43].

3.3.1 Max-flow/Min-cut Algorithm

Max-flow/min-cut algorithm can be used to optimize the energy function when dealing with binary labeling problems. Here, we will give a brief summary of related definitions and how to find max-flow/min-cut of a graph.

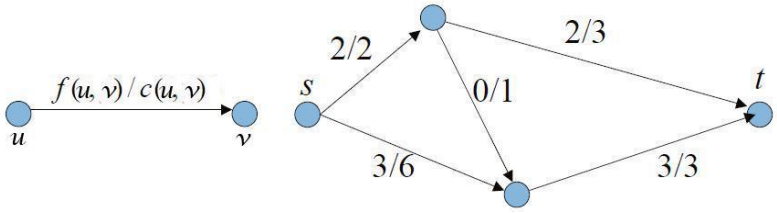


Figure 3.4: An example of flow[1] from "Introduction to Min-Cut/Max-Flow Algorithms"

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a network with $s, t \in \mathcal{V}$ being the source and the sink of $\mathcal{G}$, respectively, where $\mathcal{V}$ denotes vertices set and $\mathcal{E}$ is edges set. The capacity of an edge is a mapping $c : \mathcal{E} \mapsto \mathbb{R}^+$, denoted by $c(u, v)$. It represents the maximum amount of flow that can pass through an edge.

A flow is a mapping $f : \mathcal{E} \mapsto \mathbb{R}^+$, denoted by $f(u, v)$, subjects to the following two constraints:

1. $f_{uv} \leq c_{uv}$, for each (u, v) (capacity constraint: the flow of an edge cannot exceed its capacity)

2. $\sum_{(u,v)\in\mathcal{E}} f_{u,v} = \sum_{(u,v)\in\mathcal{E}} f_{v,u}$, for each $v \in \mathcal{V}\{s, t\}$ (conservation of flows: the sum of the flows entering a node must equal the sum of the flows exiting a node, except for the source and the sink nodes)

The value of flow is defined by $|f| = \sum_{v\in\mathcal{V}} f_{sv}$, where s is the source of $\mathcal{G}$. It represents the amount of flow passing from the source to the sink.

[1] Cited from Hong Chen (UCLA CIVS)'s ppt with a bit modification, `http://perso.telecom-paristech.fr/~tupin/cours/matim/articles/theorie_gra.pdf`

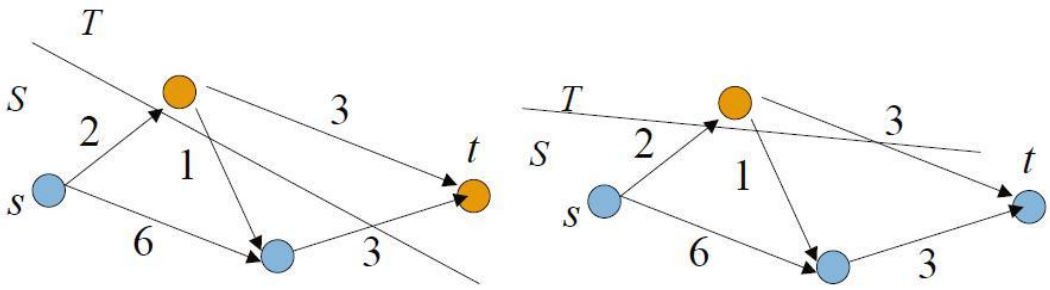


Figure 3.5: Left: a s-t cut, right: not a s-t cut

An $s-t$ cut $C = (\mathcal{S}, \mathcal{T})$ is a partition of $\mathcal{V}$ such that $s \in \mathcal{S}$ and $t \in \mathcal{T}$. The cut-set of C is the set $\{(u, v) \in \mathcal{E} \mid u \in \mathcal{S}, v \in \mathcal{T}\}$. Note that if the edges in the cut-set of C are removed, $|f| = 0$.

The capacity of an $s-t$ cut is defined by $c(\mathcal{S}, \mathcal{T}) = \sum_{(u,v)\in\mathcal{S}\times\mathcal{T}} c_{uv}$. For any $s-t$ cut and flow, the capacity of $s-t$ cut is the upper-bound of the flow across the $s-t$ cut.

The maximum flow problem is to maximize $|f|$, that is, to route as much flow as possible from s to t .

The minimum $s-t$ cut problem is minimizing $c(\mathcal{S}, \mathcal{T})$, that is, to determine $\mathcal{S}$ and $\mathcal{T}$ such that the capacity of the $s-t$ cut is minimal.

In optimization theory, max-flow/min-cut theorem is stated as follows:

Theorem [16]: If f is a flow function of a $s-t$ graph, then the follows statements are equivalent[2]:

A. f is a maximum flow

B. there is a $s-t$ cut that its capacity equals to the value of f

C. The residual graph contains no directed path from source to sink.

Max-flow/min-cut theorem means that in a flow network, the maximum amount of flow passing from the source to the sink is equal to the minimum edges' capacity in all possible way of removing edges. Removing these edges from the network should guarantee no flow can pass from the source to the sink.

There are two main approaches to solve max-flow/min-cut problem for the two-terminal graphs. In Cormen et al. [16], an augmenting path strategy is described to compute the minimum cut of a graph. Goldberg and Tarjan [26] propose an alternative approach named push-relabel to solve the minimum cut problem. Theoretically, the computational cost of minimum cut algorithms is a low order polynomial [52].

[2]for the proof see[16]

In [12], Boykov et al. developed new min-cut algorithms, two types of large moves (α – *expansion* and $\alpha-\beta$ *swap*), which can be used to solve multi-label problem. These algorithms find good approximate solutions by iteratively decreasing the energy on appropriate graphs. Experiment results show that the final solutions do not change significantly by varying the initial labelings. In practice, their algorithm is significantly faster than other standard move algorithms. As a special case of multi-label problem, in this work, we use the $\alpha-\beta$ *swap* algorithm bases on their implementation. For binary submodular energies [8, 43], the swap algorithm finds the exact optimum of the energy function.

3.4 Binary Image Segmentation with Graph Cuts

In this thesis, we deal with the problem of segmenting the foreground object from the background. It can be posed as a binary segmentation problem and can be addressed in the energy optimization framework. This was first done in the work of [8].

The main advantage of segmentation using graph cuts is that it incorporates the appearance of the foreground/background regions into data terms of energy function, meanwhile, constraints on the boundary is incorporated in smoothness term, and the energy function can be easily globally optimized. Here, we briefly review the graph cuts segmentation algorithm of [8].

First, we describe the basic terminology that pertains to graph cuts in binary image segmentation method.

A weighted graph (figure 3.6) $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is made up of vertices $\mathcal{V}$ and edges $\mathcal{E}$. There are two additional nodes: an object terminal (a source $\mathcal{S}$) and a background terminal (a sink $\mathcal{T}$). Therefore, $\mathcal{V} = \mathcal{P} \cup \{\mathcal{S}, \mathcal{T}\}$. The set of edges $\mathcal{E}$ consists of two types of undirected edges: *n-links* (neighborhood links) and *t-links* (terminal links). Each pixel p has two *t-links* $\{p, \mathcal{S}\}$ and $\{p, \mathcal{T}\}$ connecting it to each terminal. Each pair of neighboring pixels $\{p, q\}$ in $\mathcal{N}$ is connected by an *n-link*. Without introducing any ambiguity, an *n-link* connecting a pair of neighbors p and q will be denoted by $\{p, q\}$. Therefore, $E = \mathcal{N} \bigcup_{p \in P} \{\{p, \mathcal{S}\}, \{p, \mathcal{T}\}\}$. Each edge $e \in \mathcal{E}$ has a non-negative cost w_e.

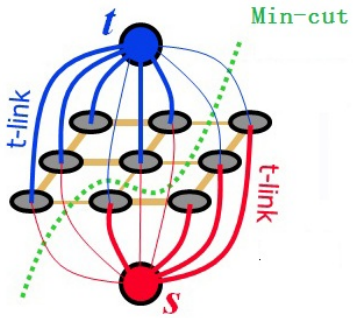


Figure 3.6: An example of graph cuts

A cut is a subset of edges $C \subset \mathcal{E}$ such that the terminals become separated on the induced graph $G(C) = \langle \mathcal{V}, \mathcal{E} \setminus C \rangle$. The cost of the cut C is the sum of cut edge weights: $|C| = \sum_{e \in C} w_e$.

The max-flow/min-cut algorithm [22] can be used to find the minimum cut (the cut with smallest cost) in polynomial time. The segmentation prosess is illustrated as follows (figure 3.7):

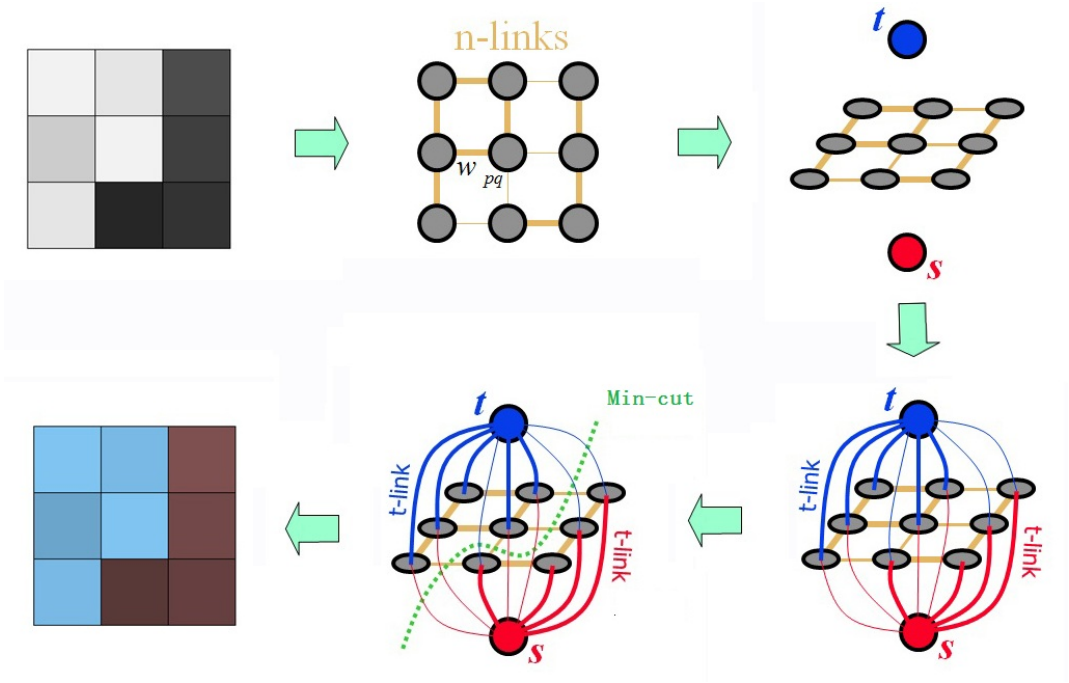


Figure 3.7: An example of graph cuts

As mentioned before, image segmentation is posed as a labeling problem, for binary segmentation, each pixel in the image has to be assigned a label either foreground or background.

To illustrate binary labeling problem, let $\mathcal{P}$ be the set of all pixels in the image, and let $\mathcal{N}$ be the standard 4- or 8-connected neighborhood system on $\mathcal{P}$, pixel pairs $\{p, q\}$ are neighbour pixels in $\mathcal{P}$. Let $f_p \in \mathcal{L}$ be the label assigned to pixel p. In binary labeling problem, the label set $\mathcal{L} = \{0, 1\}$, where 0 and 1 stand for the background and the foreground, respectively. $f = \left\{ f_p \mid p \in \mathcal{P} \right\}$ denotes the collection of all label assignments.

Energies which can be optimized through graphs cuts usually have the following form:

$$E(f) = E_{Data}(f) + E_{Smooth}(f) \tag{3.1}$$

where,

$$E_{Data}(f) = \sum_{p \in P} D_p\left(f_p\right) \tag{3.2}$$

$$E_{Smooth}(f) = \sum_{\{p,q\} \in \mathcal{N}} V_{pq}\left(f_p, f_q\right) \tag{3.3}$$

In Eq.(3.1), the first term is called the data term (definded in Eq.(3.2)) which incorporates regional constraints. Specifically, it measures how well pixels fit into the foreground or background models. $D_p\left(f_p\right)$ is the penalty for assigning label f_p to pixel p. The more likely $D_p\left(f_p\right)$ is for p, the smaller is $D_p\left(f_p\right)$.

$E_{Smooth}\left(f\right)$ is called the smoothness term (definded in Eq.(3.3)), measures the extent to which f is not smooth. $V_{pq}\left(f_p, f_q\right)$ is the penalty for assigning labels f_p and f_q to neighboring pixels. Most nearby pixels are expected to have the same label, therefore there is no penalty if neighboring pixels have the same label, and a penalty otherwise.

Kolmogorov and Zabih [43] prove that if binary energy function is regular[3], it can be optimized exactly with a graph cut.

The energy defined in Eq. 3.1 has $|\mathcal{P}|$ variables, and it can be viewed as a sum of several two-variable $V_{pq}\left(f_p, f_q\right)$ and single-variable $D_p\left(f_p\right)$ functions. If all of these functions are regular, then $E\left(f\right)$ is regular, see [43]. To check the regularity of the energy functions, we only have to examine the interaction penalty $V_{pq}\left(f_p, f_q\right)$. According to [43], all the one variable functions are regular. Therefore, if $V_{pq}\left(f_p, f_q\right)$ satisfies $V_{pq}\left(0,0\right) + V_{pq}\left(1,1\right) \leq V_{pq}\left(1,0\right) + V_{pq}\left(0,1\right)$, then $E\left(f\right)$ is regular, and it can be optimized by graph cuts.

In our work, the energy function is formulated according to the basic form. The details are described in chapter 6.

[3]A function of two variables $E\left(x_1, x_2\right)$ is regular on the variable value set $\{0,1\}$, if it satisfies: $E\left(0,0\right) + E\left(1,1\right) \leq E\left(0,1\right) + E\left(1,0\right)$.

Chapter 4

Related Work

Image segmentation is a fundamental problem in computer vision, and it has experienced tremendous growth in the past 10 years. From the extent of user dependence, segmentation algorithms can be categorized into automatic methods and interactive methods. In practice, automatic and interactive methods are often used together to improve the segmentation results. For example, some automatic segmentation methods may require interaction for setting initial parameters and some interactive methods may start with the results from automatic segmentation as an initial segmentation.

In this chapter, we will give a general summary of the related methods and a brief analysis on them. In particular, since our proposed method target to find an effect automatic initialization for binary image segmentation using graph cuts, we will also talk about some useful cues for starting graph cuts automatically.

4.1 Interactive Segmentation Methods

Interactive image segmentation becomes more and more popular in recent years. Because interactive segmentation gives the user the means to incorporate his knowledge into the segmentation process, it often makes the segmentation result more satisfying or to reduce the computing time.

For "interaction", the user is usually required to click a few "seeds" on the desired object, or near its border, and let the algorithm complete the rest segmentation task. At the same time, the user can edit the results by adding, removing or moving control points (seeds), and re-execute the segmentation algorithm to update the results.

Some popular interactive methods are introduced in this section.

4.1.1 Interactive Graph Cuts

The method of interactive graph cuts for binary image segmentation was first proposed by Boykov and Jolly [8], which is the most important inspiration of this work.

In [8], the user imposes certain hard constraints for segmentation in the way of setting certain pixels (seeds) that absolutely have to be part of the object, and certain pixels that have to be part of the background. Intuitively, these hard constraints provide clues on what the user intends to segment.

Obviously, the hard constraints by themselves are not enough to obtain a good segmentation. The general energy function mentioned in chapter 3 can be viewed as a soft constraint that incorporates both region and boundary properties for segmentation. In the work of [8], they combine the hard constraints with the soft constraints.

Consider an arbitrary set of data elements $\mathcal{P}$ and some neighborhood system represented by a set $\mathcal{N}$ of all unordered pairs $\{p, q\}$, which are neighboring elements in $\mathcal{P}$. In particular, $\mathcal{P}$ can contain pixels (or voxels) in a 2D (or 3D) grid, and $\mathcal{N}$ can contain all unordered pairs of neighboring pixels (voxels) under a standard 8- (or 26-) neighborhood system. Let $f = (f_1, \ldots, f_p, \ldots, f_{|\mathcal{P}|})$ be a binary vector whose components f_p that specify assignments to pixels p in $\mathcal{P}$. Each f_p can be either "obj" or "bkg" (abbreviations of "object" and "background"). In other words, vector f defines a segmentation. Then, the soft constraints that imposed on boundary and region properties of f are described by the energy function $E(f)$:

$$E(f) = \lambda \cdot E_{Data}(f) + E_{Smooth}(f) \tag{4.1}$$

where,

$$E_{Data}(f) = \sum_{p \in P} D_p\left(f_p\right) \tag{4.2}$$

$$E_{Smooth}(f) = \sum_{\{p,q\} \in \mathcal{N}} W_{pq} \cdot \delta\left(f_p, f_q\right) \tag{4.3}$$

$$\delta\left(f_p, f_q\right) = \begin{cases} 1 & if \quad f_p \neq f_q, \\ 0 & otherwise \end{cases}$$

The coefficient $\lambda \geq 0$ in (4.1) specifies a relative importance of the region properties (data) term $E_{Data}(f)$ versus the boundary properties (smoothness) term $E_{Smooth}(f)$. The regional term assumes that the the individual penalties for assigning pixel p to object and background, correspondingly D_p("obj") and D_p("bkg").

The boundary term comprises the "boundary" properties of segmentation f. Coefficient $W_{pq} \geq 0$ should be interpreted as a penalty for a discontinuity between p and q. Normally, W_{pq} is large when pixels p and q are similar (e.g. in their intensity) and W_{pq} is close to zero when the two are very different. The penalty W_{pq} can also decrease as a function of distance between p and q.

The graph used in [8] has the same structure with the general one introduced in the previous chapter, but incorporated with the user input hard constraint (seeds) on corresponding *s-link* and *t-link*. Table 4.1 gives weights of edges in $\mathcal{E}$.

$$K = 1 + \max_{p \in \mathcal{P}} \sum_{\{p,q\} \in \mathcal{N}} W_{pq}$$

edge	weight (cost)	for
$\{p,q\}$	W_{pq}	$\{p,q\} \in \mathcal{N}$
$\{p,\mathcal{S}\}$	$\lambda \cdot D_p(\text{“}bkg\text{”})$	$p \in \mathcal{P}, p \notin \mathcal{O} \cup \mathcal{B}$
	K	$p \in \mathcal{O}$
	0	$p \in \mathcal{B}$
$\{p,\mathcal{T}\}$	$\lambda \cdot D_p(\text{“}obj\text{”})$	$p \in \mathcal{P}, p \notin \mathcal{O} \cup \mathcal{B}$
	0	$p \in \mathcal{O}$
	K	$p \in \mathcal{B}$

Table 4.1: Weights of edges in $\mathcal{E}$

The segmenting process is illustrated in figure 4.1. In the graph, the hard constraints indicate that some pixels were marked as internal and some as external for the given object of interest. The subsets of marked pixels will be referred to as object and background seeds. The segmentation boundary can be anywhere but it has to separate the object seeds from the background seeds. In particular, the seeds can be loosely positioned inside the object and background regions. Using max-flow/min-cut algorithm, the target image can be segmented automatically by computing a global optimum among all segmentations satisfying the hard constraints.

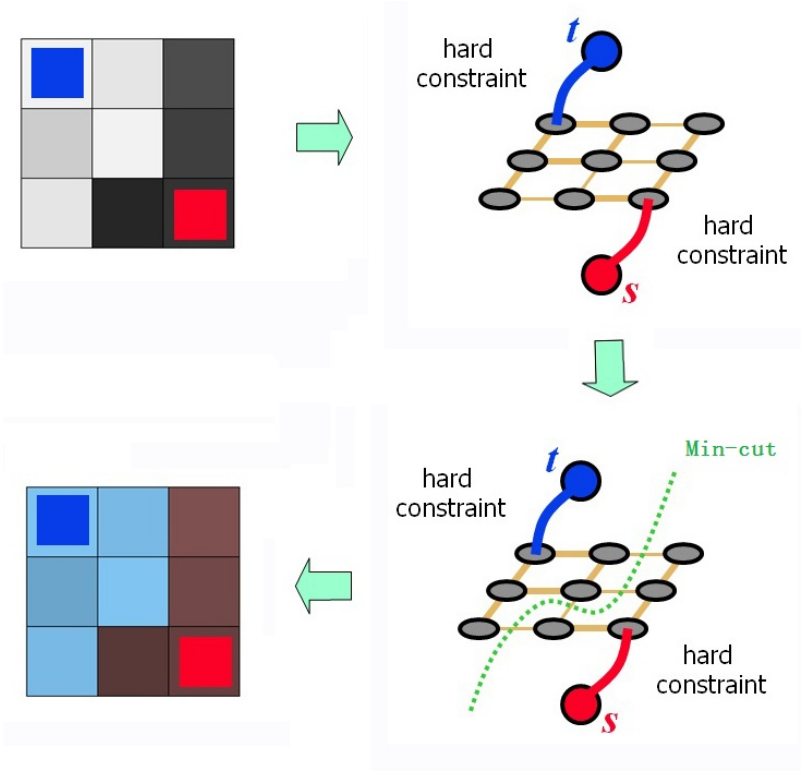


Figure 4.1: Segmentation with hard constraints

This segmentation technique is quite stable and normally produces the same results regardless of particular seed positioning within the same image object. Meanwhile, new globally optimal

segmentation can be very efficiently recomputed when the user adds or removes any hard constraints (seeds). This allows the user to get any desired segmentation results quickly via very intuitive interactions. The process of interactive segmentation is illustrated in figure 4.2.

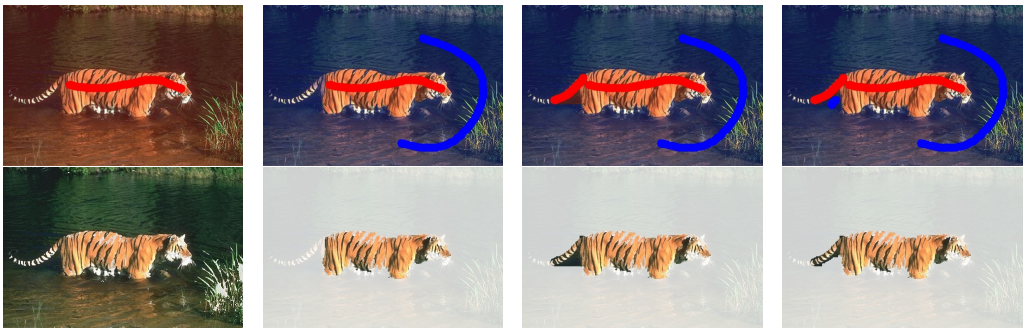


Figure 4.2: An example of segmentation using interactive graph cuts

Figure 4.2 shows how graph cuts works interactively. The first row illustrates the original images with user interactions: red (foreground brush), blue (background brush). The second row displays the according segmentation results. The degree of user interaction increases from left to right. From the marked pixels, the appearance models for the foreground and the background regions are constructed and used in the $D_p\left(f_p\right)$ terms. For the smoothness term, in this example, image gradient Potts model is used. The user marked seeds are assigned such that $D_p\left(f_p\right)$ make it impossible for them to take a label other than the one indicated by the user. That is if the user marks p as the background pixel, then the cost of assigning it to the foreground is infinity, and the cost of assigning it to the background is 0. This insures that the final segmentation is consistent with user marked seeds.

4.1.2 Grabcut

Grabcut [62] is another inspiration of our method. Actually, it is an iterative graph cuts method. During the process of iteration, the energy can be reduced step by step, hence, it can be used to refine the binary segmentation.

The energy function defined in grabcut is as follows:

$$E\left(f\right) = \sum_{p \in P} D_p\left(f_p\right) + \sum_{\{p,q\} \in N} V_{pq}\left(f_p, f_q\right) \tag{4.4}$$

where,

$$D_p\left(f_p\right) = -logPr\left(I_p \mid f_p\right) - log\pi\left(f_p\right) \tag{4.5}$$

$$V_{pq}\left(f_p, f_q\right) = \gamma \sum_{\{p,q\} \in N} \delta\left(f_p, f_q\right) exp\left(-\beta dis\left(p,q\right)\right) \tag{4.6}$$

and,

$$\beta = \left(2\left\langle\left(I_p - I_q\right)^2\right\rangle\right)^{-1} \tag{4.7}$$

$Pr(\cdot)$ is a Gaussian probability distribution, and $\pi(\cdot)$ stand for mixture weighting coefficients. I_p and f_p are intensity and label of p, respectively. The distance here is Euclidean distance in color space. $\langle\cdot\rangle$ denotes expectation over an image sample, and γ is a constant.

In grabcut algorithm, the initial user input is a rectangle placed loosely around the foreground object (see figure 4.3). The inside is assumed to be mostly the foreground, while the outside of the rectangle is assumed to be the background, and the labels of these pixels will be fixed during the entire segmentation process. From this rough initialization, initial foreground/background models are constructed and binary graph cuts are applied to get the foreground/background segmentation. From this segmentation, the foreground/background models are updated and binary optimization is performed once again. This process is iterated until convergence, in the expectation maximization (EM) [18] manner.

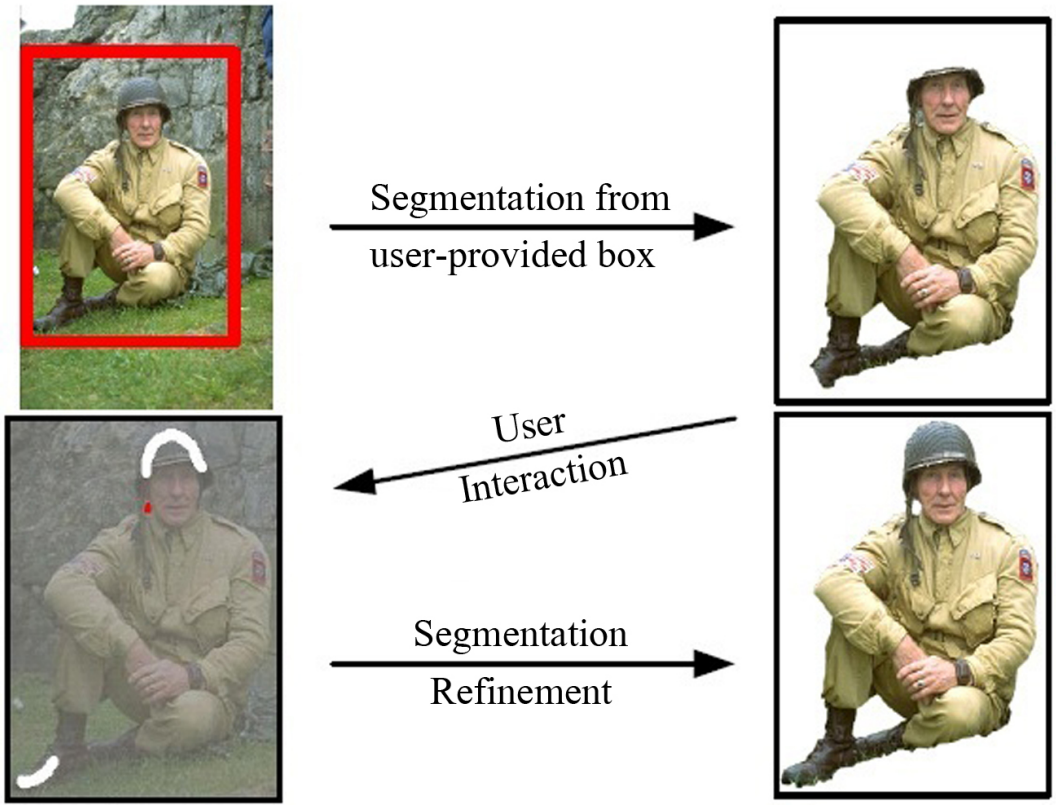


Figure 4.3: Workflow of grabcut method[1]

The energy function in the grabcut framework optimizes both over the appearance of the foreground/background, as well as the segmentation of the image into the new foreground/background parts. Although this problem is no longer optimizable exactly with graph cuts, in [62] they show that the algorithm is guaranteed to converge at least to a local minimum.

[1] figure cited from paper [62] with a bit modification

The aim of this thesis is to make grabcut from semi-automatic segmentation algorithm to a fully-automatic one. To be precise, we will try to find the initial "rectangle" of the grabcut algorithm automatically, instead of requiring the user to provide it. After that, we follow the steps of the grabcut algorithm. That is we will run iterative binary graph cuts segmentation to update the foreground/background models until convergence is achieved.

4.1.3 Other Approaches

Besides interactive graph cuts and grabcut, there are other popular interative segmentation methods, just to list a few, here.

- **Lazy Snapping**

  Lazy snapping [49] is another popular graph based segmentation method, which enables the user to edit the result by border editing to get better results.

  Actually, lazy snapping algorithm uses a similar graph construction as in graph cuts. The graph vertices in lazy snapping are at a course scale: superpixels (over-segment from watershed algorithm), and then editing border to fix errors on pixel scale. Figure 4.4 below shows the superpixels (left) and border editing (right) in lazy snapping algorithm.

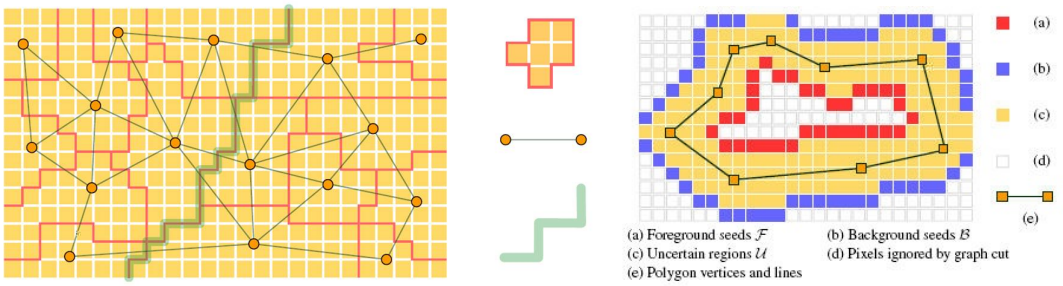


Figure 4.4: Superpixels (left) and border editing (right) in lazy snapping algorithm[2]

- **Snakes**

  Contour based interactive segmentation "snakes" was suggested by Kass et al. [36] in 1987. The model treats the desired contour as a time evolving curve, and the segmentation process is to iteratively reduce the defined energy until convergence, during which the initialized contours actively deform themselves. Convergence is achieved when reaching a balance between the "external" powers that attracts the contour to its place and the "internal" powers which keeps it smooth. The example of segmenting using snakes is illustrated in figure 4.5.

[2]Figure cited from paper [49]

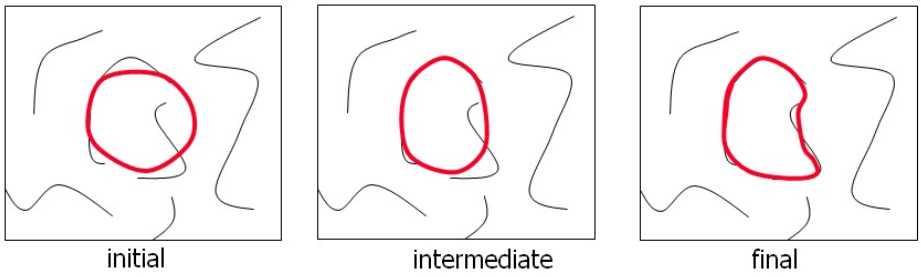


Figure 4.5: Deformable contour of snakes algorithm

"Snakes" can only find the local minimum of the energy function, so it requires the initialized contour not too far from the object border.

- **Livewire**

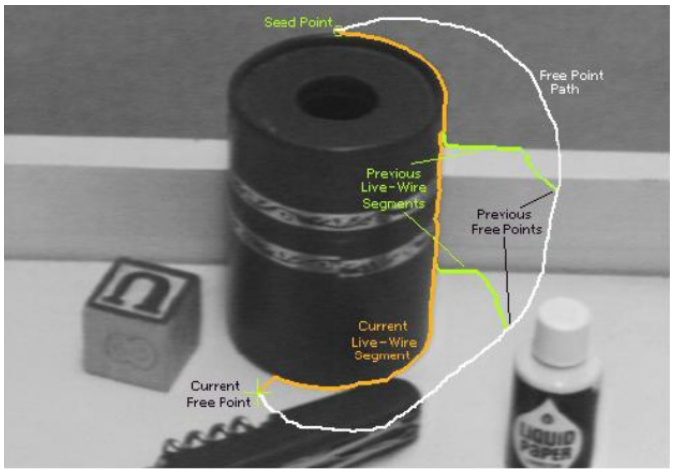


Figure 4.6: Example of livewire algorithm[3]

Livewire is a shortest path based method, which can reach the global minimum between two user inputted points. It was developed independently by two groups: Mortensen & Barret [58] and Udupa et. al. [21]. Livewire algorithm expects that the user input points should be exactly on the object boundary, which is a strong requirement, while, the graph based segmentation algorithm can perform well as long as there are certain seeds inside and outside the object.

In figure 4.6, the user-cursor path is shown in white. The corresponding detected boundary is shown in orange. The current livewire path in an intermediate point is drawn from "free point" through "boundary point" (green segment) to "start point" (orange segment).

[3]Figure cited from paper [58]

4.2 Automatic Segmentation Methods

Although automatic segmentation is a desirable goal in computer vision, we have to admit that fully automatic segmentation is still an open problem.

4.2.1 Saliency Segmentation

Saliency segmentation [57] is another work related to our approach, we both try to explore an effective initialization to graph cuts, and to a large extent, we are similar in using image features such as color and texture. The main difference between the work in [57] and ours is that the former learns a saliency map that is used to initialize data terms from a manually labeled dataset. In our proposed method, we find foreground region only based on the input image. Other difference is that [57] combines color and texture feature to produce a saliency map. Our approach is to find good features that are useful for distinguishing the foreground from the background.

In [57], the input image is first oversegmented into superpixels. Next, a trained classifier is used to output a confidence value, independently for each superpixel, and the confidence map is also taken to be the saliency map. Then the saliency map is partitioned into classifications of salient object and background. The classifier results will be further refined using iterative graph cuts (see figure 4.7).

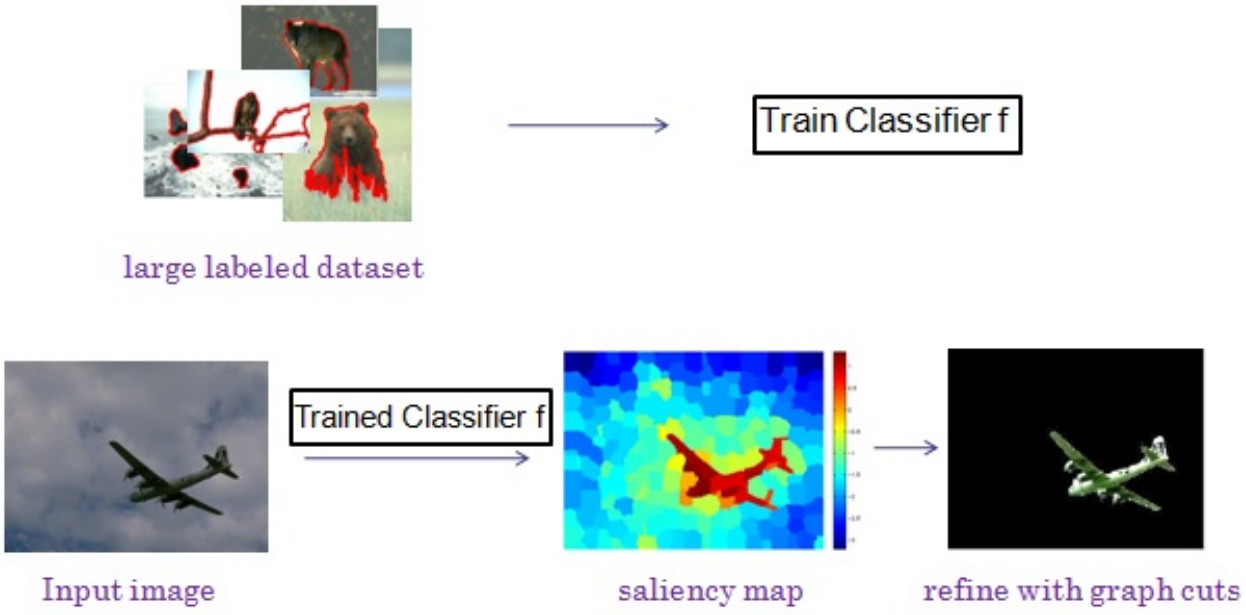


Figure 4.7: Saliency segmentation workflow

The energy function in [57] has the basic form defined in [8], and it can be minimized using graph cuts. In particular, the smoothness term is as follow:

$$V_{pq}\left(f_p, f_q\right) \propto exp\left(-\triangle I^2/2\sigma^2\right) \cdot \delta\left(f_p, f_q\right) \tag{4.8}$$

where, $\triangle I$ denotes the intensity difference of two neighboring pixels, σ^2 is the variance of intensity difference of pixels.

The data term consists of two parts,

$$D_p(1) = -lnPr\left(C_p|1\right) - \gamma \cdot ln\left(m_p\right) \tag{4.9}$$

$$D_p(0) = -lnPr\left(C_p|0\right) - \gamma \cdot ln\left(1 - m_p\right) \tag{4.10}$$

where, 1 is the salient object and 0 is the background, C_p is the quantized color of pixel p, m_p is the confidence of pixel p, and constant γ controls the relative importance of the two parts.

Although saliency segmentation implements automatic segmentation seemingly, it needs large training dataset as pre-condition, which is usually collected and labeled by a human.

4.2.2 Other Automatic Segmentation Methods

There are very few truly automatic segmentation methods that can serve the goal well. They either over-segment the image into small pieces, and then merge by certain techques such as, watershed [75](figure 4.8), meanshift [23](figure 4.9), splitting and merging [30](figure 4.10), normalized cuts [68](figure 4.11), or simply choose a threshold [27](figure 4.12), which is likely to "leak" on the weak boundary.

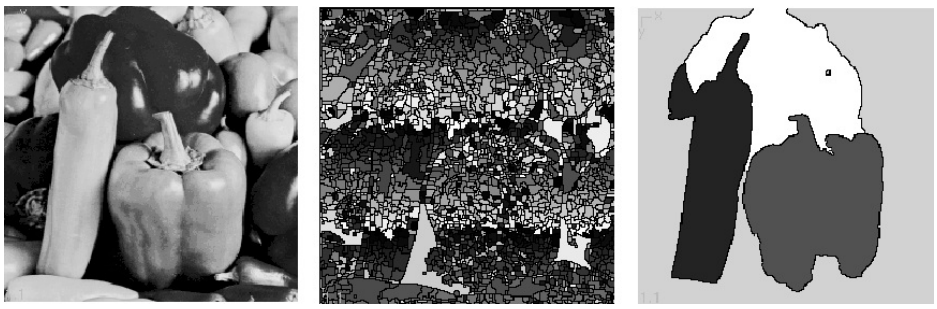


Figure 4.8: Left: original image, middle: row watershed segmentation, right: watershed segmentation using region marks to control over segmentation[4]

[4]Figure cited from `http://www.engineering.uiowa.edu/~dip/LECTURE/Segmentation3.html#splitmerge`

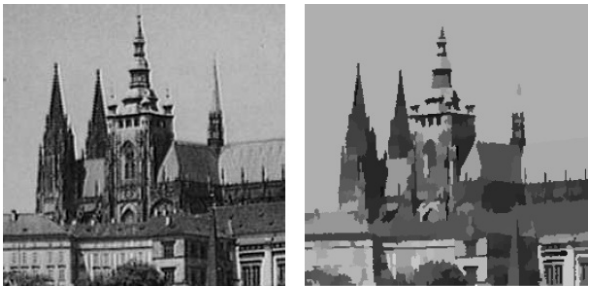


Figure 4.9: Left: original image, right: mean shift result[5]

Figure 4.10: Left: original image, middle: image after quad spliding, right: image after merging[6]

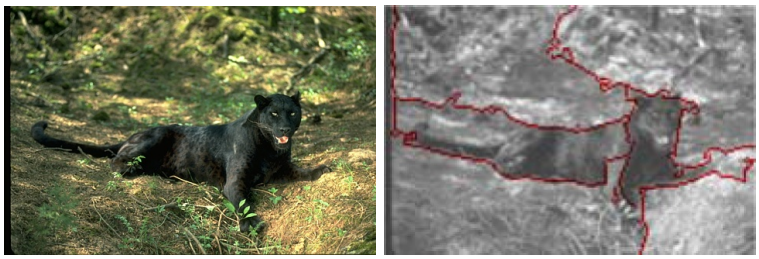


Figure 4.11: Left: original image, right: normalized cut result[7]

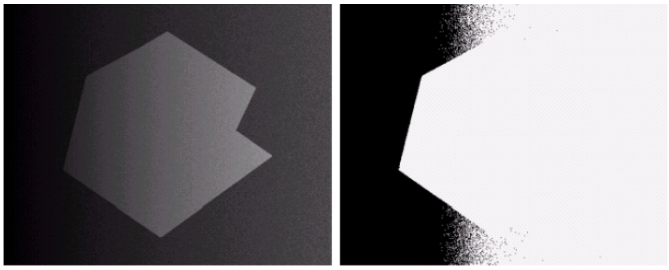


Figure 4.12: Left: original image, right: threshoding result[8]

[5]Figure cited from `http://cmp.felk.cvut.cz/cmp/courses/33DZOzima2007/slidy/meanShiftSeg.pdf`

[6]Figure cited from `http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MARBLE/medium/segment/split.htm`

[7]Figure cited from [61]

4.2.3 Useful Cues for Foreground Initialization

In this thesis, we aim to explore a relatively effective method to start graph cuts, therefore, to make it independent from user guidance. By this indirect way, the goal of automatic segmentation might be fulfilled. In this part, we will focus on automatic initialization of graph cuts, and divide them into certain useful cues. Here, we go over some typical methods.

- **Machine learning techniques**

  Machine learning techniques are often used to automatically find initial solution for graph cuts. For example, Fukuda et al. [29] use AdaBoost to automatically find the approximate location of the object. A model of saliency-based visual attention is integrated with graph cuts since some object regions appear to increase visual attention more than background regions. In this way, the saliency map is used as a prior probability of the object model (spatial information) (see figure 4.13).

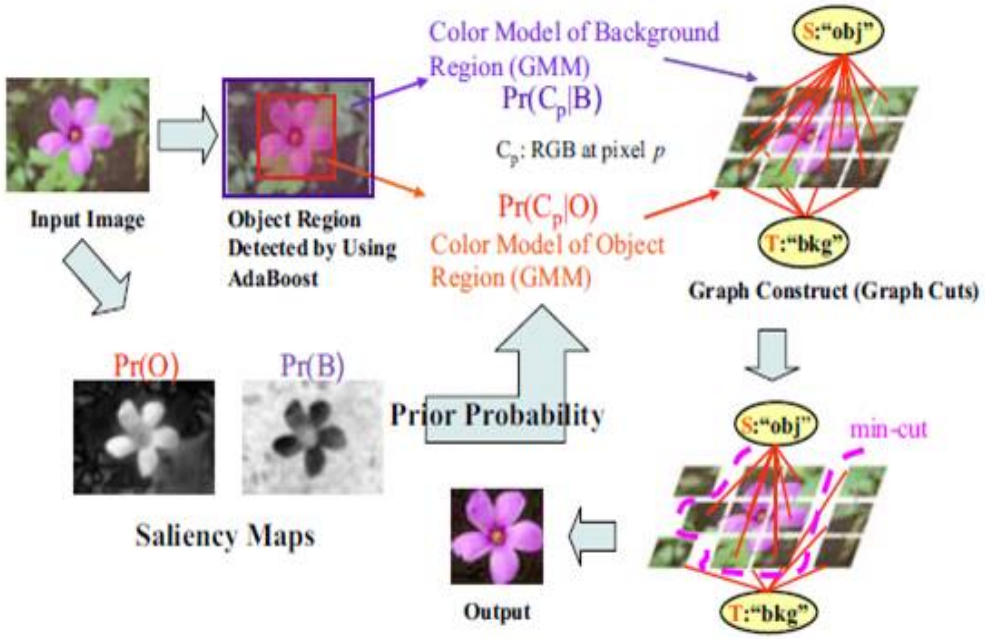


Figure 4.13: Graph cuts based on saliency maps and AdaBoost[9]

We must notice that machine learning approache based on manually labeled dataset for automatic segmentation has a significant drawback, it can only be used to find a specific foreground object/objects that were present in the training dataset. Unfamiliar objects cannot be handled. Obviously, it is not a general method.

[8]Figure cited from book [27]
[9]Figure cited from paper [29]

- **Shape prior**

  Shape prior is also a useful help to automatic object segmentation, which incorporates the shape knowledge into energy function, therefore, it might lead to better results.

  In the work [74], a star shape (figure 4.14) is defined, and a pairwise shape constraint term S_{pq} (Eq.4.11) is designed.

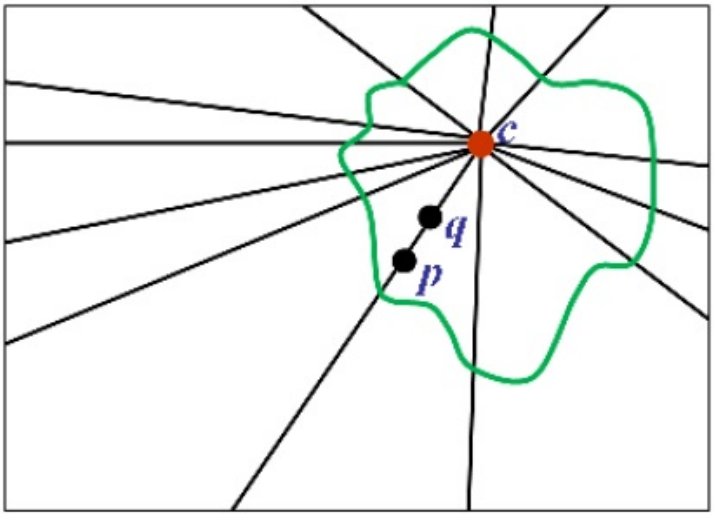


Figure 4.14: An example of a star shape, the center of the star shape is marked with a red dot c, and the star shape is outlined in green. Some of the straight lines passing through c are shown in black[10].

$$S_{pq}\left(f_p, f_q\right) = \begin{cases} 0 & if \quad f_p = f_q, \\ \infty & if \quad f_p = 1 \quad and \quad f_q = 0, \\ \beta & if \quad f_p = 0 \quad and \quad f_q = 1 \end{cases} \tag{4.11}$$

For any point p inside the object, Eq.4.11 insures that every single point q on the straight line connecting c and p is also inside the object. β is a parameter finding by binary search.

$$E\left(f\right) = \sum_{p \in P} D_p\left(f_p\right) + \lambda \sum_{(p,q) \in N} V_{pq}\left(f_p, f_q\right) + \sum_{(p,q) \in N} S_{pq}\left(f_p, f_q\right) \tag{4.12}$$

Eq. 4.12 is the energy function incorporated with star shape prior, and the global minimum can be guaranteed by max-flow/min-cut algorithm.

Though in [74], the star centers are provided by the user, star prior is not based on a shape of a specific object class, but rather on simple geometric properties of the objects. In certain restricted domains, such as in medical imaging, it may be possible to calculate the center automatically.

[10] Figures cited from paper [74]

- **Closely-related images**

  Closely-related images supply another cue of assisting graph cuts. For instance, in paper [63], the information from two common material shared images is compared (figure 4.15). This is implemented by jointly cosegmentation the image pair using a proper MRF coherence prior and a histogram matching cost.

Figure 4.15: Image pair for co-segmentation

  In order to arrive at an objective function for cosegmentation, the proposed method in [63] begins with setting out a generative model for an image pair, and then evaluate the hypothesis that the images share common material. The recovered cosegmentation will be that pair of regions, one from each image.

  The author choose a generative model for the foreground histograms as a whole, rather than individual pixels. A further Ising prior on segmentations, gated by image contrast, encourages smooth boundaries.

  Since the MRF term is an essential part of the energy, it is desirable to use the well-established technique for binary MRFs-min-cut/max-flow algorithm [10]. The global term of proposed algorithm can be solved by using submodular-supermodular procedure [59].

  There are also other cases of closely-related images segmentation, for instance, [12] makes a good use of spatial information hidden in stereo pair, while [46] works on image sequences.

- **Specific object**

  For a specific object, it is possible to come up with a automatic intialization. For example, in paper [55], the target object is already known as liver, therefore, an adaptive threshold method is proposed to automatically provide the initialization.

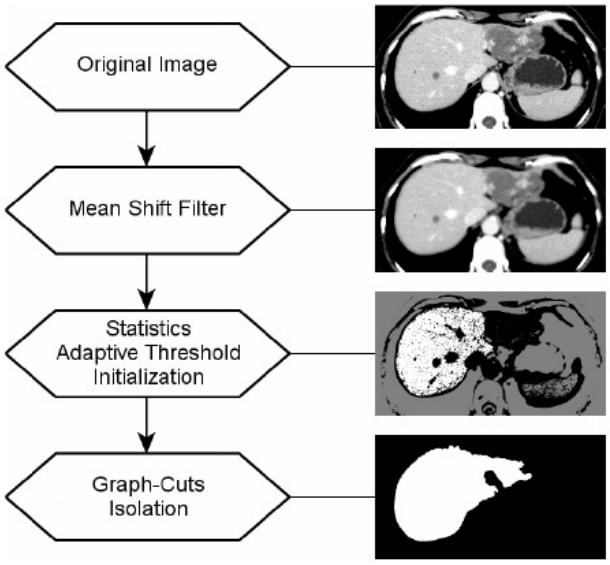


Figure 4.16: Flow diagram[11] of algorithm in paper [55]

4.3 The Weak Points of the Existing Methods

Although interactive methods can improve the accuracy by incorporating prior knowledge from the user, in some practical applications, such as handling a large number of images, they can be laborious and time consuming. If we can find grabcut (graph cuts) an automatic starting point, a big jump will be made in automatic segmentation method, at the same time, it can be applied in more fields.

On the other hand, pure bottom up approaches such as mean shift [23] or normalized cuts [68] can only segment objects that are highly salient with respect to the background [5]. Intensity-based methods, such as (local) binary thresholding [27] or region growing [27], tend to produce discontinuous contours and "leakage". Active contour models, such as snakes [36] or level-set [60] methods, are sensitive to initialization and are of limited use in areas of low gradient [44].

About the automatic initialization methods themselves, we agree that the cues mentioned in previous section do help automatic segmentation, but the additional information makes these approaches restricted to a particular object class/classes or computationally demanding.

So in this thesis, we will work on single natural images without user interaction or labeled training data. In other words, we can only mine the input image itself, and take use of low-level visual cues, such as intensity, color and texture, that is exactly where the challenge and the beauty of our work lies in.

[11] figure cited from paper [55]

Chapter 5

Automatic Foreground Detection

As mentioned before, our work is within the framework of energy minimization, and our aim is to find a good initialization for graph cuts. In this chapter, the algorithm of finding the object contained rectangle is described. This process can be divided into three main steps: a) feature extraction and clustering, b) feature selection and c) search for a rectangle containing the object. We will explain each step of the algorithm in details.

Before going into the details, let's first make the basic assumptions clear.

5.1 Basic Assumptions

Our work is based on the hypothesis that there exists an unknown feature space in which the feature vectors from foreground and background have detectable dissimilarity. The assumption is reasonable for a large majority of images. This is because an object is an entity different from the background and its appearance characteristics were developed independently from various backgrounds, see images in figure 5.1.

Figure 5.1: Images of the same object in different backgrounds

Another assumption of this thesis is that the major part of the object is not located on the boundary of the input image. We make this assumption purely for convenience of searching and modifying the binary labels. If this assumption is violated, it is easy to fix by enlarging the central part of the image or padding the boundary of the image with thick enough border.

Figure 5.2: Major part of foreground on the image boundary

5.2 Feature Extraction and Clustering

The goal of binary image segmentation is to give each pixel a label to show which part it belongs to, namely, foreground or background. In order to fulfill this task, we need enough information to help to group the pixels in the target image. The more distinguishable information there is, the easier it is to segment. Of course, we should not expect that all the useful information can be found, but we hope to find a sufficient useful subset.

The first step of our work is to get the characteristics of every pixel, in other words, to find a proper "feature description". In this work, two kinds of features (texture and color) are used to represent pixel characteristics. In order to extract the texture feature, image patch based method is used, which considers neighbor pixels near the target pixel. For color features, we read color values from three color channels in RGB space.

For the purpose of feature clustering, k-means algorithm is employed for extracted texture vectors, and color quantization technique is used on color features. The details are described in the following sections.

5.2.1 Teature Extraction

Generally speaking, texture characteristicis represented by a group of pixels instead of single pixel, so they can supply more information, therefore give us more chance to fulfill our goal. On the other hand, textures have large stochastic variations (see figure 5.3), which makes modeling them is a difficult task.

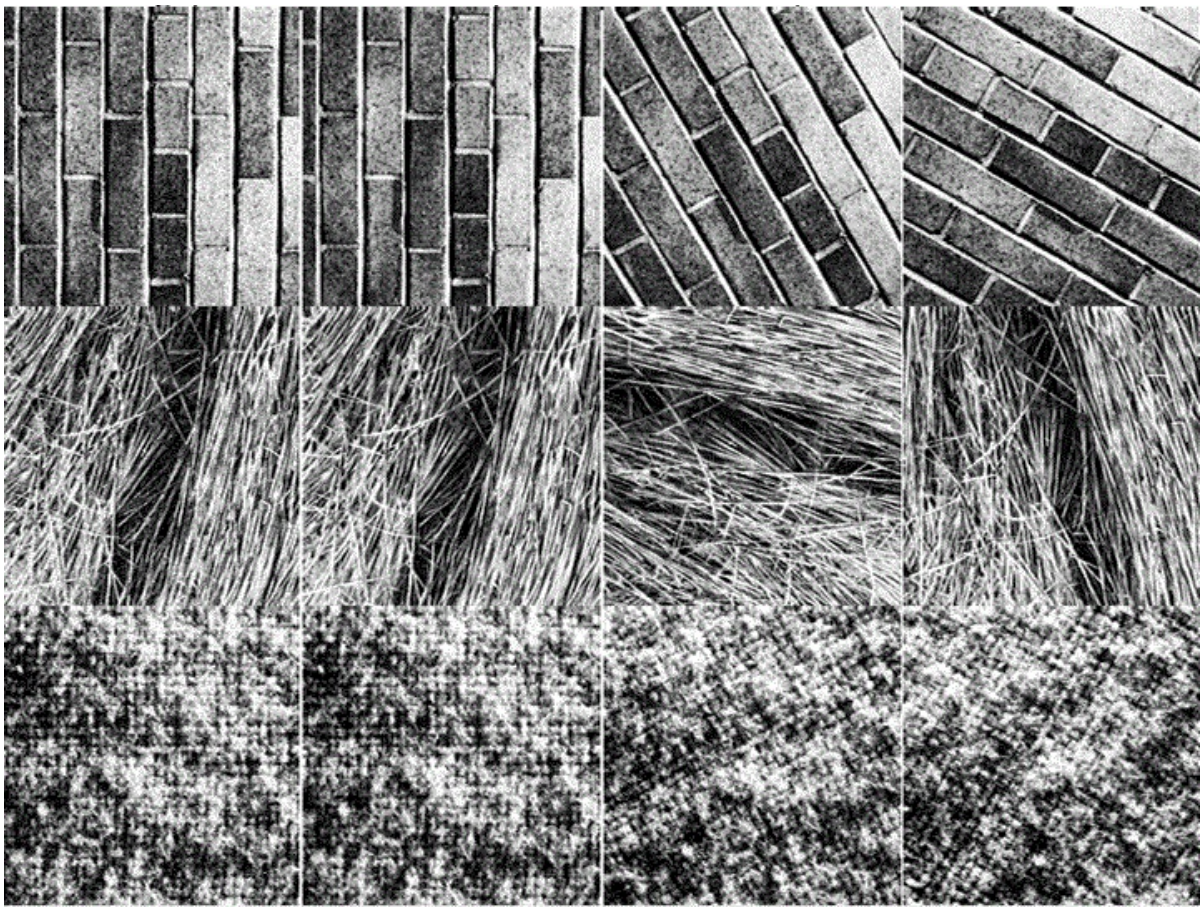


Figure 5.3: Texture rotation[1]

In this thesis, we test three popular texture extraction techniques. In particular, SIFT feature doesn't work well in our task, and it is computationally expensive, because it extracts a 128 dimensional vector at each key point.

Between filter responses and image patch [72] based methods, we choose the latter. Because patch based method uses the RGB values directly instead of derivatives, it is more efficient to compute. Also, there are evidences to suggest that the patch based classifiers give better results than filter banks even when only a small number of training images are used [73].

From the testing results, image patch with the size of 7×7 neighborhood performs well. The central pixel is not included in the representation. Thus, the image patch vector in our work is 48 dimension. Meanwhile, the patch vectors are sorted before we cluster them. The reason for reordering the row vectors is that it can make the texture feature rotationally invariant, and reduce the sensitivity toward the texture geometric deformations (see figure 5.4) to some extent.

[1]Figure cited from `http://www.cs.columbia.edu/CAVE/projects/histograms/images/mrhMatchBrodatz.gif`

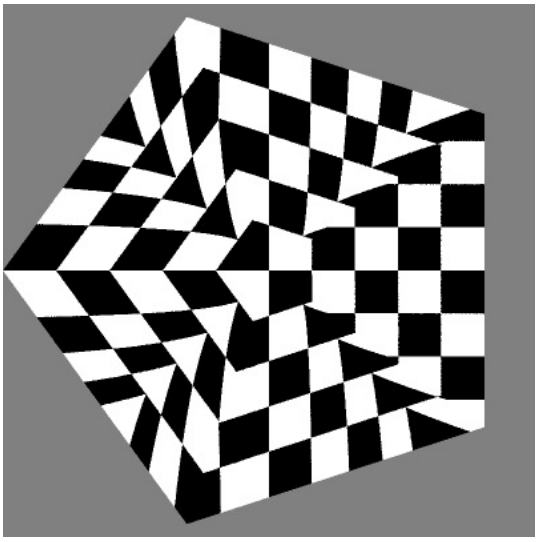


Figure 5.4: Texture rotation and geometric deformation[2]

5.2.2 Texture Clustering

Because the texture we used is a 48 dimensional vector at each pixel, theoretically, the number of the texture types is 256^{48} in a grayscale image. Considering that too many texture classes will reduce the discriminability when detecting the foreground, unsupervised learning technique k-means is employed to quantize the texture features. They are clustered into groups that contain similar texture vectors and indexed by integer. In experimental stage, we test different parameters, and set the largest number of iterations of k-means to 100, the texture features are clustered into 10 groups, and indexed with integers $1, 2, \ldots, 10$. After quantization, all texture descriptors falling into a particular cluster correspond to the same feature type. For example, features with index k correspond to cluster k ($k \in [1, 10] \cap \mathbb{N}$). We need to mention that, we find k-means clustering algorithm is unstable for some images, because it is a randomized algorithm. There may be slight difference in detection results if we test the same image many times. This due to the initialization of k-means is random, but k-means performs well in most of the cases.

In order to illustrate clustered results clearly, random colors are used to show different clusters, each color corresponds to a cluster center. In figure 5.5, the image in the middle is an example of patch based textures at each pixel that are clustered into 10 groups, while the right image shows SIFT features (6 clusters), at every 10^{th} pixel. From the testing results, patch based textures present pixel characteristics well, and are faster to extract.

[2]Figure cited from `http://imgur.com/a/UDm5B`

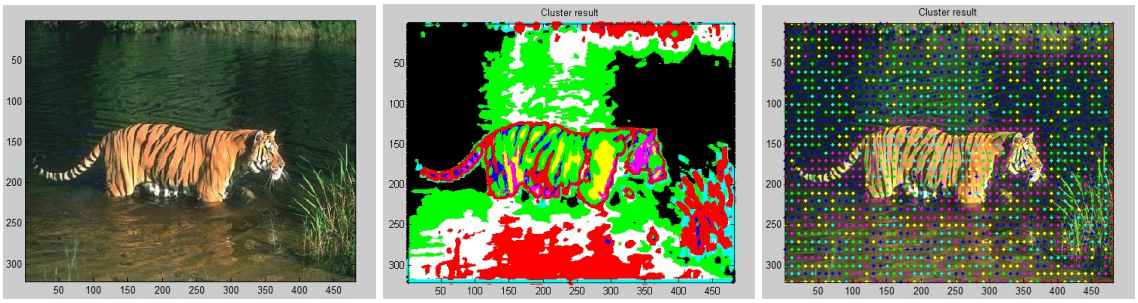


Figure 5.5: Left: original image, middle: patch based texture map (random color) with k = 10, right: SIFT features map (sampling at every 10^{th} pixel) with k = 6

We should notice that object textures often undergo an appearance change when they are imaged by a camera in different illumination conditions. Figure 5.6 shows that k-means clustering technique can weaken those effects to a large extent, the stripes of tiger are successfully clustered into the same feature group, though there are texture rotation as well as geometric deformation.

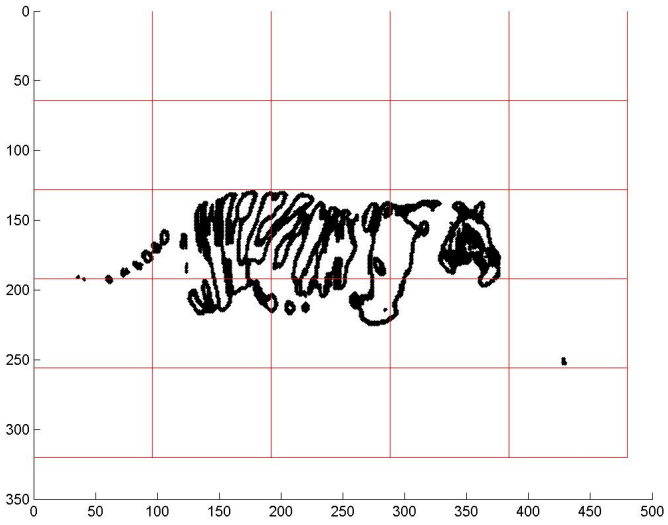


Figure 5.6: Pixels assigned to one particular texture cluster after k-means clustering of the patches, with $k = 10$. This cluster roughly corresponds to tiger stripes

5.2.3 Color Quantization

Usually, each image region contains pixels from a small subset of the color classes and each class is distributed in a few image regions [19], which means detecting object region relying on color cues is reasonable. There is a similar problem to textures that three channel images have 256^3 colors, counting all of them is time consuming and too many colors will make object detection a hard task. In order to reduce the number of colors and to make the algorithm

insensitive to small color changes, we preprocess the input image with minimum variance quantization algorithm [31], and quantized colors are assigned with indices.

What needs to mention is that we test our algorithm in both RGB and CIELAB space. Although many algorithms that process color images produce better results in CIELAB[3], it doesn't work better in our case. In this work, we stick to RGB color space.

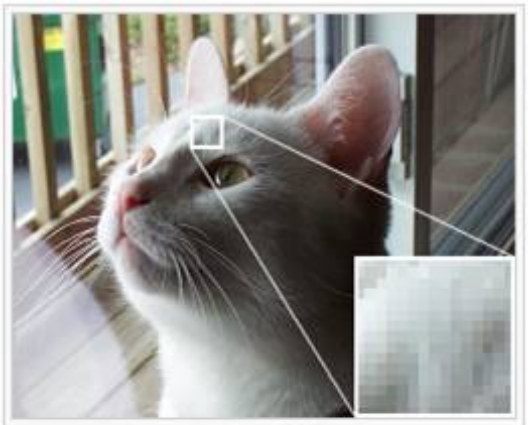


Figure 5.7: image before quantization[4]

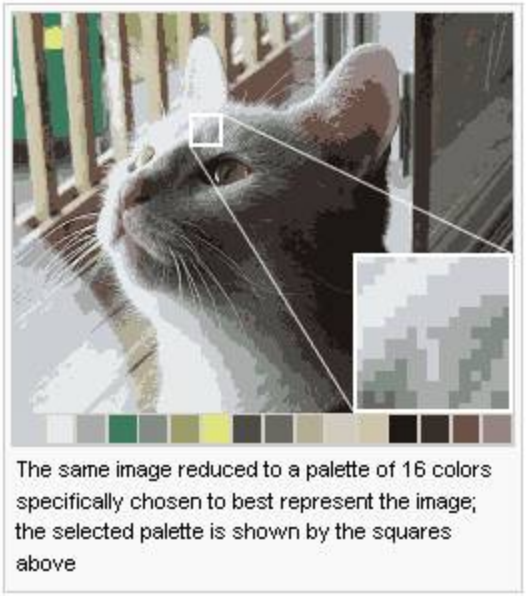


Figure 5.8: image after quantization[5]

[3] http://www.mathworks.com/matlabcentral/fileexchange/24010
[4] Figure cited from website http://en.wikipedia.org/wiki/Color_quantization
[5] Figure cited from website http://en.wikipedia.org/wiki/Color_quantization

Figure 5.7-5.8 illustrate the image before and after color quantization, we can see that after color quantization, the number of color become lesser and the image become rougher. Figure 5.9 is one of our testing results in RGB space, and in the final algorithm, we quantize all the images into 10 colors.

Figure 5.9: Color quantized image (color = 8)

5.3 Feature Selection

After the texture and color features extraction, we should have enough information to detect object contained rectangle accordingly. The reason for including this feature selection step is that not all the features will contribute to distinguish the foreground from features that are evenly spread across the image. On the other hand, in the process of comparison, counting all of them is a time consuming task. Therefore, designing a ranking function to get rid of the useless features is necessary.

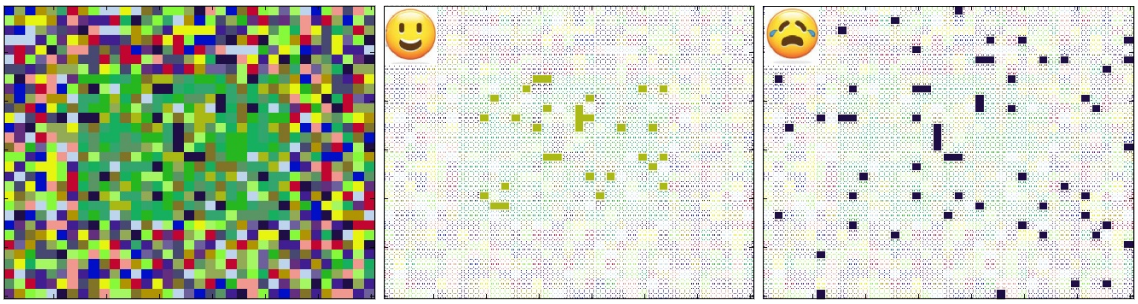


Figure 5.10: Left: artificial features map, middle: good features, right: bad features

Figure 5.10 shows examples of good features and bad features in our task. The ranking function should map the features to different indices according to their characteristics. More specifically, features that are concentrated in some areas of the image but not in others, will be ranked higher

by our function, and only the high ranked features are good features to be chosen for histogram computation.

5.3.1 Ranking Function

In this section, we will introduce the criteria to rank features and the formulation of our ranking function.

First, we partition the target image into smaller sized rectangular regions, which we refer to as "boxes". Notice that boxes are non-overlapping (see figure 5.11). Actually, before feature selection, there is a process of box selection. To select the boxes, we should count, in each feature cluster, how many features fall into every box, and choose the boxes containing more features than average in that cluster. The criteria we used to judge whether a feature cluster is good is the chosen boxes' variance, the smaller the box variance is, the better the feature is (see figure 5.12-5.13). Notice that only the boxes containing more than average feature points are useful to compute the box variance.

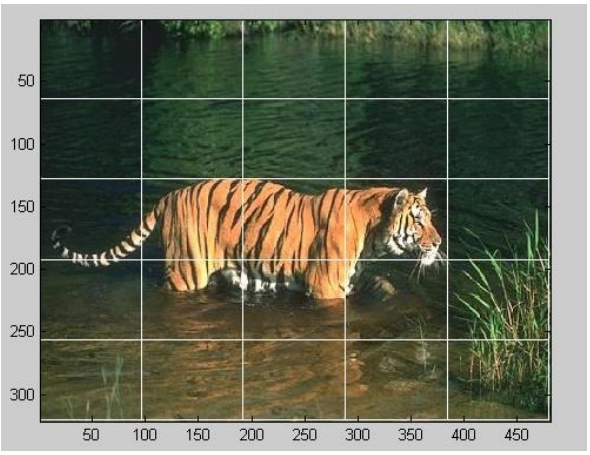


Figure 5.11: Image divided into 25 boxes

The notations and details for the ranking function that we designed are as follows.

Suppose we have features $f_1, f_2, \ldots, f_k$. We will define a ranking function $R(\cdot)$ on features, such that, if $R\left(f_p\right) < R\left(f_q\right)$, f_p is considered a better feature than f_q. Intuitively, a feature is good if it is concentrated in several nearby boxes.

Before defining $R(\cdot)$, we need to define the following notations. Let the boxes be indexed with $1, 2, \ldots, b\,(b \in \mathbb{N})$, and let $n(i, j) \in \mathbb{N}$ be the number of pixels with feature $i \in [1, k] \cap \mathbb{N}$ that are detected in box $j \in [1, b] \cap \mathbb{N}$, and $c_i \in \mathbb{N}$ be the total number of pixels with feature i in the image. If feature i was spread uniformly in the whole image, we would expect to have average feature number $\mu_i \in \mathbb{R}$ in each box, where

$$\mu_i = c_i / b \tag{5.1}$$

Our intuition is that a good feature i should be concentrated in several boxes. However, due to outliers and other random effects, we expect that there will be some boxes with very low feature concentration, and we want to exclude such boxes from feature ranking computation. In our algorithm, we will exclude all boxes with feature count $n(i, j)$ less than μ_i.

Let S_i be the set of box indexes that survive the selection step, that is, if $j \in S_i$, then $n(i, j) \geq \mu_i$. Intuitively, box j ($j \in S_i$) is the active box that participates in the ranking of the feature i. For a high rank, we expect the boxes with indexes in S_i to be spacially coherent, or in other words, they should be not too far from each other.

To measure how spatially close the selected boxes are, we will compute the variance of their centers. Let $\left(x_j, y_j\right)$ be the geometric center of box j. We separately compute variances for the x and y coordinates and add them up, namely, we use

$$\sigma(S_i) = var\{x_j | j \in S_i\} + var\{y_j | j \in S_i\} \tag{5.2}$$

Now we are ready to define our full ranking function,

$$R(f_i) = \underset{i \in [1,k]}{rank}\left(\sigma(S_1), \sigma(S_2), \ldots, \sigma(S_k)\right) \tag{5.3}$$

where, $rank(\cdot)$ returns the index of $\sigma(S_i)$ in the sequence $\sigma(S_1), \sigma(S_2), \ldots, \sigma(S_k)$ which is sorted in non-decreasing order. That is, the smaller the variance, the better is the feature.

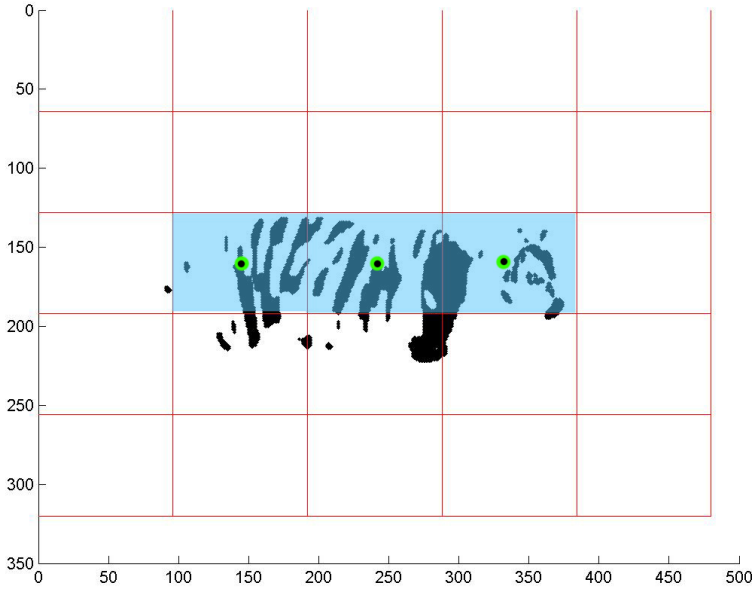


Figure 5.12: A feature with high rank (2), that is small variance in selected box centers. Selected boxes are shown in shaded blue, green circles denote the box centers.

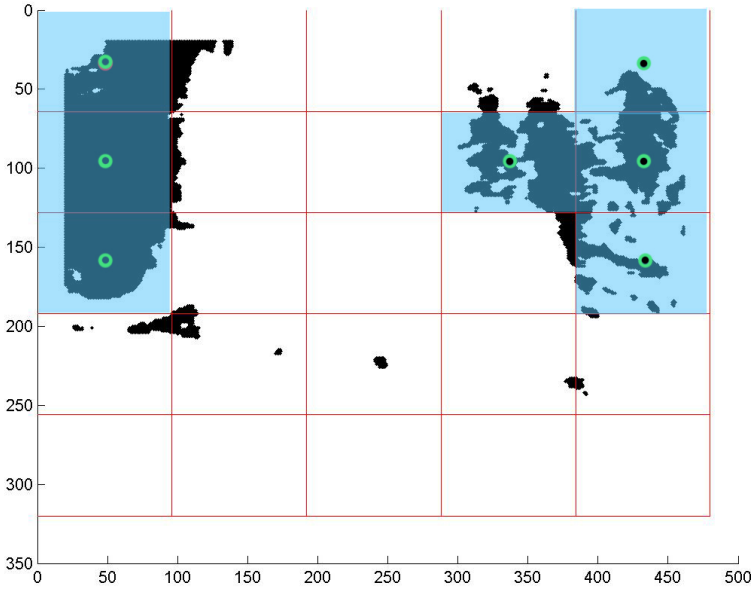
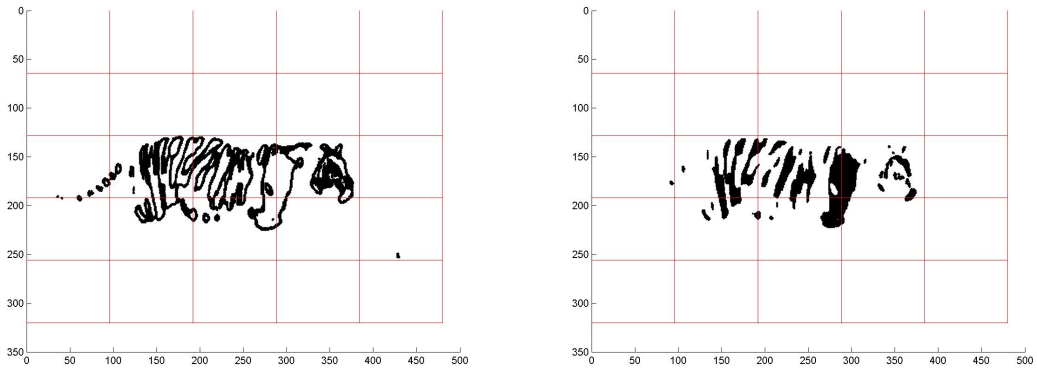


Figure 5.13: A feature with low rank (10), that is a large spread of selected box centers. Selected boxes are shown in shaded blue, green circles denote the box centers.

In the two example feature maps (figure 5.12-5.13), we use blue shadow to mark the chosen boxes. The feature (ranked 2) in figure 5.12 has smaller box center variance and therefore is ranked higher than the feature (ranked 10) in figure 5.13. It also corresponds to our intuition about these two features. Almost all pixels with feature 2 in figure 5.13 belong to the same object, while those in figure 5.13 are spread all over different objects in the scene.

Figure 5.14 below shows the patch based features map (tiger image, $k = 10$) ranked by our ranking function. In this thesis, we choose top 50 percent of texture features to compute foreground/background histograms.



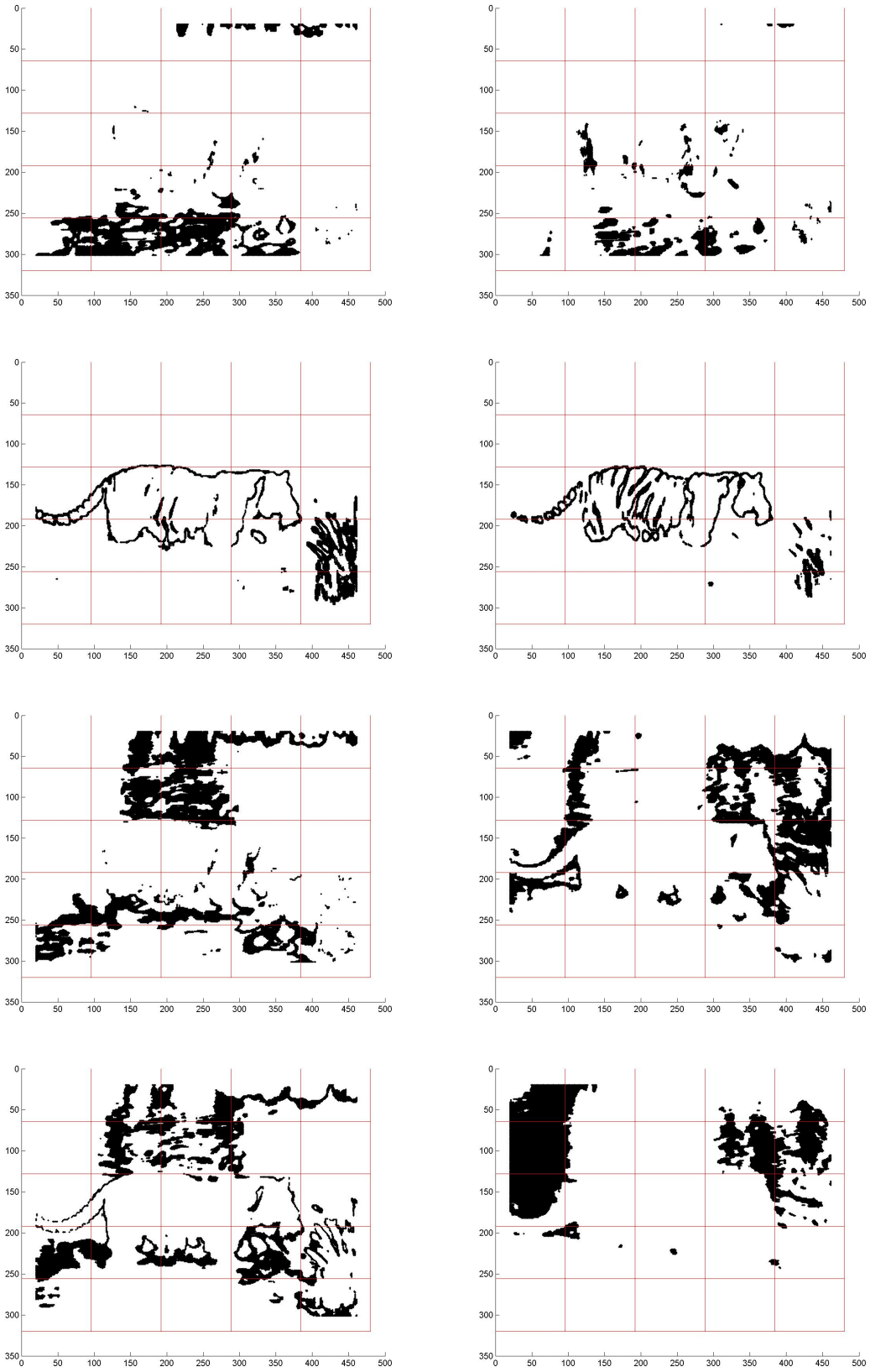


Figure 5.14: Ranked patch based features map (tiger image), from left to right and top to down, the features are sorted in non-decreasing order of box variances.

5.4 Foreground Rectangle Detection

When we get the useful features, the region containing the foreground can be detected by the sliding window approach. That is we slide a window of fixed size across numerous positions in the entire image, searching for the best placement. In our work, we employ α-divergence to measure the extent of histogram difference, and histograms are computed based on integral images [76]. In this section, we explain the foreground search details.

5.4.1 Similarity Measurement

In our algorithm, the essential part of object detection is finding the region where the inside and outside histograms show the largest dissimilarity. Therefore, choosing a proper measurement is a key point. Here, we make a choice from α-divergence family, which can be used to measure difference between two probability distributions P and Q. In this thesis, we use the foreground and background histograms to simulate P and Q, respectively. The α-divergence was proposed by Chernoff [15] and have been extensively investigated and extended by many researchers. The basic asymmetric α-divergence can be defined as [2]:

$$D_A^{(\alpha)}(P \parallel Q) = \frac{1}{\alpha(\alpha-1)} \int \left(p^{\alpha}(x)\, q^{(1-\alpha)}(x) - \alpha p(x) + (\alpha-1)\, q(x) \right) d\mu(x), \alpha \in \mathbb{R} \setminus \{0,1\} \tag{5.4}$$

The well known Pearson Chi-square, Hellinger and inverse Pearson, also called the Neyman Chi-square distances, are the special cases of α-divergence family, when $\alpha = 2, 0.5, -1$, given respectively by

$$D_A^{(2)}(P \parallel Q) = D_P(P \parallel Q) = \frac{1}{2} \int \frac{(p(x) - q(x))^2}{q(x)}\, d\mu(x) \tag{5.5}$$

$$D_A^{\left(\frac{1}{2}\right)}(P \parallel Q) = 2D_H(P \parallel Q) = 2 \int \left(\sqrt{p(x)} - \sqrt{q(x)} \right)^2 d\mu(x) \tag{5.6}$$

$$D_A^{(-1)}(P \parallel Q) = D_N(P \parallel Q) = \frac{1}{2} \int \frac{(p(x) - q(x))^2}{p(x)}\, d\mu(x) \tag{5.7}$$

The basic α-divergence is asymmetric, that is, $D_A^{(\alpha)}(P \parallel Q) \neq D_A^{(\alpha)}(Q \parallel P)$. But our aim is binary image segmentation, the foreground and background labels can be swapped, in other words, our problem is symmetric. We test Hellinger distances first, but it doesn't behave very well, therefore, we try to find a proper symmetry measurement in α-divergences family.

Generally speaking, there are two types of aymmetrized α-divergence:

Type 1

$$D_{S1}(P \parallel Q) = \frac{1}{2}[D_A(P \parallel Q) + D_A(Q \parallel P)] \tag{5.8}$$

Type 2

$$D_{S2}(P \parallel Q) = \frac{1}{2}[D_A\left(P \parallel \frac{P+Q}{2}\right) + D_A\left(Q \parallel \frac{P+Q}{2}\right)] \tag{5.9}$$

An alternative wide class of symmetric divergences can be described from them. In our work, the formula we use to compute the histograms is Eq. 5.10, which is a type 2 symmetric α-divergence.

$$\begin{aligned} D_{AS2}^{(\alpha)}(P \parallel Q) &= D_A^{(\alpha)}\left(P \parallel \frac{P+Q}{2}\right) + D_A^{(\alpha)}\left(Q \parallel \frac{P+Q}{2}\right) \\ &= \frac{1}{\alpha(\alpha-1)} \int \left(\left(p^{1-\alpha} + q^{1-\alpha}\right) + \left(\frac{p+q}{2}\right)^{\alpha} - (p+q)\right) d\mu(x) \end{aligned} \tag{5.10}$$

In particular, we set $\alpha = -1$, and use the discrete form of the measurement:

$$D_{AS2}^{(-1)}(P \parallel Q) = \frac{1}{2} D_{TD}(P \parallel Q) = \frac{1}{2} \sum \frac{(p(x) - q(x))^2}{p(x) + q(x)} \tag{5.11}$$

where, $p(x)$ and $q(x)$ stand for bins with the same index from foreground/background histograms, see figure 5.15.

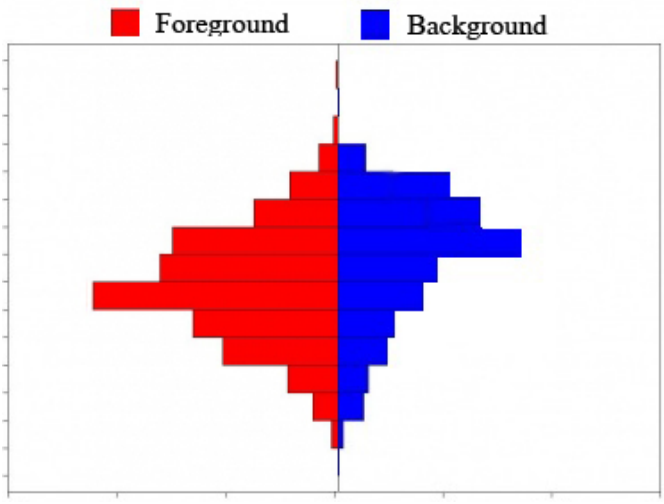


Figure 5.15: Foreground/background histograms

5.4.2 Sliding Window

In this work, we use sliding windows to search for a potential object region. For the purpose of improving accuracy and efficiency, we use the following techniques.

- **Integral Image**

  The shape of searching window is designed as a rectangle. Actually, the foreground region detection algorithm itself is not sensitive to the shape of the search window, but rectangle can speed up the searching process based on integral images [76].

  Figure 5.16 illustrates how to make use of integral image. At every point of integral

image, it records the accumulation value in the rectangle from origin to that point. In precise, the integral image at location x, y contains the sum of the pixels above and to the left of x, y, inclusive:

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{5.12}$$

Where $ii(x, y)$ is the integral image and $i(x, y)$ is the original image. The integral image can be computed in one pass over the original image using the following pair of recurrences:

$$s(x, y) = s(x, y - 1) + i(x, y) \tag{5.13}$$

$$ii(x, y) = ii(x - 1, y) + s(x, y) \tag{5.14}$$

Where $s(x, y)$ is the cumulative row sum, $s(x, -1) = 0$ and $ii(-1, y) = 0$.

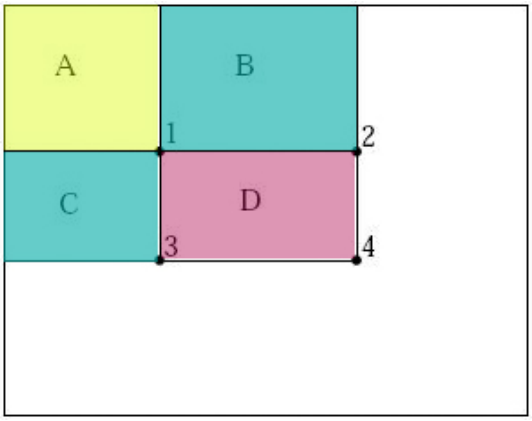


Figure 5.16: The sum of the pixels within rectangle D can be computed with four array references. The value of the integral image at location 1 is the sum of the pixels in rectangle A. The value at location 2 is $A + B$, at location 3 is $A + C$, and at location 4 is $A + B + C + D$. The sum within D can be computed as $4 + 1 - (2 + 3)$.

In the searching step of our algorithm, for every position of sliding window, the inside and outside histograms of searching window are needed to count, so computing based on integral image is efficient. Of course, this approach can only outperform the straightforward method when the cluster number is few, in our case, $k = 10$, so on every position, the total computation for foreground histogram is 4×10, which reduces the computing time to a large extent.

- **Representation Windows**

Again, considering the search efficiency, we do not search over all possible window sizes and aspect ratios, but choose three typical window sizes: a high rectangle, a wide rectangle and a square one. Actually, these three typical window sizes can cover most of the object types. In practice, at least one of the three searching windows can make a

good choice. Figure 5.17 shows some results of searching with the three representative windows, the window with the largest histogram divergence is colored with yellow.

Figure 5.17: Foreground detection results using three representation windows

- **Search Range and Step**

  Based on the assumption that the major part of the objects are not on the boundary of the input image, we ignore the image border of width 80 pixels when searching. With this constraint, not only the efficiency, but also the accuracy of the algorithm is improved. Because it is a useful technique in cases when the image is dominated by low texture or pure texture region, such as sky, grass or water.

  When searching for a foreground rectangle, we make the step size to be 10 pixels, as

opposed to every pixel. For there is a trade off between the searching accuracy and efficiency. The rough object location is enough to initialize graph cuts algorithm, and the segmentation will be refined later. Figure 5.18 illustrates the search range and step size of our algorithm.

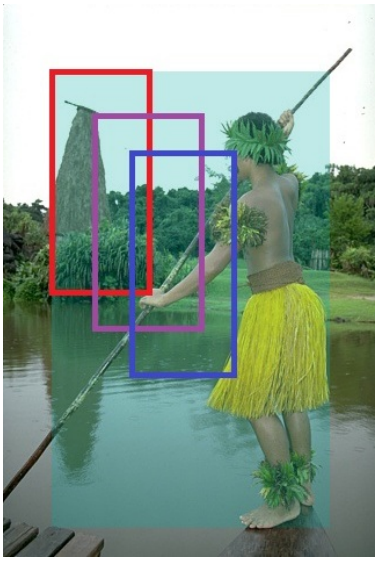


Figure 5.18: Sliding window range and step

- **No Stop at Bad features**

  No stop constraint directly arises from feature selection. As we discussed in previous section, the ranking function will choose the features which focus in certain part of the image. Therefore, after feature selection, the different bad features will connect to each other to make up some bad feature focus regions (see figure 5.19). Remembering that this region is made up of bad features, so forbiding the searching window to stop in bad features region makes sense. In our special case, the no stop constraint is defined like this: if more than $\frac{2}{3}$ of the features in the current search window are bad features, then the divergence on that position will be set to a small constant.

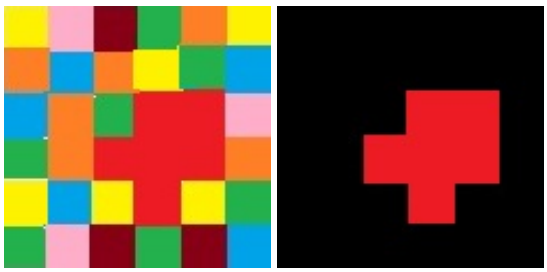


Figure 5.19: Left: artificial feature map, right: good features(red) and bad features focus region(black)

For more detection results and related analysis, see chapter 7.

Chapter 6

Segmentation Using Graph Cuts

As explained in previous chapters, the rectangle segmentation result will be used to automaticly start graph cuts algorithm. Notice that the proposed method tries to mine the useful information from feature space, what is important to us is not "what" but "where" in the detection stage. To be more precise, the top-left and down-right corners of the best detection rectangles are recorded in the previous stage. In the following step, the rough rectangle segmentation result will be refined by graph cuts. Graph cuts execute based on the color models learned from the inside and outside of the rectangle, respectively.

In this chapter, we will explain the specific energy function and the algorithm that refines binary segmentation.

6.1 Energy Function

According to chapter 3, segmentation of an image is a labeling problem which can be addressed in the energy minimization framework. Here, our goal is to partition the image into two sets, the object and background, by utilizing graph cuts algorithm. The energy function has the general form as introduced before:

$$E(f) = \sum_{p \in \mathcal{P}} D_p\left(f_p\right) + \sum_{\{p,q\} \in \mathcal{N}} V_{pq}\left(f_p, f_q\right) \tag{6.1}$$

Where, $\mathcal{N}$ is a 4-neighborhood system, $\mathcal{P}$ is the set of image pixels, $D_p\left(f_p\right)$ is the data term, and $V_{pq}\left(f_p, f_q\right)$ is the smoothness term for two neighboring pixels p and q.

Again, for the purpose of reducing color number, image quantization is used. In this stage, the target images are quantized into 250 colors by minimum variance quantization algorithm [31], and the foreground/background histograms are counted based on the quantized image.

6.1.1 Data Term

Given the rectangle detected in the previous step, we initialize all pixels inside it to belong to the foreground, and all the pixels outside belong to the background. We then compute the his-

togram counts inside the foreground and histogram inside the background regions, normalize the two histograms separately into the range [0, 1]. After normalization, the data terms related to the histograms are computed as below:

$$D_p\left(f_p\right) = -logPr\left(Q_p \mid f_p\right) \tag{6.2}$$

where Q_p denotes the quantized color of the pixel p. In our binary segmentation task, $f_p \in \{0, 1\}$, therefore

$$D_p\left(1\right) = Pr\left(Q_p \mid 1\right)$$

$$D_p\left(0\right) = Pr\left(Q_p \mid 0\right)$$

The data term for each pixel is assigned according to the foreground/background histograms. For any pixel, if it's color is more frequent according to the foreground histogram, its data term for label 1 (foreground) is smaller than the data term for label 0 (background). If a pixel's color is more frequent in the background histogram, then the data term for the background label is smaller than for the foreground label. Thus pixels with colors that occur more often in the foreground are encouraged to be assigned to the foreground label, and other pixels are encouraged to be assigned to the background label.

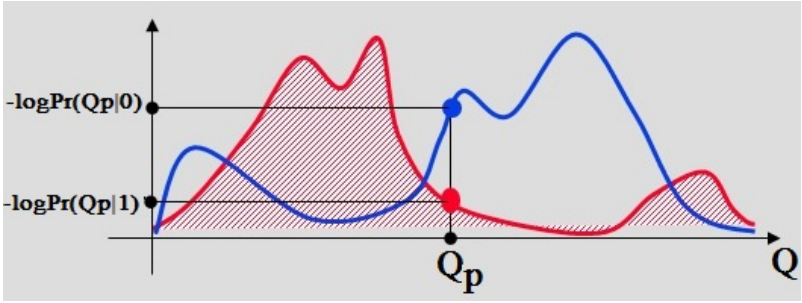
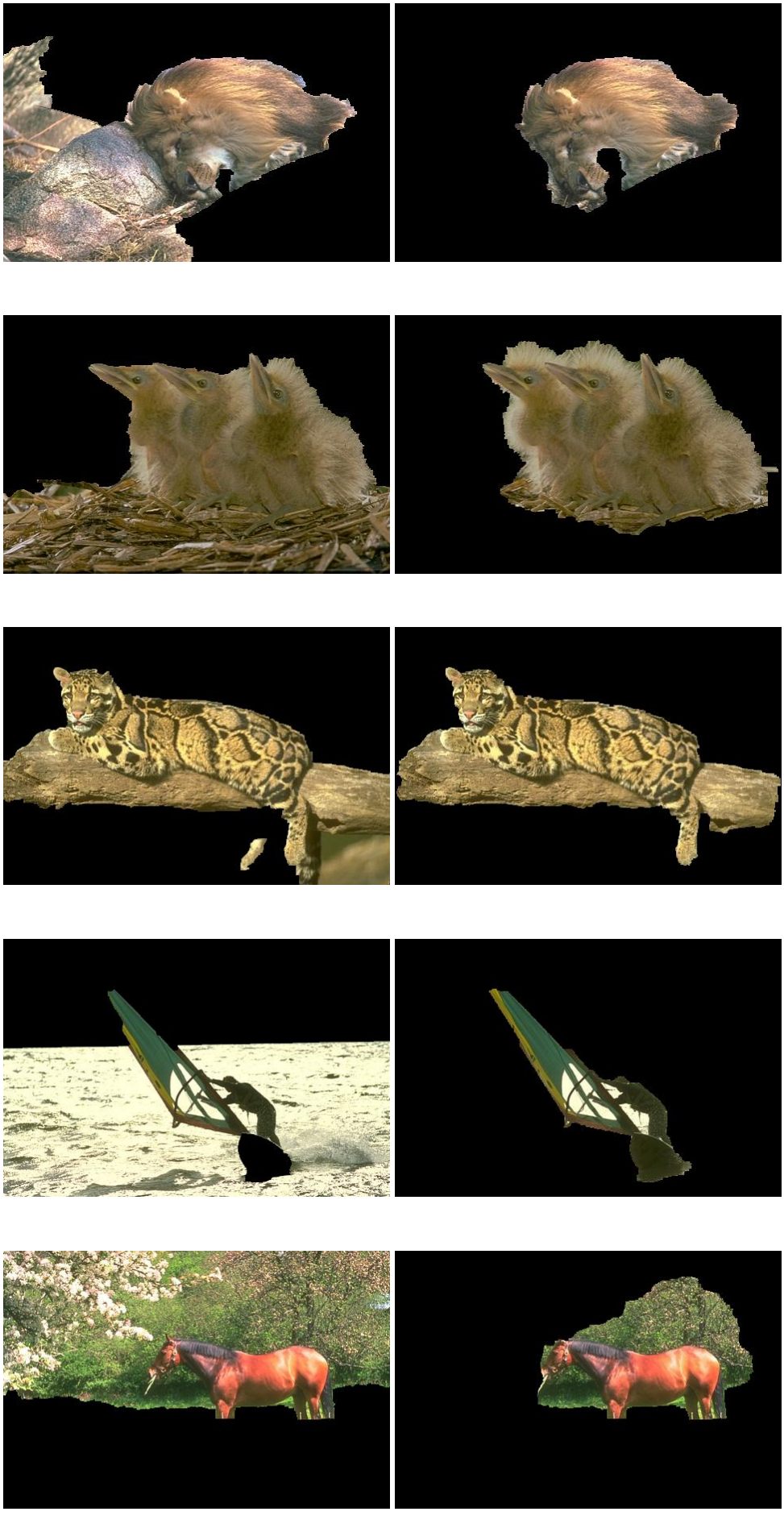


Figure 6.1: Computing data terms from foreground/background color modes

Figure 6.1 illustrates computing data terms from foreground/background histograms, respectively. In this figure, red curve corresponds to the foreground and the blue curve corresponds to the background. Consider pixel p, its quantized color is more frequent in the foreground, therefore, the negative log probability is lower for the red curve. This pixel will be encouraged to be assigned to the foreground in the segmentation.

According to the assumption that the major part of the object is not located on the boundary of the input images, we introduce the hard constraints on the image boundary, that is, we chain the border (one pixel width) on the image boundary to be the background. In particular, the border pixels need to pay 10000 for choosing foreground label, and 0 for background label, therefore, all these pixels will choose background label after energy minimization. Notice that these hard constrains are fixed for all the input images beforehand, so this rule will not violate the goal of automatic segmentation, and the results can be improved significantly, see figure 6.2.



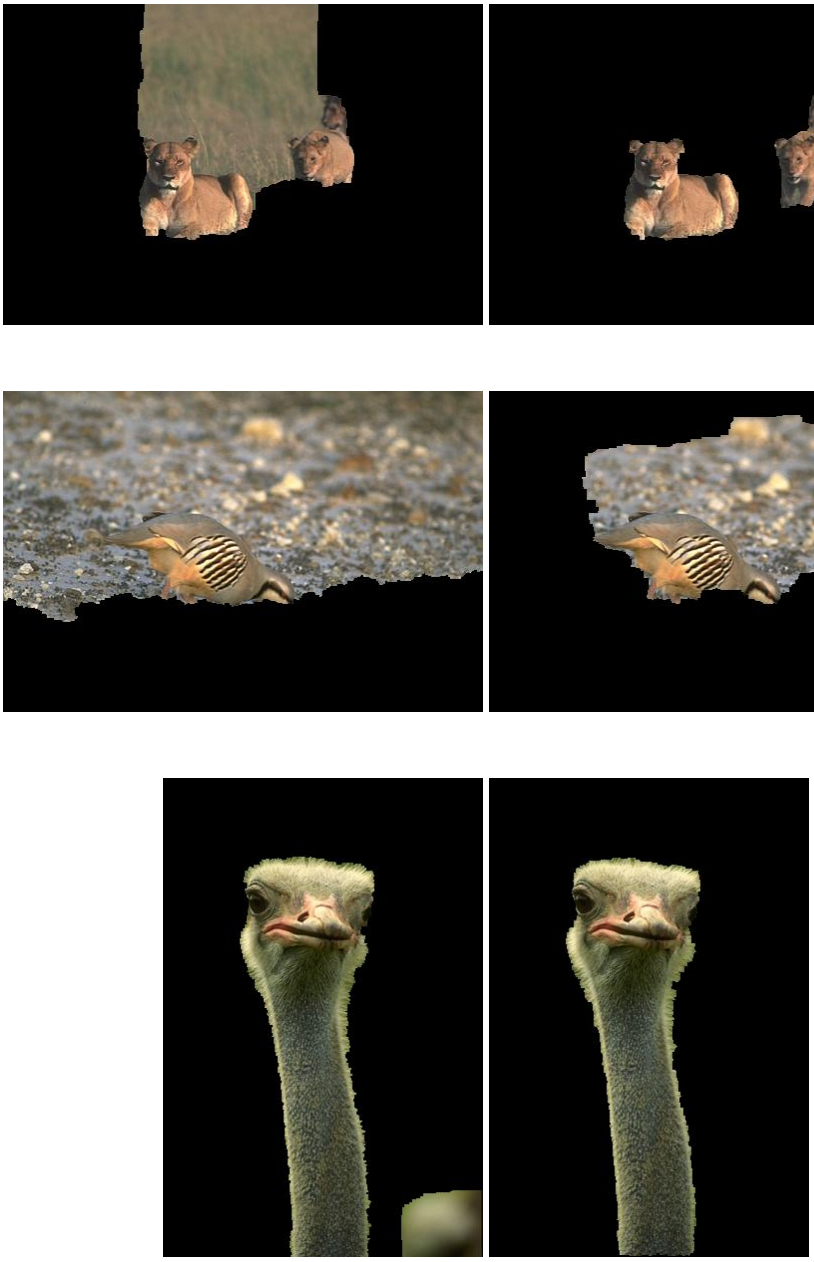


Figure 6.2: Column 1: segmentation results without boundary constraints, column 2: boundary constraints added

6.1.2 Smoothness Term

The smoothness term in Eq. 6.1 penalizes discontinuities between neighbour pixels p and q. To make it clear, if the two pixels are similar in color and have different labels, the penalty is high for the discontinuity. Otherwise, the penalty is smaller, because pixels different in appearance are more likely to belong to different segments (foreground or background), then the penalty for discontinues in this case is small.

The smoothness term used in this work is defined in paper [8]:

$$V_{pq}\left(f_p, f_q\right) = \delta\left(f_p, f_q\right)\left(1 + \gamma \cdot exp^{-\frac{dist(C_p, C_q)}{\beta}}\right) \tag{6.3}$$

$$\delta\left(f_p, f_q\right) = \begin{cases} 1 & if \quad f_p \neq f_q, \\ 0 & otherwise \end{cases}$$

In Eq. 6.3, C_p denotes the original color of pixel p in the input image, and we use L_1 norm[1] to represent color distance. In particular, $dist\left(C_p, C_q\right) = \frac{1}{3}\|C_p - C_q\|_1$, and the constant $\beta = \frac{1}{3}\left\langle\|C_p - C_q\|_1\right\rangle$, which is the average of color difference between neighboring pixels in the image. This ensures that if the color difference between neighboring pixels is smaller than average, we have to pay a heavier penalty to put them in different segments. If color difference is larger than average, then the separation cost is not so large. We assign constant $\gamma = 10$ in this work, so Eq. 6.3 can be rewritten as below:

$$V_{pq}\left(f_p, f_q\right) = \delta\left(f_p, f_q\right)\left(1 + 10 \cdot exp^{-\frac{\|C_p - C_q\|_1}{\left\langle\|C_p - C_q\|_1\right\rangle}}\right) \tag{6.4}$$

6.1.3 Energy Function Sub-modularity

It is easy to see that the energy function used in this chapter is sub-modular [12, 43]. Therefore, it can be minimized by graph cuts algorithm.

As explain in chapter 3, to prove the sub-modularity, it is sufficient to prove that

$$E(0,0) + E(1,1) \leq E(0,1) + E(1,0) \tag{6.5}$$

Since $1 + \gamma \cdot exp^{-\frac{dist(C_p, C_q)}{\beta}} \geq 0$, we only need to prove the formula

$$\delta(0,0) + \delta(1,1) \leq \delta(0,1) + \delta(1,0) \tag{6.6}$$

and it is obvious from the definition of $\delta(\cdot)$.

6.2 Automatic Segmentation Algorithm

Similar to grabcut, the algorithm presented in this thesis works iteratively to update the segmentation. The difference between our method and grabcut is in the aspect of hard constraint. In grabcut, the outside of the rectangle chosen by user works as background seeds and the labels of these pixels will be fixed during the entire iterative process, while, there will be no user input label in our case. Moreover, we should not expect that the rectangle covers the entire object perfectly with our automatic initialization method. It is most likely that the rectangle that we find only covers part of the object (figure 6.3). But our rectangle contains a good portion

[1] p-norm: $\|X\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}$, $p \leq 1$

of the object, because a large portion does stick out. During the iteation, we allow all pixels to update their labels, except the one pixel width image border, which is hard-constrained to be the background.

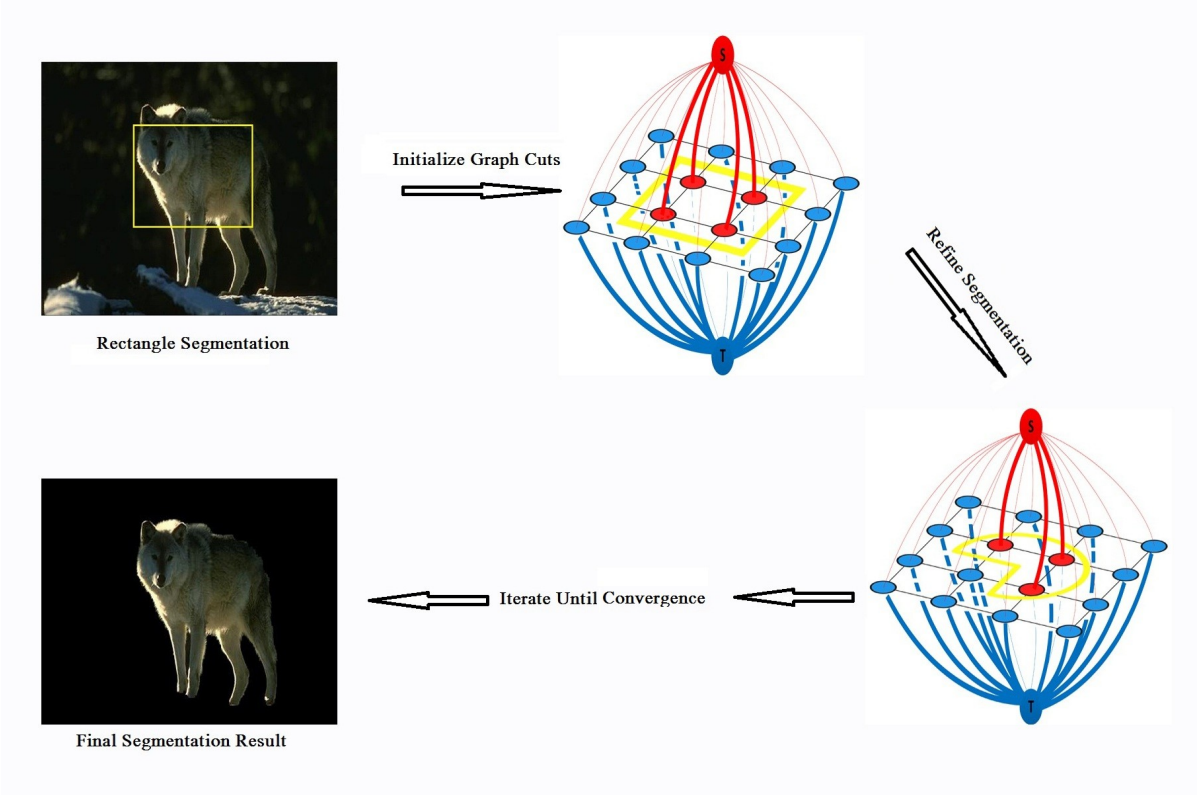


Figure 6.3: Segmentation using iterative graph cuts

Figure 6.3 is an illustration of the proposed segmentation algorithm. First, we initialize binary labels of an image according to the rectangle segmentation, compute histogram counts and the data costs, a graph cut is used to compute the labeling which minimizes the energy. After that, we recompute the histogram counts for the new foreground/background regions and run a new round of graph cuts, obtaining, potentially, a new segmentation. This process will be repeated until the energy convergence is reached.

For comparison, we segment the same wolf image using grabcut implementation by Wang [77](see figure 6.4), the red rectangle is the user initialization, and the background brush (blue) are used to refine the segmentaion. it can be seen that in order to get the segmentation results in the same level, at least three interactions are needed.

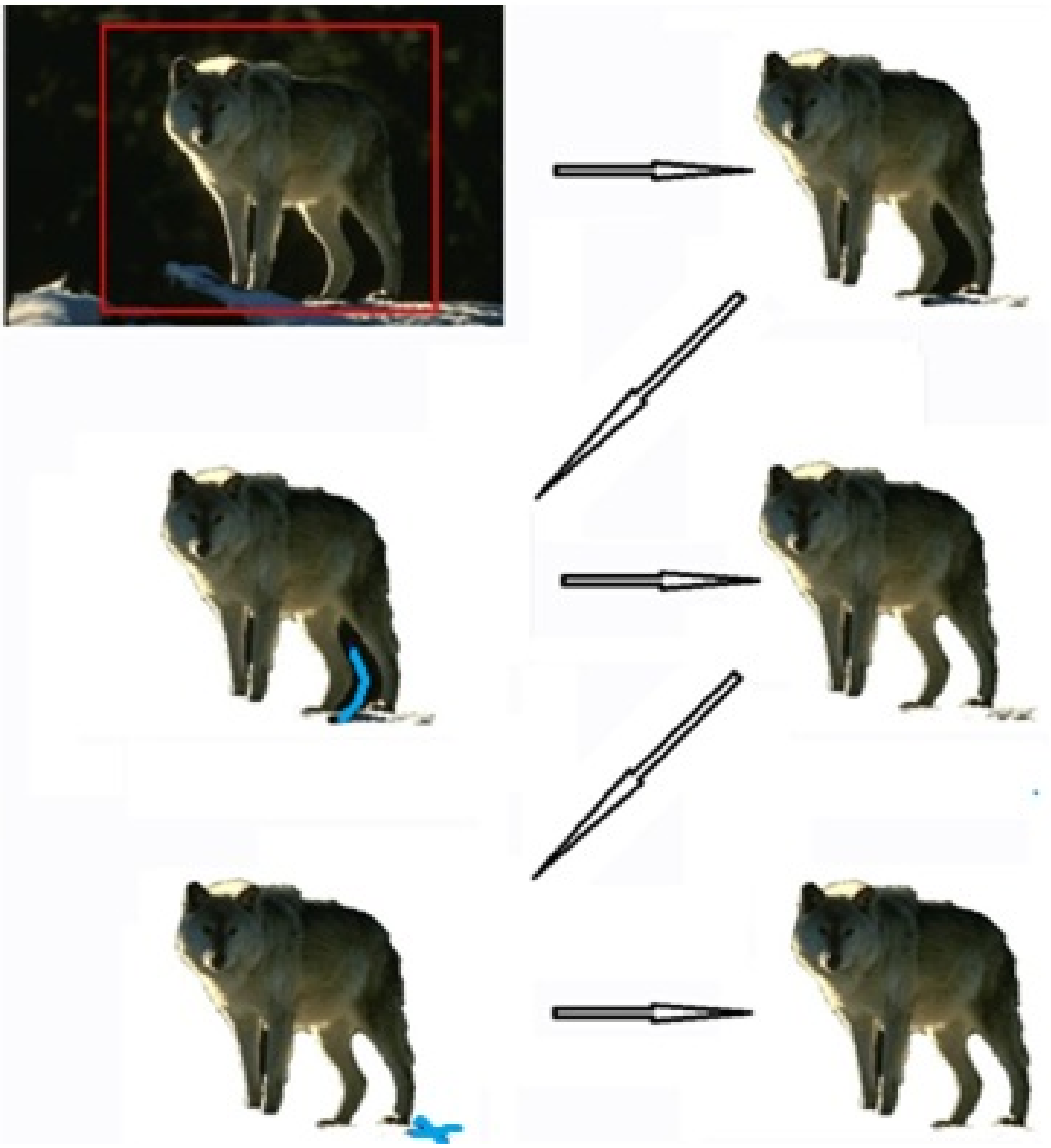


Figure 6.4: Segmentation using grabcut, column 1: user initialization rectangle (shown in red) and interactons (shown in blue), column 2: refined segmentation results

The main reason for using iterative segmentation is that it can improve the initial segmentation by refining the color models. To make it clear, suppose that the one shot segmentation does not segment the whole object. In the following re-estimating step, the new color models will be constructed based on the previous round of segmentation, those wrong labeled pixels might fit the new object color model better. Therefore, these pixels will more likely be labeled as foreground in a new round of segmentation. The same case is true for the background pixels.

During the process of re-estimating, the smoothness terms will keep the same, while the data terms will be updated in every round, we describe the new energy function as $\hat{E}(f) = \hat{E}_{Data} + E_{Smooth}$, and the proposed iterative segmentation algorithm is represented as follows:

Algorithm 1 Proposed Algorithm (segmentation stage)

1: Initialize data terms E_{Data} from object contained rectangle;
2: Compute β value of the image;
3: Compute smooth terms E_{Smooth} of the graph;
4: $minE(f) = E_{Data} + E_{Smooth}$;
5: **while** Segmentation algorithm did not converge **do**
6: update data terms $\hat{E}_{Data}$ from previous segmentation;
7: $min\hat{E}(f) = \hat{E}_{Data} + E_{Smooth}$;
8: **end while**

Iterative segmentation is a very useful technique, which guarantees the energy to converge at least to a local minimum. the following is an example (figure 6.5) to illustrates the process of updating segmentation, and the chart (figure 6.6) shows the energy reduction. For more experimental results, see chapter 7.

Original starfish image

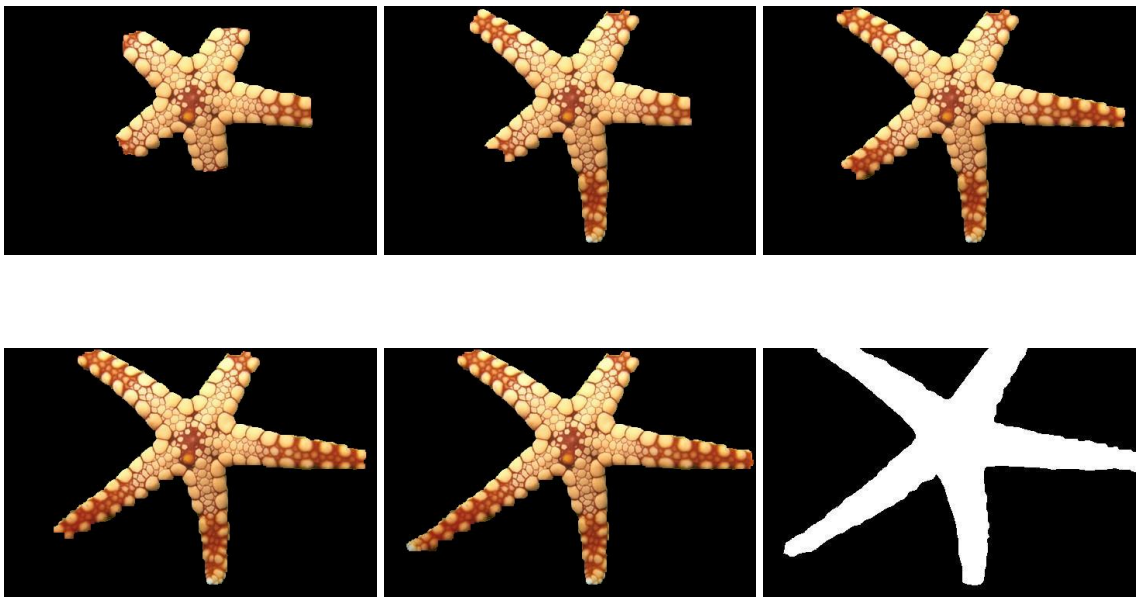


Figure 6.5: Segmentation results in iteration 1 to 5, compared with ground truth

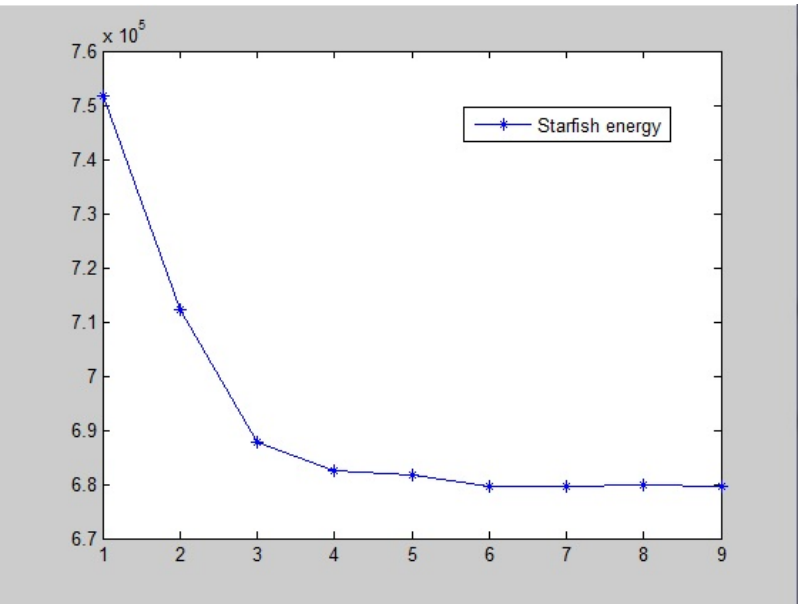


Figure 6.6: Graph showing convergence process of the energy on the starfish image. Horizontal axis plots iteration number. Vertical axis plots the energy value. Convergence is achieved after 9 iterations, but most of the progress is made during the first three iterations.

Chapter 7

Experimental Results

This chapter presents extensive experimental results of the proposed automatic segmentation algorithm. In addition, we will explain more details about parameter selection and results analysis.

7.1 Parameter Selection

In order to implement automatic segmentation, besides algorithm design, parameter selection is another challenge of this work. Because during the whole segmentation process, no user guidance or results editing is allowed. On the other hand, natural images show a significant amount of variation, and we have to set parameter values that work well for a large portion of images. Hence, choosing proper common parameters becomes a key point to acquire good results.

For the task of accurate automatic segmentation, we test and analyze different parameters, which are listed below:

- **Box Size**

  The size of box in feature selection stage affects the average pixels number when computing box variance, in other words, it is important in the process of box selection. Theoretically, too large box size will not measure pixel focus extent well, while too small box size is compositionally more expensive, since there are many more places to check for a smaller box placement.

  Figure 7.1 is an example of SIFT features aggregated in boxes of different sizes. We can notice that in small box case, less features show up in every box, while in large box, the features may only occupy part of box, which will decrease the meaning of box selection.

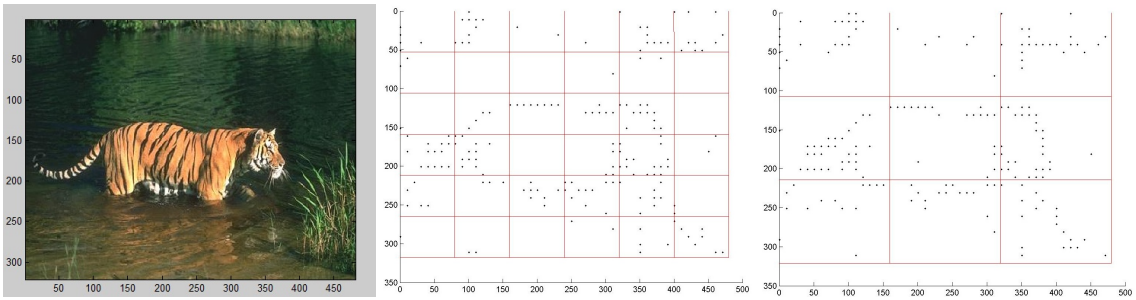


Figure 7.1: Left: original image, middle: one group of SIFT features in 6×6 boxes, right: the same features in 3×3 boxes

In the experimental stage, we test the box with the sizes of 3×3, 5×5 and 8×8 for patch based features, and only little change are found in rectangle detection results, all of the three sizes are reasonable (see figure 7.2-7.3). In the implementation, we choose to use box with size 5×5.

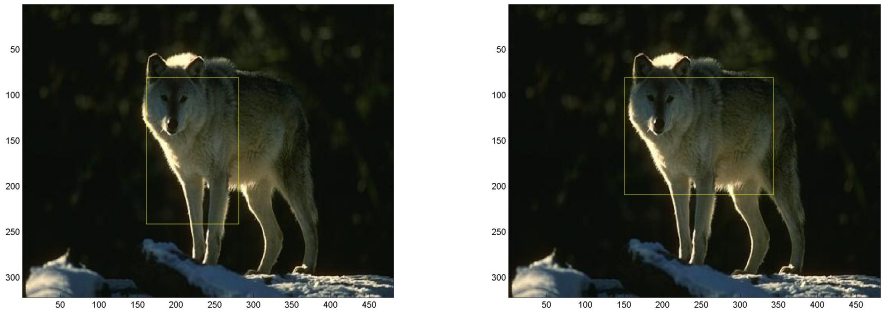


Figure 7.2: Left: detection result with 3×3 boxes, right: detection result with 5×5 boxes

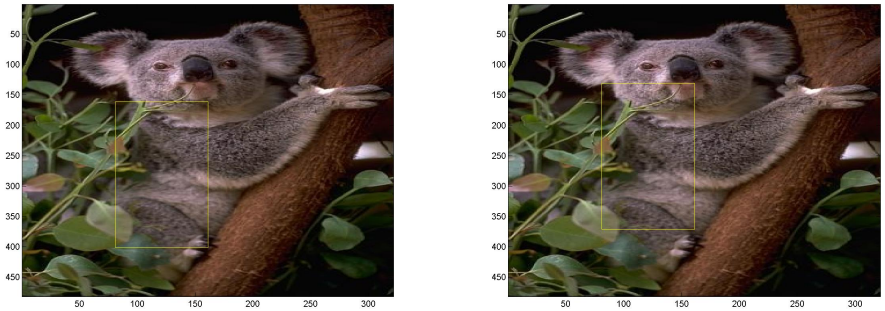


Figure 7.3: Left: detection result with 8×8 boxes, right: detection result with 5×5 boxes

- **Number of Clusters**

  The number of clusters determines how many groups the features are divided into. If

we use too many clusters, then there may be too many feature types, therefore, too few samples for each feature type (see figure 7.4). This may hinder finding interesting features, that is features that focused in some particular region in the image. In addition, if we have too many features, computing histogram counts becomes too expensive, as explained in chapter 5. So in the final algorithm, we set both texture and color cluster numbers to 10, and these choice make the algorithm perform well.

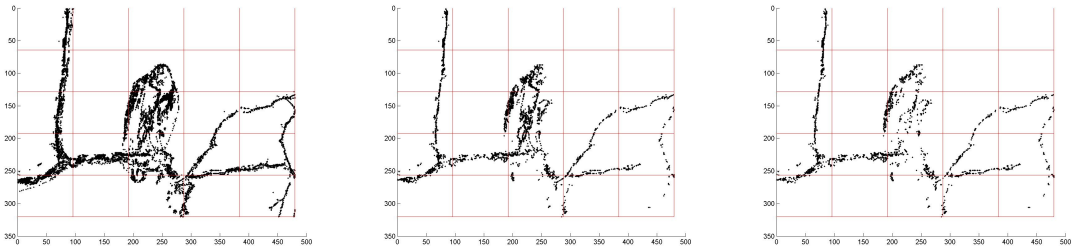
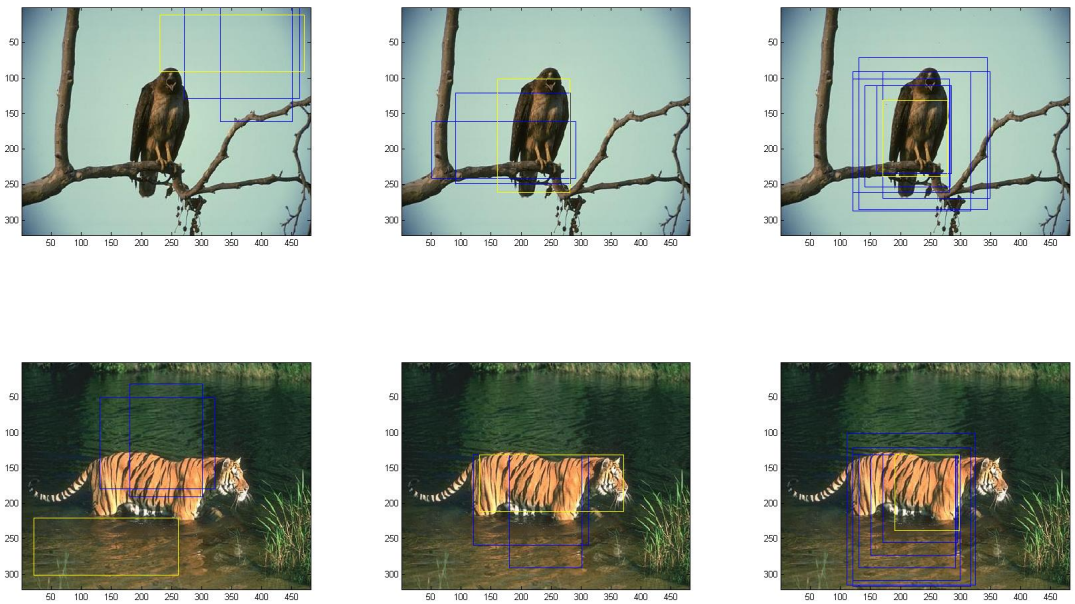


Figure 7.4: Color cluster results, from left to right, color number equals to 10, 20 and 40.

- **Window Size and Number**

  Ideally, we should test all the window sizes and shapes, before choosing the best object position. But it is unfeasible in practice, as explained in chapter 5, we choose three typical windows instead. In figure 7.5, columns 1, 2 are search results using 3 windows, while column 3 is the result of search using 7 different windows. From these illustrations, it seems that most search windows concentrate around the same most promising area, and thus we can save search time by using only a few distinct window sizes. In the final implement, the three representative window size are $[\frac{1}{2}, \frac{1}{4}]$, $[\frac{1}{4}, \frac{1}{2}]$ and $[\frac{2}{5}, \frac{2}{5}]$ of the image width and height, respectively. Notice that it is a "square" in ratio.



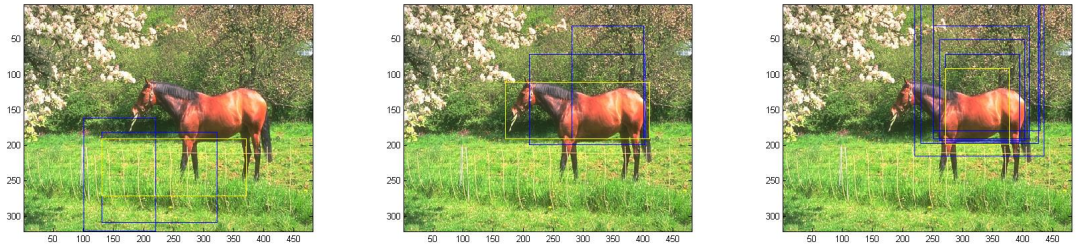


Figure 7.5: Column 1: detection results based on half of the color, column 2: detection results based on all color features, column 3: detection results with seven different windows.

- **Useful Feature Proportion**

  In this thesis, we only select the top 50% of texture features. From the experimental results, detection based top half of the color features causes significant failure (see figure 7.5 column 1 and column 2), so we give up feature selection in color space.

  In order to make a good choice, we test different percentage of features. 80% and 30% perform slightly worse than half texture (see figure 7.6-7.7).

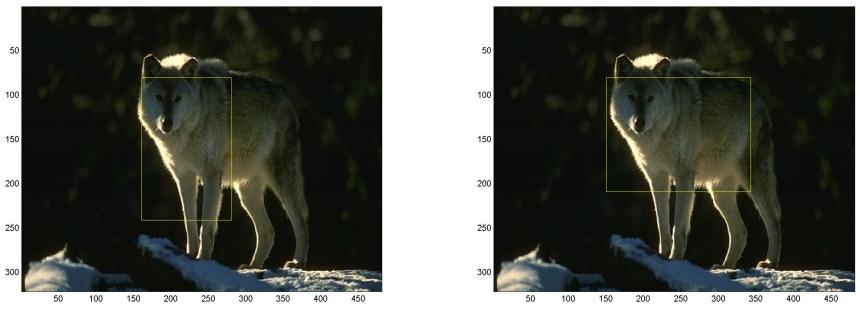


Figure 7.6: Left: detection result on top 30% of texture, right: detection result on top 50% of texture

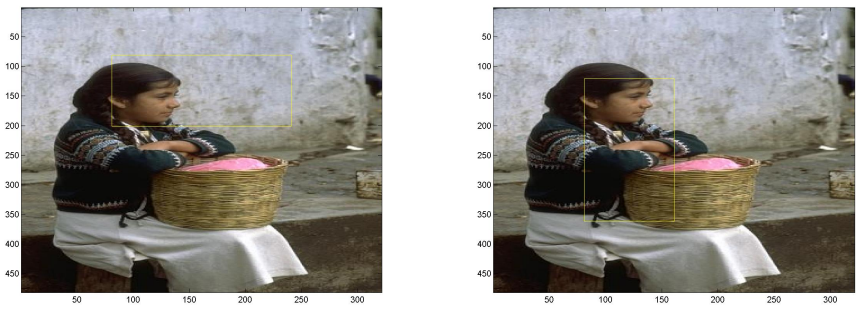


Figure 7.7: Left: detection result on top 80% of texture, right: detection result on top 50% of texture

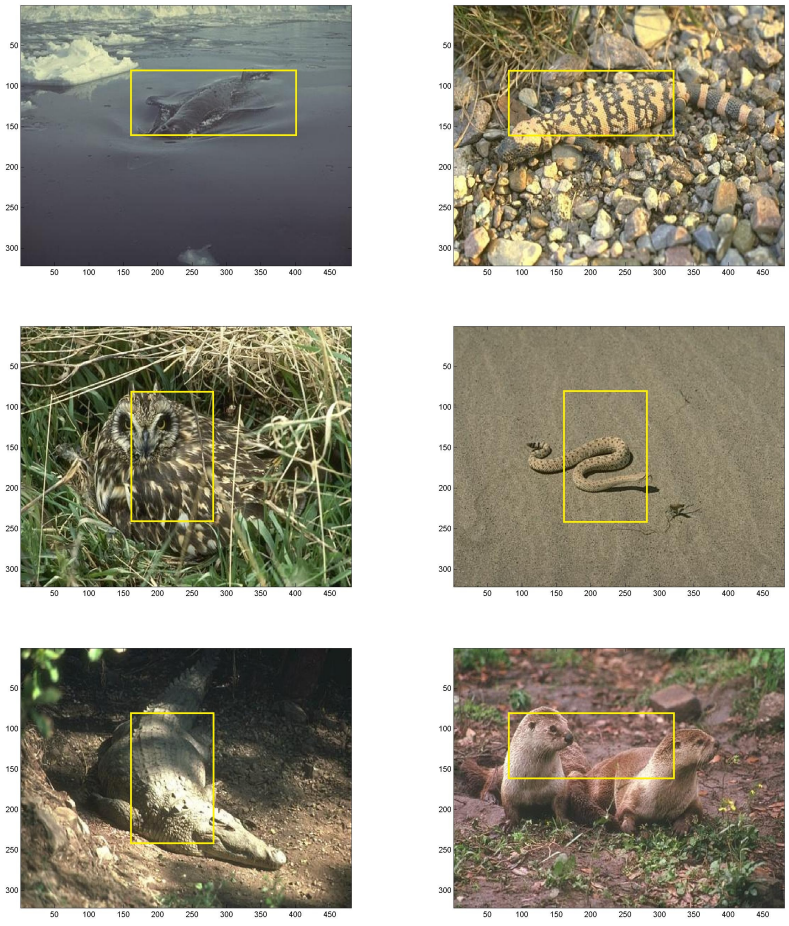
- **Patch Size**

  Patch size is related to the length of texture vector, therefore it affects the computational efficiency directly. Meanwhile, the quality of texture depends on how many nearby pixels are considered. During experiment, patch size 5×5, 7×7 and 9×9 are tested. We find that the computational time is more sensitive to different patch sizes than texture quality in our algorithm, and our final patch size is 7×7.

- **Texture Color Balance**

  The balance between texture and color decides the weight we put on these features. For example, large balance means the detection depends on texture more than color. We test three balances 0.3, 0.5 and 0.7, and judging from the testing results, all of them can make relatively good choice. In this work, we rely on them in the same extent, so the balance is set to 0.5.

  What is surprising is that detection based on combination of texture and color features succeed in many cases of animal camouflage, figure 7.8 shows some examples.



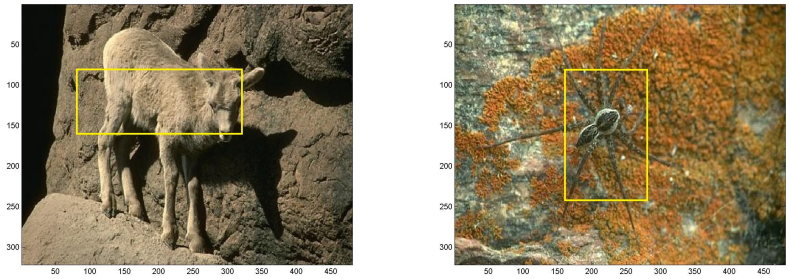


Figure 7.8: Successful results on animal camouflage

- **Data and Smoothness Terms Balance**

  The balance λ between data terms and smoothness terms decides the weight between the regional and boundary constraints. Large λ will cause segmentation result to be over-smoohted, while when λ is too small, the reault is not smooth enough. In figure 7.9-7.10, the segmentation results for different λ are illustrated.

  Actually, for different dataset, the best λ is likely to be different. And we should choose the one which tends to work best for all dataset, in our work, we choose $\lambda = 10$ for all the experiments.

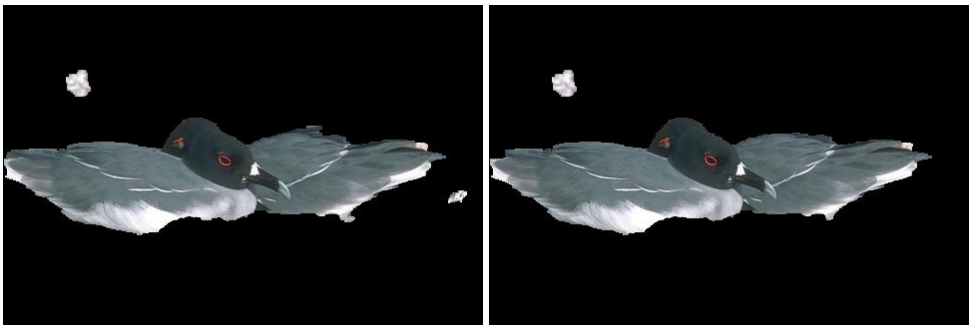


Figure 7.9: Left: $\lambda = 6$ (not smooth enough), right: $\lambda = 8$ (not smooth enough)

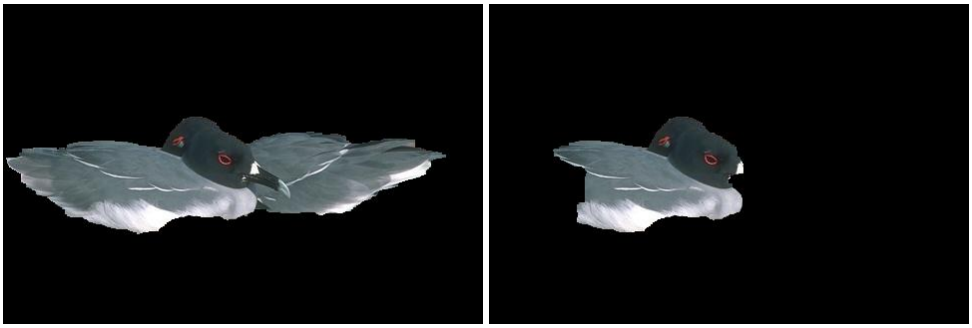


Figure 7.10: Left: $\lambda = 10$ (just right), right: $\lambda = 12$ (over-smoothed)

7.2 Experimental Results

7.2.1 Image Database and Running Time

Our algorithm is tested on three dataset: Berkeley Segmentation Dataset[1](BSD, 300 images) [54], Grabcut Dataset[2](GSD, 50 images), and Achanta et al.(ASD, 1000 images)[1].

The foreground detection algorithm is implemented with MATLAB, and the average running time for a 321×481 image in BSD is 38 seconds, on a 3.10 GHz processer with 4 GB RAM. The image sizes in ASD are different, most of them with the size of 300×400, and the average foreground detection time is 27 seconds. On all the datasets, segmenting one image using graph cuts (developed in C++) runs in 0.3-0.6 second.

7.2.2 Evaluation of the Results

We evaluate the segmentation results by computing the error rate compared with the ground truth (hand-labeled by certain people), that is counting the percentage of mislabeled pixels. We compute the "pixel error"(PE) and the "class error"(CE). For "pixel error", the formula used is

$$PE = \frac{|L_{GT} \cap L_{Seg}|}{|L_{GT}|} \tag{7.1}$$

For "class error", the "foreground" and "background" errors are computed separately, and then we record the mean error. The mean error can be compute as follows:

$$CE = \frac{1}{2}\sum_{i=0}^{1}\frac{|L^i_{GT} \cap L^i_{Seg}|}{|L^i_{GT}|} \tag{7.2}$$

where, L_{GT} denotes the ground truth label map, and L_{Seg} means label map from our algorithm, $i = 0, 1$ denote background and foreground, respectively.

BSD	Paria's PE	Pixel error	Foreground error	Background error	Mean F/B E
Iteration 1	0.2164	0.2376	0.5955	0.0962	0.3459
Convergence	0.2012	0.2385	0.5306	0.1172	0.3239

Table 7.1: Average errors for pixel, foreground, background, and mean of them in different iterations (300 images in BSD)

	Paria's PE	Pixel error	Foreground error	Background error	Mean F/B error
GSD	-	0.1626	0.3468	0.1148	0.2308
ASD	0.0758	0.0955	0.1902	0.0760	0.1331

Table 7.2: Average errors for pixel, foreground, background, and mean of them when reaching convergence (50 images in GSD, 1000 images in ASD)

[1] http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/

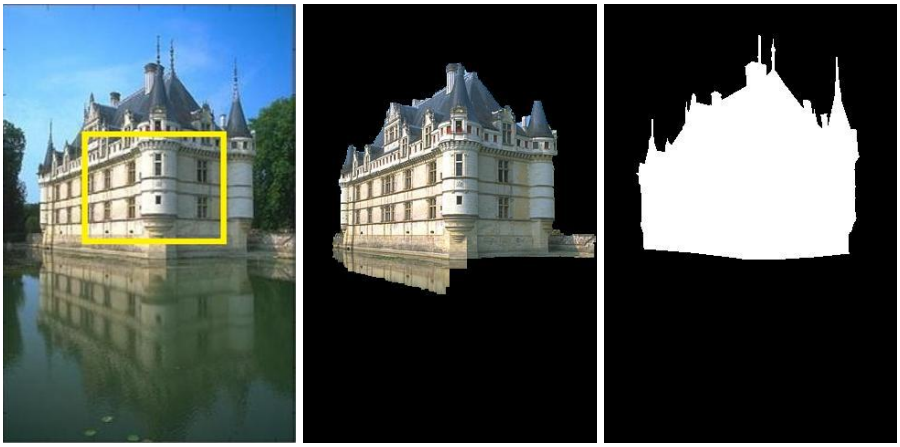
[2] http://research.microsoft.com/en-us/um/cambridge/projects/visionimagevideoediting/segmentation/grabcut.htm

Table 7.1-7.2 show the error rates for different set of experiments. What is surprising is that the mean error using only one iteration of graph cuts is less than the error iterating until convergence. A possible reason is that some of images in the testing dataset are not suitable to our task, for example, landscape images, and more iterations will make the segmentation results worse. However, reasonable segmentations are obtained for most of the images .

7.2.3 Experimental Results

The experiments in this work can be viewed from different angles, for example, foreground detection and segmentation error rates, segmentation from different graph cuts iteration, and energy convergence. We display some typical results in each case and compare our results with the counterpart from related works. In addition, some fail examples are shown at the end.

- **Foreground Detection and Segmentation Error**



Pixel error: 0.0573, Class error: 0.0430

Pixel error: 0.0194, Class error: 0.0203

Pixel error: 0.0542, Class error: 0.0825

Pixel error: 0.4037, Class error: 0.3319

Pixel error: 0.3725, Class error: 0.2157

Pixel error: 0.0450, Class error: 0.0654

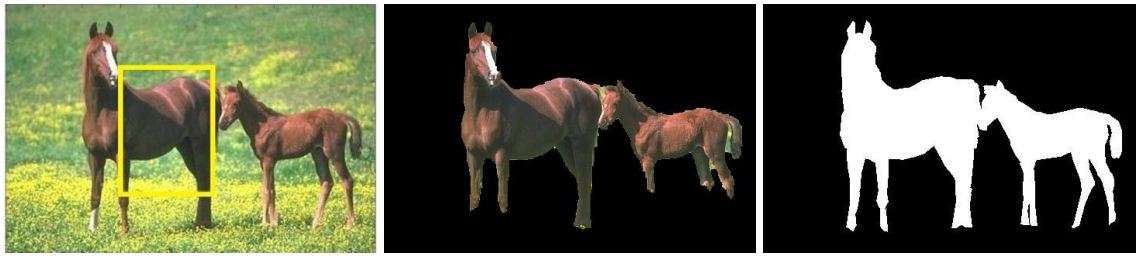
Pixel error: 0.0516, Class error: 0.0630

Pixel error: 0.0406, Class error: 0.0446

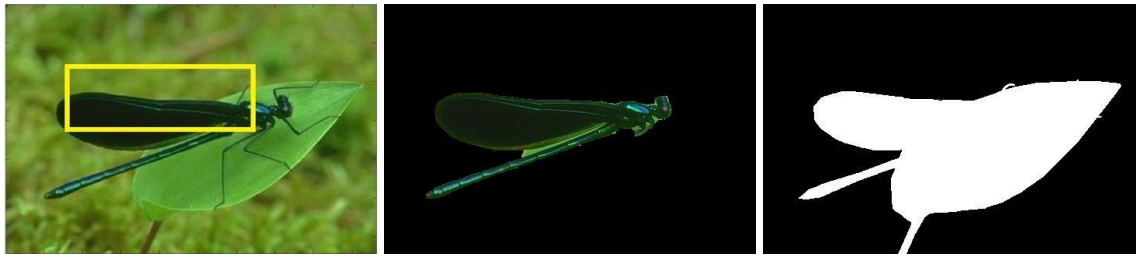
Pixel error: 0.0372, Class error: 0.0803

Pixel error: 0.1264, Class error: 0.0869

Pixel error: 0.0194, Class error: 0.0150

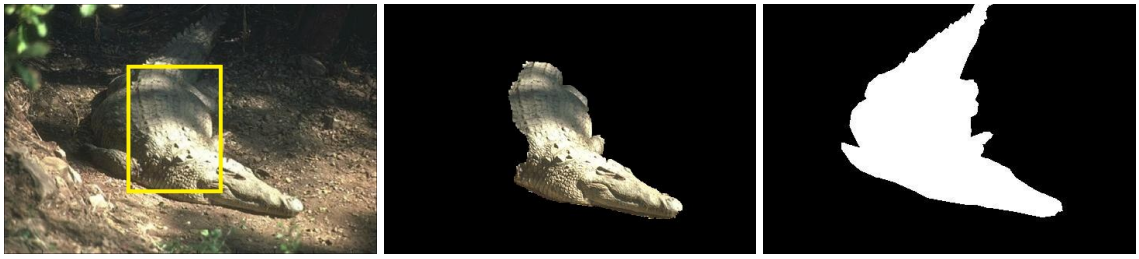


Pixel error: 0.0336, Class error: 0.0337



Pixel error: 0.1565, Class error: 0.2792

Pixel error: 0.0410, Class error: 0.0617



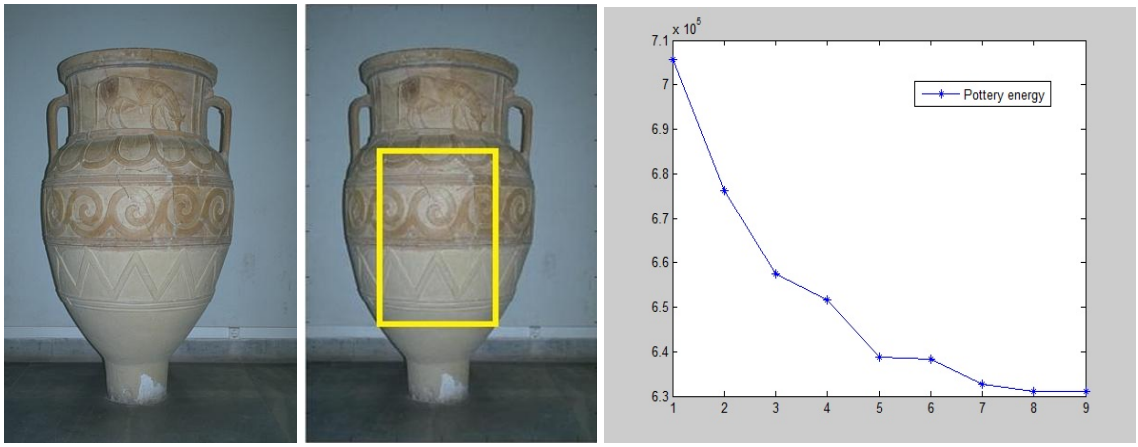
Pixel error: 0.0793, Class error: 0.1721

- **Iteration and Energy**
  As explained in chapter 6, the goal of iterative graph cuts is to improve the segmentation. Judging by the testing results, most of the images can be well segmented within 5 iterations.

  Figure 7.11-7.13 illustrate the process of segmentation refinement, which are segmentation results from iteration 1 to 5. The black and white images show ground truth. The energies at every iteration step are recorded, too.

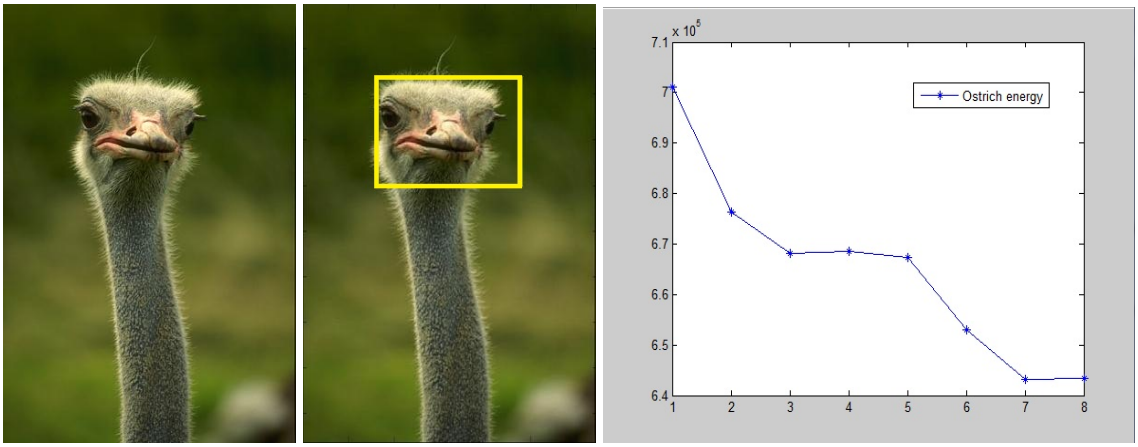
Left: original flower image, middle: foreground detection, right: the convergence process of energy

Figure 7.11: Segmentation results in iteration 1 to 5, compared with ground truth



Left: original pottey image, middle: foreground detection, right: the convergence process of energy

Figure 7.12: Segmentation results in iteration 1 to 5, compared with ground truth



Left: original ostrich image, middle: foreground detection, right: the convergence process of energy

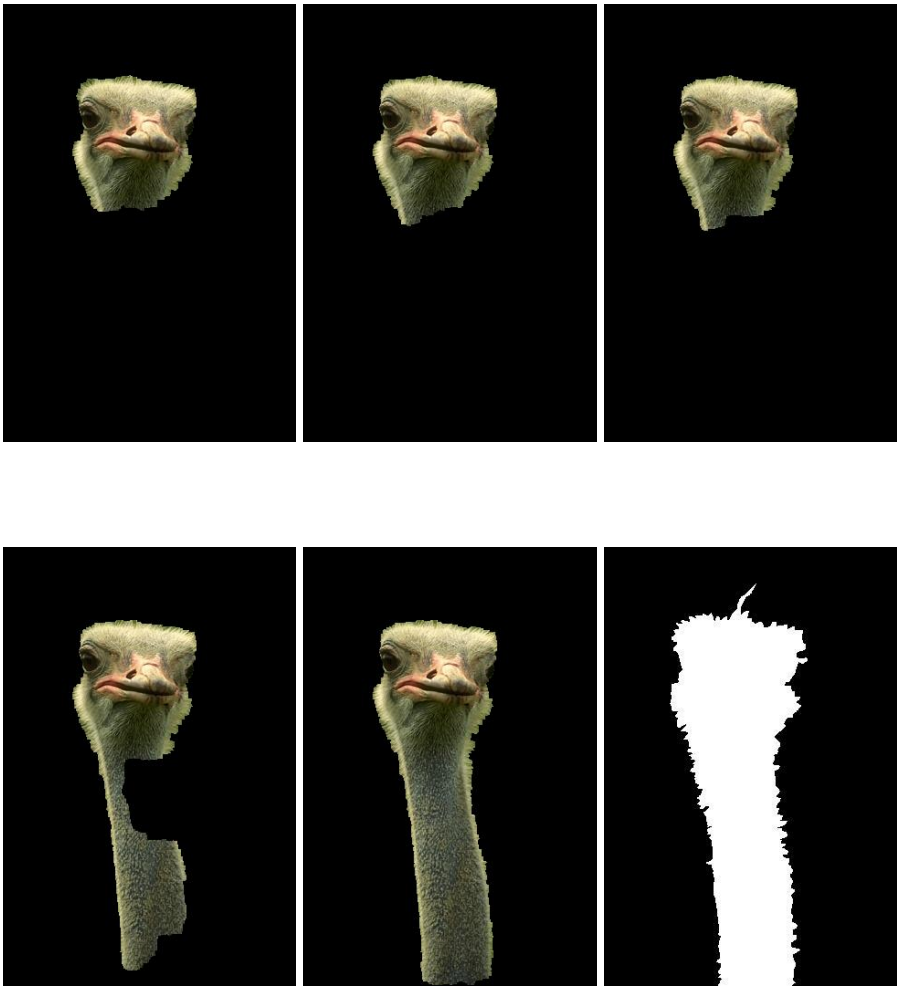


Figure 7.13: Segmentation results in iteration 1 to 5, compared with ground truth

- **Comparison with Grabcut Results**

  In figure 7.14-7.17, our results are compared with the counterparts from paper [62], it is not difficult to find that our algorithm can segment as well as grabcut without user guidance.

Figure 7.14: Left: grabcut initialization rectangle, middle: grabcut segmentation result, right: our result

Figure 7.15: Left: grabcut initialization rectangle, middle: grabcut segmentation result, right: our result

Figure 7.16: Left: grabcut initialization rectangle, middle: grabcut segmentation result, right: our result

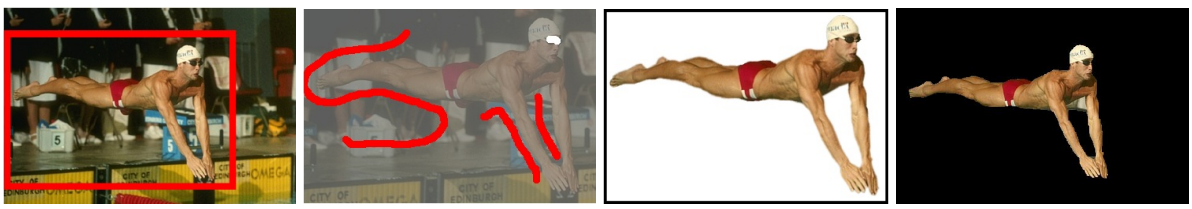


Figure 7.17: From left to right: grabcut initialization rectangle, grabcut user editing, grabcut segmentation result and our result

- **Comparison with Saliency Segmentation Results**
  We also compare our algorithm with that of Mehrani [56], who initialize graph cuts with salient object detection trained on manual labeled dataset. In both BSD and ASD, our experimental results reach the similar level to Mehrani's in error rates.

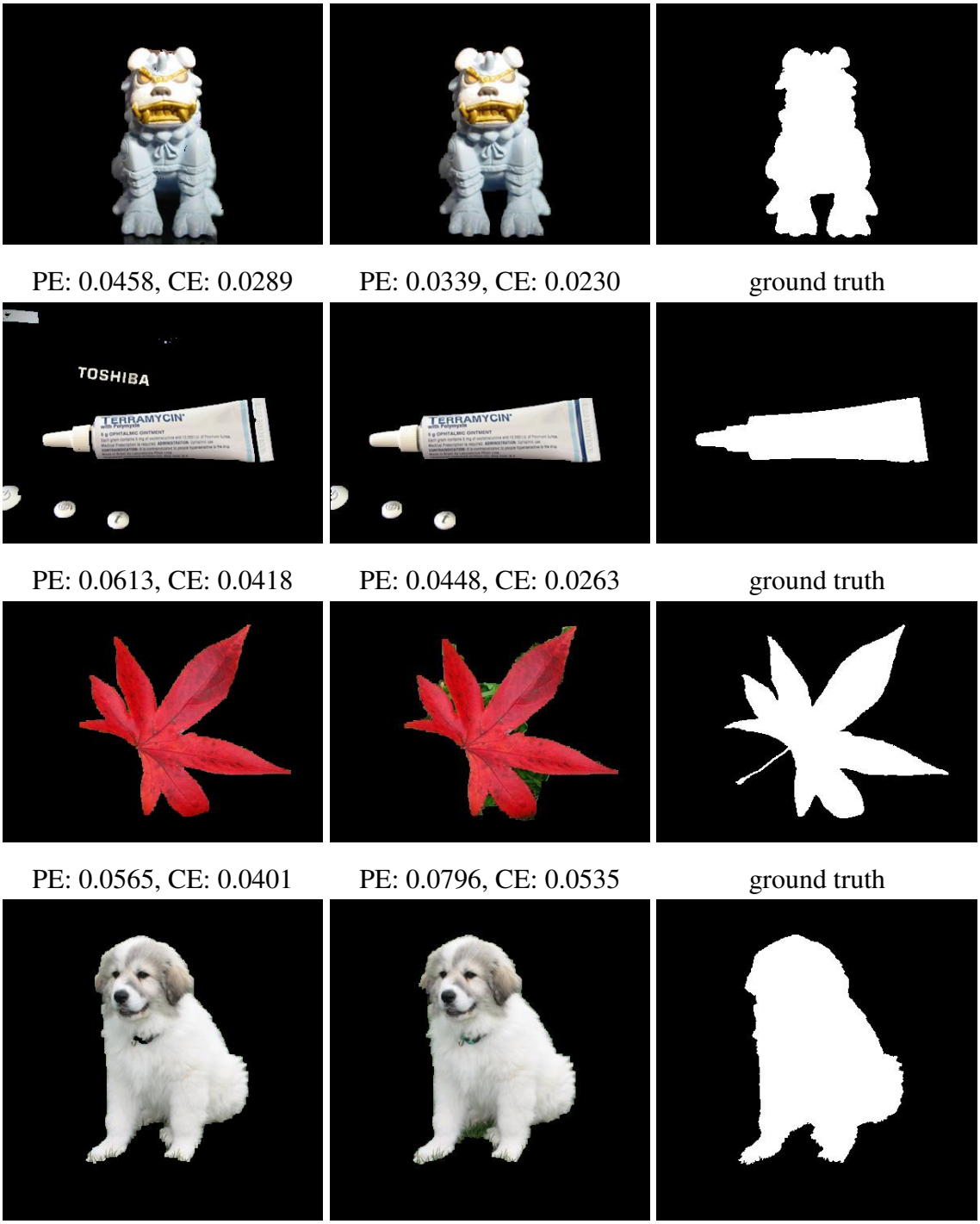


Figure 7.18: Column1: Mehrani's results, column 2: our results, column 3: ground truth

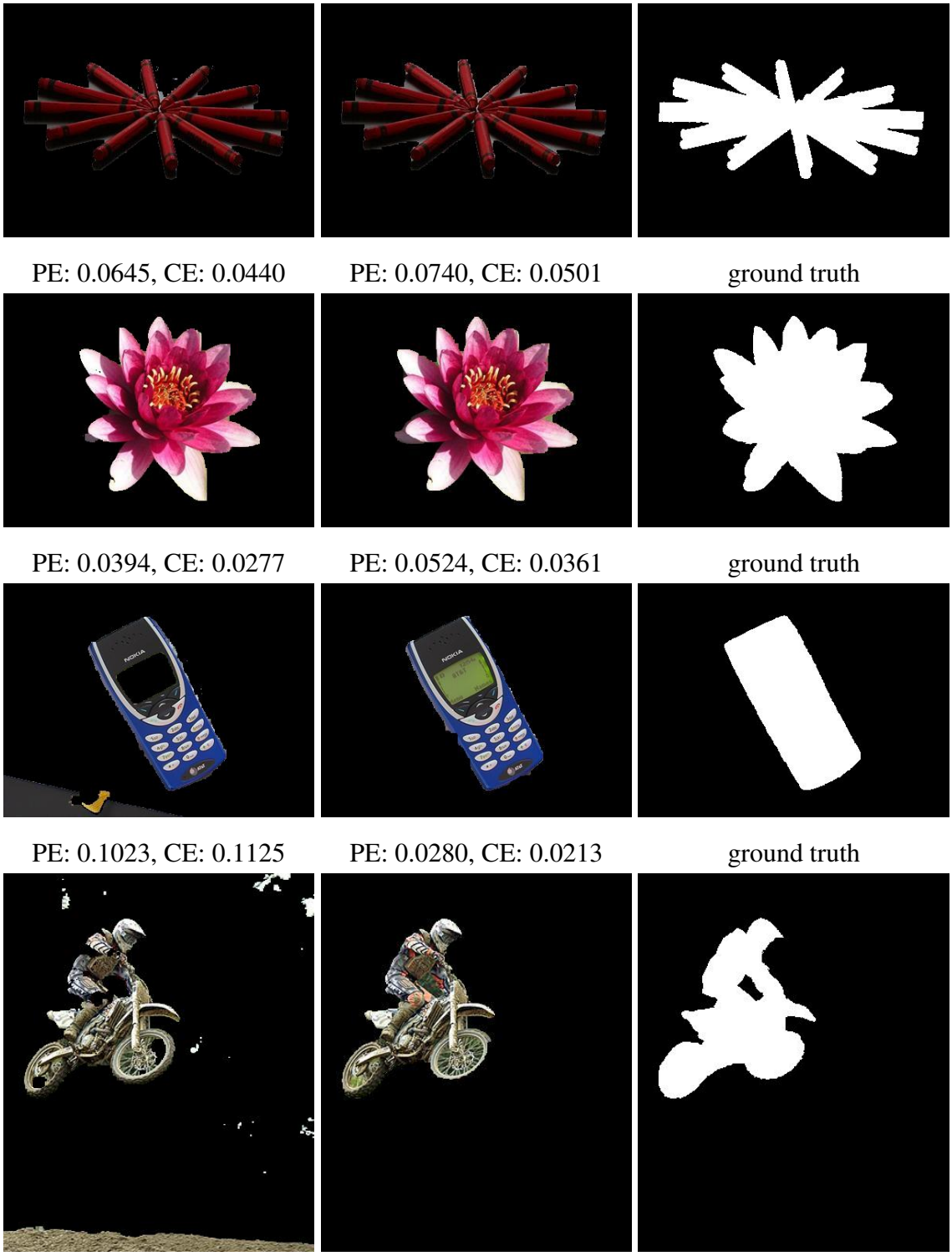


Figure 7.19: Column1: Mehrani's results, column 2: our results, column 3: ground truth

PE: 0.0128, CE: 0.0139 PE: 0.0141, CE: 0.0094 ground truth

PE: 0.0342, CE: 0.0229 PE: 0.0267, CE: 0.0179 ground truth

PE: 0.0464, CE: 0.0293 PE: 0.0476, CE: 0.0389 ground truth

PE: 0.1424, CE: 0.1819 PE: 0.0348, CE: 0.0338 ground truth

Figure 7.20: Column1: Mehrani's results, column 2: our results, column 3: ground truth

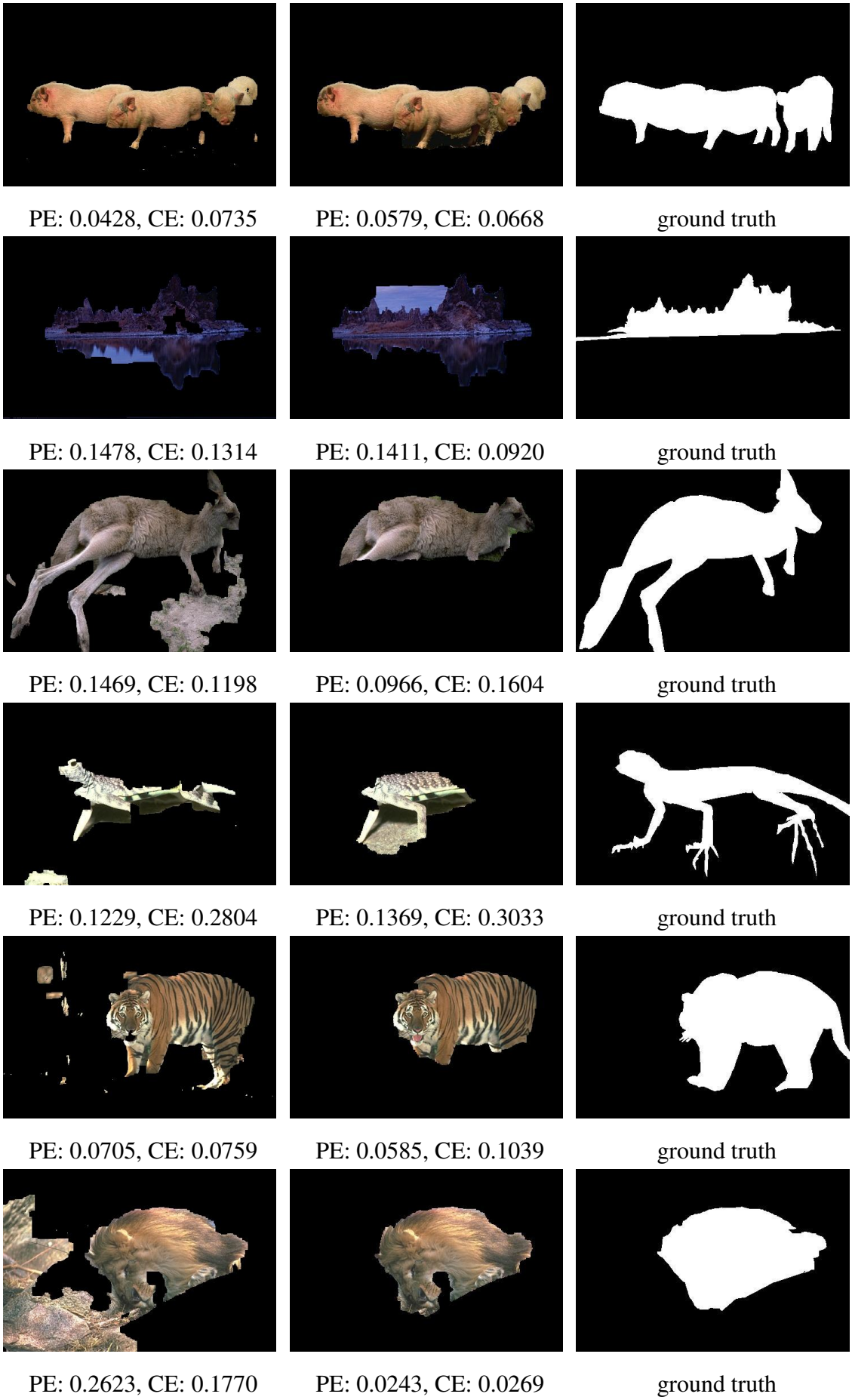
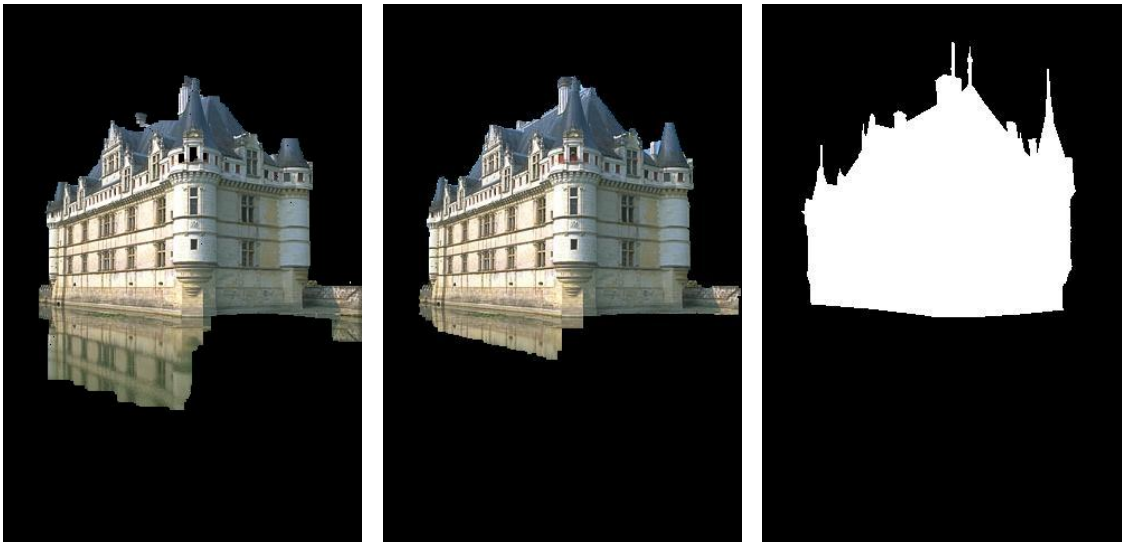


Figure 7.21: Column1: Mehrani's results, column 2: our results, column 3: ground truth

PE: 0.3835, CE: 0.2896 PE: 0.2341, CE: 0.2968 ground truth

PE: 0.2737, CE: 0.2262 PE: 0.0852, CE: 0.0670 ground truth



PE: 0.1051, CE: 0.0778 PE: 0.0605, CE: 0.0446 ground truth

Figure 7.22: Column1: Mehrani's results, column 2: our results, column 3: ground truth

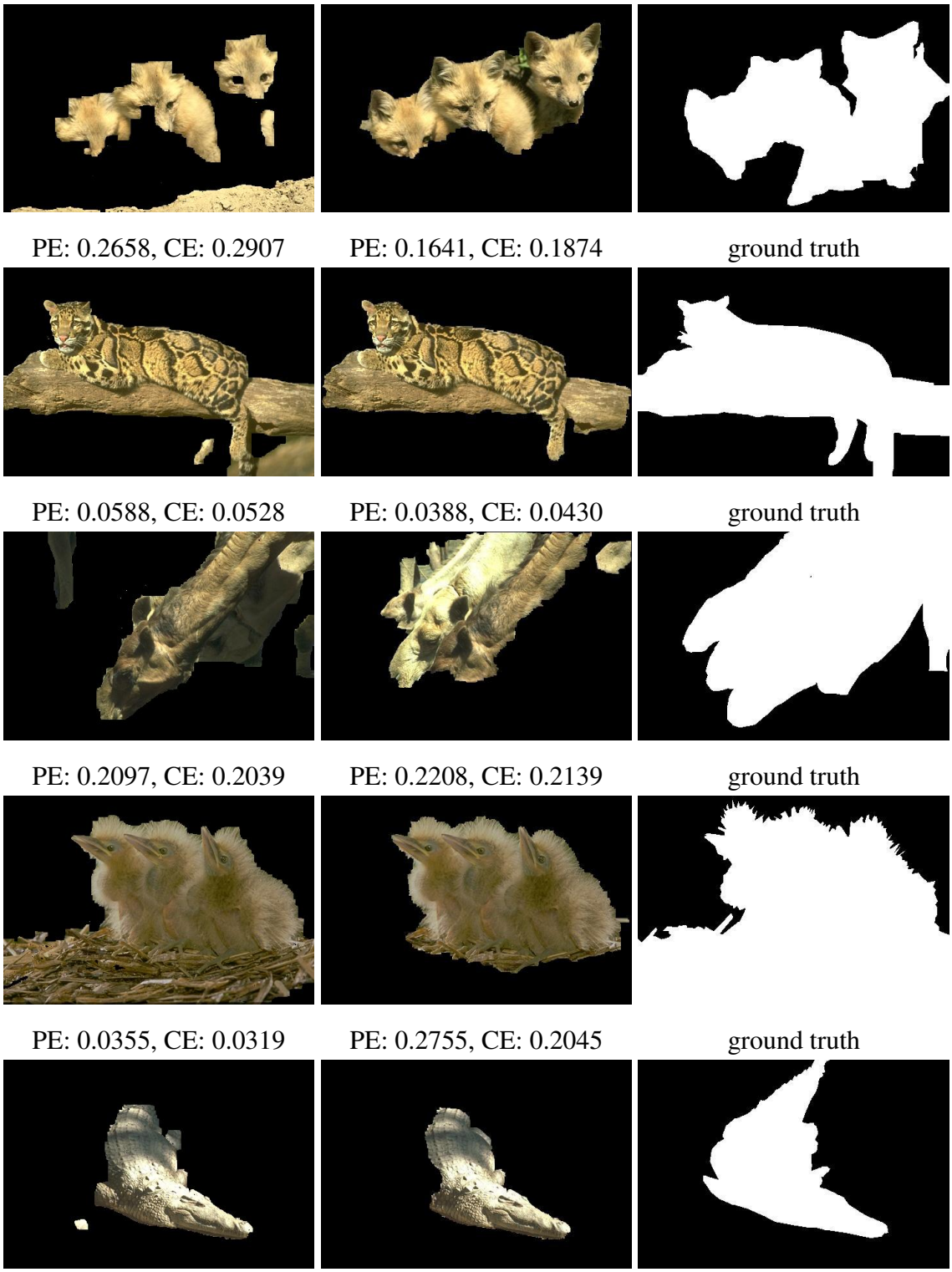


Figure 7.23: Column1: Mehrani's results, column 2: our results, column 3: ground truth

- **Failure Cases**
  Of course, our algorithm cannot automatically segment all image types, here we list some failure cases in foreground detection stage and segmentation stage.

  The proposed method tends to fail in the cases of landscape, and very confusing animal camouflage images. When the foreground object is too small or too sparse, the result is not satisfying.

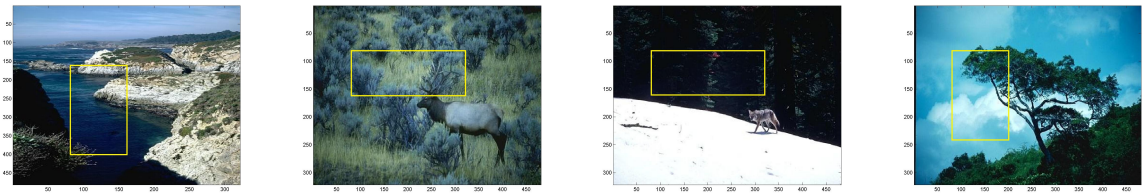


Figure 7.24: Failure detection examples, from left to right: landscape, animal camouflage, too small object and too sparse object

There are also some examples that succeed in foreground rectangle detection, but fail in segmentation stage. This is not surprising, because in the segmentation algorithm, we only use color information without texture, which means texture is also a very useful image feature when doing segmentation.

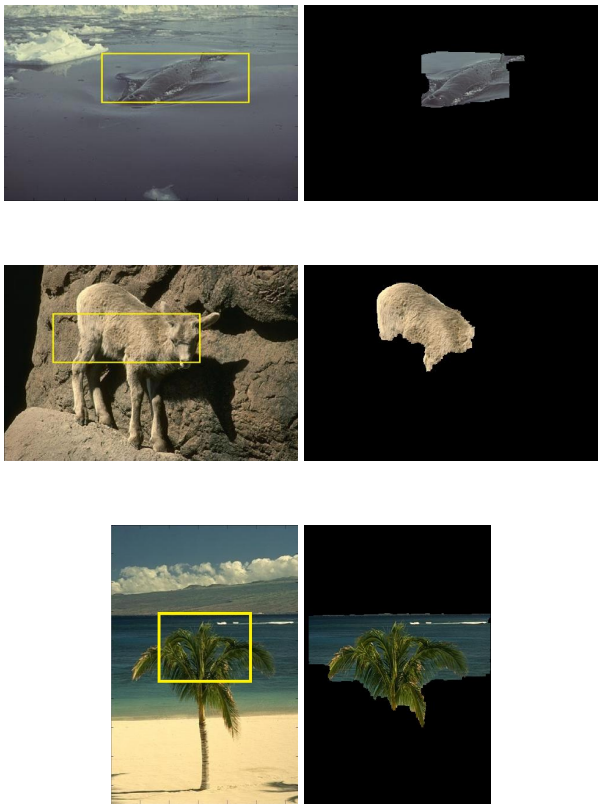


Figure 7.25: Left: successful rectangle detection, right: segmentation failure

Chapter 8

Conclusion and Future Work

To a large extent, scientific research makes progress with the guidance of the demand in practice. Automatic segmentation is desirable in many fields in real life, such as, “Google glass”, and “machine vision”, to name just a few. Therefore, our work targets to explore a new approach to implement automatic segmentation by giving graph cuts algorithm a proper initialization.

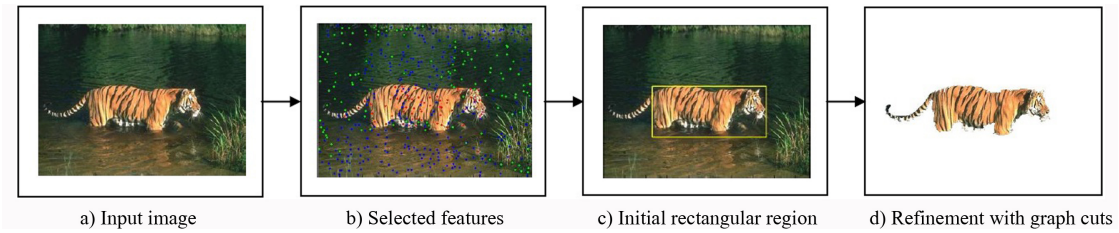


Figure 8.1: Illustration of proposed workflow

8.1 Summary

In order to fulfill our goal, we design an algorithm to compare features inside and outside the sliding window. The sliding window will stop at the position where the maximum difference shows up. And this dissimilarity is measured using α-divergence of foreground/background histograms.

Before the phase of searching the rectangle containing the foreground, some preparation are needed, they are feature extraction, feature clustering and feature selection. In this work, two feature types, texture and color are combined in equal percentage (50%), that is, we think the two features are equally important. After feature extraction, we employ k-means to cluster patch based texture vectors, while image quantization technique is used to quantize RGB color vectors. A ranking function is defined to select the texture features.

In the stage of segmentation, iterative graph cuts are used to minimize the energy function. This energy function addresses both the boundary and regional properties. In the first round

graph cuts, the initial data terms are set based on the rectangle segmentation acquired from the previous phase, and in the following rounds of graph cuts, the data terms are computed based on the last labeling. This iteration continues until the energy has converged, and during this process, the segmentation result is updated.

The experimental results of our algorithm are exciting, most of them reach the similar level compared with the results from grabcut (initial foreground rectangle by the user) and saliency detection (training saliency map with hand labeled dataset). To our knowledge, this is the first approach that initializes graph cuts automatically, without pre-training on a labelled dataset, just from the image content itself.

8.2 Future Work

Although the approach presented in this work has many advantages compared with the previous works, as explained in chapter 1, there are still some issues to investigate, which are listed as follows:

- **Foreground detection phase**

  As explained through the thesis, using informative features will result in better detection, so it seems promising to test more feature types and construct informative feature vector. Another aspect that deserves investigation is testing different clustering algorithms to find a more stable one.

- **Segmentation stage**

  From the animal camouflage images, we know that the foreground detection algorithm can succeed in many case, but it is a pity that their final segmentation results are not satisfying. This is not surprising because we only use the color mode to refine the segments by graph cuts. Learning both color and texture modes may lead to better results in animal camouflage images.

- **Accuracy and efficiency**

  Considering the application in real life, segmentation efficiency is an aspect that should not be ignored. We can increase the speed of our method by rewriting the foreground detection algorithm in C++ instead of implementing in MATLAB, and parallelizing the foreground searching part. At the same time, in parallel computing case, we can compare more sliding window sizes and shapes to get more precise foreground location, so as to acquire more accurate segmentation results. Notice that searching in parallel will not reduce the efficiency as long as we were supplied with enough processor.

- **Algorithm extension**

  The feature dissimilarity detection idea can be extended to higher dimensional segmentation, for example, volume segmentation in 3D space. Though the situation in higher

dimension should be more complicated, extending our algorithm from pixel features to voxel features is feasible and might be useful in video analysis or medical volume analysis.

In conclusion, the proposed method generates good foreground location detection and segmentation results, and it can help computers to automatically understand natural images better. Meanwhile, more strategies are waiting for exploration to improve the algorithm, extension to N dimensional segmentation is expected, too.

Bibliography

[1] R. Achanta, F. Estrada, P. Wils, and S. Ssstrunk. Salient region detection and segmentation. *In International Conference on Computer Vision Systems*, pages 66–75, 2008.

[2] S. Amari and H Nagaoka. *Methods of Information Geometry*. Oxford University Press: New York, NY, USA, 2000.

[3] A. Amini, T. Weymouth, and R. Jain. Using dynamic programming for solving variational problems in vision. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 12(9):855–867, 1990.

[4] S. Barnard. Stochastic stereo matching over scale. *IJCV*, 3(1):17–32, 1989.

[5] Luca Bertelli, Tianli Yu, Diem Vu, and Burak Gokturk. Kernelized structural svm learning for supervised object segmentation. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2153–2160, 2011.

[6] S. Birchfield and C. Tomasi. Multiway cut for stereo and motion with slanted surfaces. *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 489–495, 1999.

[7] Y. Boykov and M.-P. Jolly. Interactive organ segmentation using graph cuts. *Proc. Medical Image Computing and Computer-Assisted Intervention*, pages 276–286, 2000.

[8] Y. Boykov and M.P Jolly. Interactive graph cuts for optimal boundary and region segmentation. *ICCV*, I:105–112, 2001.

[9] Y. Boykov and V. Kolmogorov. Computing geodesics and minimal surfaces via graph cuts. *Proc. Intl Conf. Computer Vision*, pages 26–33, 2003.

[10] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *PAMI*, 26:1124–1137, 2004.

[11] Y. Boykov, O. Veksler, and R. Zabih. Markov random fields with efficient approximations. *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 648–655, 1998.

[12] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on PAMI*, 23(11):1222–1239, 2001.

[13] M. Casares, A. Almagambetov, and S. Velipasalar. A robust algorithm for the detection of vehicle turn signals and brake lights. *Advanced Video and Signal-Based Surveillance (AVSS)*, pages 386–391, 2012.

[14] Xiaohan Chen and N.A. Schmid. Empirical capacity of a recognition channel for single- and multipose object recognition under the constraint of pca encoding. *Image Processing, IEEE Transactions*, 18:635–651, 2009.

[15] H Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on a sum of observations. *Ann. Math. Statist*, 23:493–507, 1952.

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, USA, 2001.

[17] Andrew Delong. *Advances in graph-cut optimization: multi-surface models, label costs, and hierarchical costs*. PhD thesis, The University of Western Ontario, 2011.

[18] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, Series B (Methodological) 39(1):1–38, 1977.

[19] Y. Deng and B. S.Manjunath. Unsupervised segmentation of color-texture regions in images and video. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI '01)*, 23(8):800–810, 2001.

[20] A. Efros and T. Leung. Texture synthesis by non-parametric sampling. *In Proceedings of the International Conference on Computer Vision*, 2:1039–1046, 1999.

[21] A.X. Falcao, J.K. Udupa, S. Samarasekera, S. Sharma, B.E. Hirsch, and R. de A. Lotufo. User-steered image segmentation paradigms: Live-wire and live-lane. *Graphical Models and Image Processing*, 60(4):233–260, 1998.

[22] L. Ford and D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[23] Fukunaga, Keinosuke, and Larry D. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1):32–40, 1975.

[24] J. Garcia-Consuegra, J. Cisneros, G.and Ballesteros, and R. Molina. Remote sensing segmentation through a filter bank based on gabor functions. *Acoustics, Speech and Signal Processing*, 2:1169–1171, 1998.

[25] Stas Goferman and Lihi Zelnik-Manor. Context-aware saliency detection. *CVPR*, pages 2376–2383, 2010.

[26] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *In Proceedings of the eighteenth annual ACM symposium on Theory of computing, STOC' 86*, pages 136–146, 1986.

[27] Rafael C. Gonzalez and Richard E Woods. *Digital Image Processing*. Publishing house of Electronics Industry, 2002.

[28] D. Greig, B. Porteous, and A. Seheult. Exact maximum a posteriori estimation for binary images. *J. Royal Statistical Soc., Series B*, 51(2):271–279, 1989.

[29] D. Han, X. Lu W. Li, T. Wang, and Y. Wang. Automatic segmentation based on adaboost learning and graph-cuts. *In Proc. ICIAR*, pages 215–225, 2006.

[30] R. M. Haralick and L. G. Shapiro. *Computer and Robot Vision*. Addison-Wesley Publishing Company, 1992.

[31] Paul S. Heckbert. Color image quantization for frame buffer display. *Computer Graphics*, 16(3):297–303, 1982.

[32] Xiaodi Hou and Liqing Zhang. Saliency detection: A spectral residual approach. *CVPR*, pages 1–8, 2007.

[33] H. Ishikawa and D. Geiger. Occlusions, discontinuities, and epipolar lines in stereo. *Proc. European Conf. Computer Vision*, pages 232–248, 1998.

[34] H. Ishikawa and D. Geiger. Segmentation by grouping junctions. *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 125–131, 1998.

[35] Chanho Jung, Beomjoon Kim, and Changick Kim. Automatic segmentation of salient objects using iterative reversible graph cut. *ICME*, pages 590–595, 2010.

[36] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, pages 321–331, 1988.

[37] J. Kim, J. Fisher, A. Tsai, C. Wible, A. Willsky, and W. Wells. Incorporating spatial priors into an information theoretic approach for fmri data analysis. *Proc. Medical Image Computing and Computer-Assisted Intervention*, pages 62–71, 2000.

[38] J. Kim, V. Kolmogorov, and R. Zabih. Visual correspondence using energy minimization and mutual information. *Proc. Intl Conf. Computer Vision*, pages 1033–1040, 2003.

[39] J. Kim and R. Zabih. Automatic segmentation of contrast- enhanced image sequences. *Proc. Intl Conf. Computer Vision*, pages 502–509, 2003.

[40] Sung Hoon Kim, Hyon Soo Lee, and Hyung Ho Kim. Robust extraction of face candidate through segmentation and conditional merging in skin area. *ICIS*, 1:547–551, 2009.

[41] V. Kolmogorov and R. Zabih. Visual correspondence with occlusions using graph cuts. *Proc. Intl Conf. Computer Vision*, pages 508–515, 2001.

[42] V. Kolmogorov and R. Zabih. Multi-camera scene reconstruction via graph cuts. *Proc. European Conf. Computer Vision*, 3:82–96, 2002.

[43] V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts? *Pattern Analysis and Machine Intelligence, IEEE Transactions*, 26(2):147–159, 2004.

[44] Marcel Krcah, Gabor Szekely, and Remi Blanc. Fully automatic and fast segmentation of the femur bone from 3D-CT images with no shape prior. *Biomedical Imaging: From Nano to Macro*, pages 2087–2090, 2011.

[45] V. Kwatra, A. Scho dl, G. Turk I. Essa, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Trans. Graphics, Proc. SIGGRAPH 2003*, 2003.

[46] Xianpeng Lang, Feng Zhu, Yingming Hao, and Qingxiao Wu. Automatic image segmentation incorporating shape priors via graph cuts. *Proceedings of IEEE International Conference on Information and Automation*, pages 192–195, 2009.

[47] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43(1):29–44, 2001.

[48] Ran Li, Weiguang Xu, Jianjiang Lu, Yafei Zhang, and Zining Lu. Technique of large-scale image set construction based on web image searching engine. *ICIS*, pages 622–626, 2009.

[49] Y. Li, J. Sun, C-K. Tang, and H-Y. Shum. Lazy snapping. *ACM Transaction on Graphics*, 23(3), 2004.

[50] M.H. Lin. *Surfaces with Occlusions from Layered Stereo*. PhD thesis, Stanford Univ., 2002.

[51] Jiangyu Liu, Jian Sun, and Heung-Yeung Shum. Paint selection. *siggraph*, 2009.

[52] Yu Liu. *Classic Mosaics and Visual Correspondence via Graph-Cut based Energy Optimization*. PhD thesis, The University of Western Ontario, 2011.

[53] David G Lowe. Object recognition from local scale-invariant features. *Proceedings of the International Conference on Computer Vision*, pages 1150–1157, 1999.

[54] 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 International Conference Computer Vision*, 2:416–423, 2001.

[55] Laurent Massoptier and Sergio Casciaro. Fully automatic liver segmentation through graph-cut technique. *Proceedings of the 29th Annual International Conference of the IEEE EMBS*, pages 5243–5246.

[56] Paria Mehrani. *Automatic Salient Object Detection and Segmentation*. MSc thesis, The University of Western Ontario, 2010.

[57] Paria Mehrani and Olga Veksler. Saliency segmentation based on learning and graph cut refinement. *BMVC*, pages 1–12, 2010.

[58] E.N. Mortensen and W.A. Barrett. Intelligent scissors for image composition. *Proc. Of ACM Siggraph*, pages 191–198, 1995.

[59] M. Narasimhan and J. Bilmes. A supermodular-submodular procedure with applications to discriminative structure learning. *UAI*, 2005.

[60] S. Osher and N. Paragios. *Geometric Level Set Methods in Imaging Vision and Graphics*. Springer Verlag, 2003.

[61] Bo Peng and Lei Zhang. A survey of graph theoretical approaches to image segmentation. *http://www.sciencedirect.com/science/article/pii/S0031320312004219*.

[62] C. Rother, V. Kolmogorov, and A. Blake. grabcut - interactive foreground extraction using iterated graph cuts. *Proc. of ACM, SIGGRAPH*, I:309–314, 2004.

[63] C. Rother, T. Minka, A. Blake, and V. Kolmogorov. Cosegmentation of image pairs by histogram matching - incorporating a global constraint into mrfs. *IEEE Computer Society Conference on In Computer Vision and Pattern Recognition*, 1:993–1000, 2006.

[64] S. Roy. Stereo without epipolar lines: A maximum flow formulation. *IJCV*, 1(2):1–15, 1999.

[65] S. Roy and I. Cox. A maximum-flow formulation of the n-camera stereo correspondence problem. *Sixth International Conference on Computer Vision*, pages 492–499, 1998.

[66] Martin Sewell. Feature selection. *http://machine-learning.martinsewell.com/feature-selection/feature-selection.pdf*, 2007.

[67] Linda G. Shapiro and George C. Stockman. *Computer Vision*. New Jersey, Prentice-Hall, 2001.

[68] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *Computer Vision and Pattern Recognition*, pages 731–737, 1997.

[69] D. Snow, P. Viola, and R. Zabih. Exact voxel occupancy with graph cuts. *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 3:345–352, 2000.

[70] K.and Yimmun S. Suapang, P.and Dejhan. A web-based dicom-format image archive, medical image compression and dicom viewer system for teleradiology application. *SICE Annual Conference*, pages 3005–3011, 2010.

[71] Antonio Torralba and Alexei A. Efros. Unbiased look at dataset bias. *CVPR*, pages 1521–1528, 2011.

[72] M. Varma and A Zisserman. Texture classification: Are filter banks necessary? *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2:II–691–8, 2003.

[73] M. Varma and A. Zisserman. A statistical approach to material classification using image patch exemplars. *In Proceedings of the International Conference on Computer Vision*, 31:2032–2047, 2009.

[74] Olga Veksler. Star shape prior for graph-cut image segmentation. *European Conference on Computer Vision*, pages 454–467, 2008.

[75] L. Vincent and P. Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI*, 13:583–598, 1991.

[76] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1:511–518, 2001.

[77] Peng Wang. https://mywebspace.wisc.edu/pwang6/personal/.

[78] Thomas Weise. *Global Optimization Algorithms Theory and Application*. http://www.it-weise.de/projects/book.pdf, 2009.

Curriculum Vitae

Name: Wei Li

Education:

University of Western Ontario
London, Ontario
M.Sc. (Computer Science), expected, December 2012

Wuhan University
Wuhan, China
M.Sc. (Computational Mathematics), received, June 2006

Wuhan University
Wuhan, China
B.Sc. (Information and Computational Science), received, June 2004
B.A. (Finance), received, June 2004

Related Work Experience:

Research Assistant
Department of Computer Science,
The University of Western Ontario, London, Ontario
September 2011 - December 2012

Teaching Assistant
Department of Computer Science,
The University of Western Ontario, London, Ontario
September 2011 - April 2012, September 2012 - December 2012

Research Assistant
Supercomputing Center, Chinese Academy of Sciences
Beijing, China
July 2006 - July 2009