LIVER BIOPSY IMAGE PROCESSING

PAWESUDA SUMITPAIBUL
ANURAK DAMRONGPHITHAKKUL

BACHELOR OF ENGINEERING PROGRAM IN SOFTWARE ENGINEERING
INTERNATIONAL COLLEGE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2012

Liver Biopsy Image Processing

Ms.Pawesuda Sumitpaibul 52090017

Mr. Anurak Damrongphithakkul 52090022

Dr. Vorapranee Khu-smith Advisor

Dr. Ukrit Watchareeruetai Co-Advisor

Academic Year 2012

ABSTRACT

This project presents a liver image processing with possibility of developing software to play a role in liver biopsy image diagnosis. Diagnosing liver biopsy images is important to prognosis and treatment planning of pathologists and specialized doctors. Non-alcoholic fatty liver disease (NAFLD) is one cause of a fatty liver and slowly progressive liver disease that may lead to liver cancer. Nevertheless, the pathologists need to confirm NAFLD. By viewing under microscopic images of biopsy slides, pathologists can determine the severity of NAFLD and group the patients by range of the amount of fat occurred in the liver. Different patient group gets different treatment. Thereby, the estimation of amount of fat is crucial.

However, before we start to research, study and develop software that relevant to the liver biopsy image diagnosing, we started by meeting with the pathologist team at Rama Hospital to discuss the problem and motivation of the project. Then we collected the user and software requirements from the pathologists e.g. what they want the software to look like, how user-friendly it should be, and especially how accurate the diagnosis result should come out.

Acknowledgements

It is a great pleasure for us to acknowledge who have contributed towards the conception of this project.

We would like to express our most sincere thanks to our project advisors, Dr. Vorapranee Khu-smith and Dr. Ukrit Watchareeruetai for helping throughout the semester. Their wide knowledge and their logical way of thinking have been of great value for us. Their understanding, encouraging and personal guidance have provided a good basis for our project.

Very special thanks to Asst. Prof. Pattana Sornmayura and M.L. Taya Kitiyakara from Rama Hospital, for providing us an opportunity to participate in this project and all information throughout the semester.Also, our grateful thanks to our department which provides us an opportunity as a project subject to develop a report work skill in software project II.

We would also like to extend our thanks to our classmates for all the support and motivation in all our efforts during the semesters, especially Mr.Kiatkachorn Saripan who has been there with a helping hand with his ability throughout the period of our work.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

The purpose of this thesis is to present a detailed description of the "Liver biopsy image processing" or liver biopsy image segmentation system. It will explain the purpose and features of the system, limitations and constraints under which it must operate. This document is intended for both the stakeholders and the developers of the system and will be proposed to the committee of Software Project II for approval.

The project involves an image processing studies and possibility of developing software to play a part in the diagnosis of liver biopsy image. There are a number of studies in image processing techniques such as image segmentation, image enhancement using different algorithms to analyze the images etc. In practice, several techniques are combined in this project to complement each other.

The following is the project scenario under which our research was conducted.

A patient who may have liver disease is asked to have a liver biopsy which is a procedure involving putting a needle through the skin and into the liver to get sample tissue from the liver. A liver biopsy procedure typically begins with a radiologist, using ultrasound or CT scans to guide the placement of the needle, the doctor then slowly advances the needle to obtain liver sample tissue. The tissue is then sent to a lab for pathologist to analyze. The tissue is usually cut, dyed, and put onto a slide which is finally looked under a microscope for diagnosis. An image of the slide can be taken with specialized equipment for further distribution and discussion.

Our project aims to research and study the possibility of using software as well as develop the software to help analyze the liver images taken from liver biopsy. The system will be designed to facilitate pathologists and doctors to diagnose liver biopsy images in particular to calculate the amount of fat presented in the sample liver tissue.

1.2 Motivation

Liver cancer is the fifth most common cancer in the world and the primary cause of cancer related deaths especially in Southeast Asia and tropical Africa. "Non-alcoholic fatty liver disease (NAFLD) is one cause of a fatty liver, occurring when fat is deposited (Steatosis) in the liver not due to excessive alcohol use"[1]. NAFLD is a slowly progressive liver disease that may ultimately lead to cirrhosis or cancer.

Imaging tests (such as ultrasound, CT scan, or MRI) may reveal fat accumulation in the liver but cannot differentiate NAFLD from other causes of liver disease that have a similar appearance. A liver biopsy is required to confirm NAFLD. The severity of NAFLD may be based on the following scoring. Please see Table 1-1.

Table 1-1: NAFLD grading

Score	Percentage	Description
0	<5%	Normal
1	5–33%	Mild fatty change
2	33–66%	Moderate fatty change
3	>66%	Severe fatty change

[Source: D.F.Chan, A.M.Li. Hepatic Steatosis in Obese Chinese Children]

This scoring is usually done by pathologists who estimate the amount of fat and give a score by a so-called eye-ball technique.

The eye-ball technique is used by specialized doctors/pathologists to estimate the amount of fat in the liver by examining microscopic tissue images. However, the amount of fat that specialized doctors estimate is commonly given as 1/3 and 2/3. The estimated amount of fat as 1/3 can then be categorized to the NAFLD score of 2, which considers the condition as moderate. The 2/3 group therefore has the score of 3 which is severe. The severity score grouping is very important especially in public hospitals since the patients will get different treatment if they are in a different severity group. To be specific, patients in severity group 2 will get a more effective treatment than group 3 since they are considered to have a better chance to be cured.

We now can see that the estimated amount of fat is very crucial and currently it is not performed in a precise manner. If specialized doctor classified patients into a wrong group that would be a risk.

Another risk for the patients stems from the fact that, in most public hospitals, the biopsy result is not usually available within a few days to a week after the biopsy since there are a lot of patients to be handled. These patients then have to wait a long time for the result. Because the disease is progressive, the longer the patients waited, the more severe the disease can be. There are very few specialized doctors who are trained to read biopsy slides. This makes the procedure of diagnosis take even longer time.

Moreover, because one specialized doctor need to handle many patients a day, this would cause them eye-strain and exhaustion. Understandably, the efficiency of their work may be reduced.

1.3 Objectives

From issues mentioned above, this project intends to develop software with possible usages to help doctors to estimate fat proportion in the liver. The percentage of fat would come out in a more precise number by image segmentation process. Generally, the software would segment the fat areas out from overall tissue then calculate percentage of fat presented, and compared it to the normal tissue.

The software would provide a semi-automatic usage for users. This means that the users could edit diagnosis manually by further image adjusting or choosing more fat areas to re-calculate when users did not satisfy with the initial results.

1.4 Scope of project

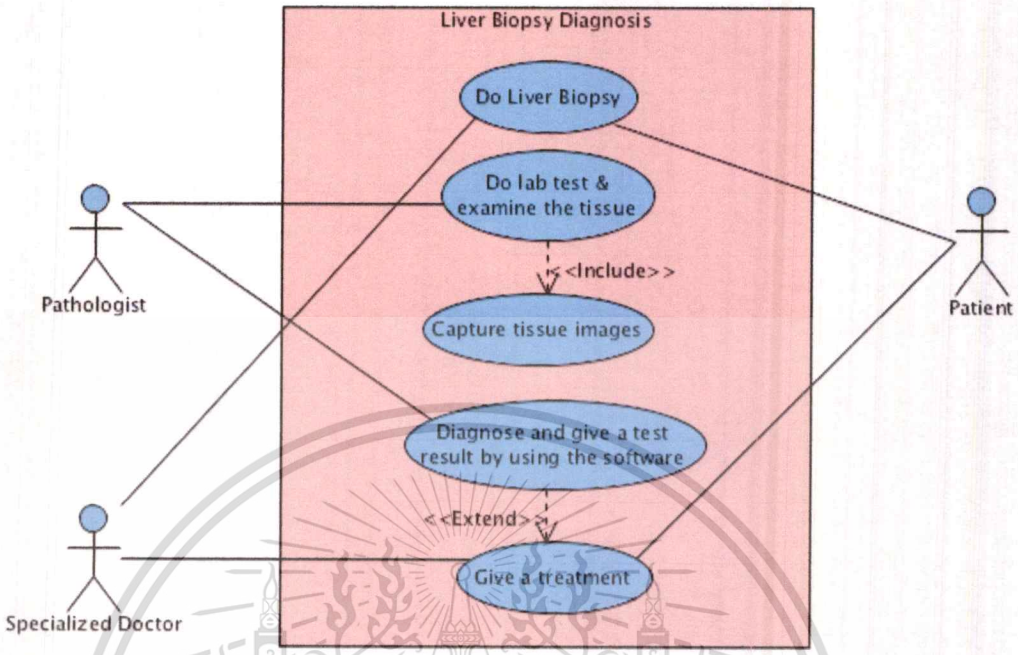


Figure 1-1: System environment

The scope of the project can be summarized in the system environment diagram(see in Figure 1-1). First, a liver biopsy is performed on a patient. Then a pathologist will take patient's tissue from the liver biopsy procedure to examine in the lab. For pathologist to examine the tissue, he will put the tissue slide into a scanner (microscope + camera) in order to capture the tissue image then get a microscopic digital image for the entire piece of tissue. After that, pathologist will use the software to help diagnose and give a test result. Then after the diagnosed result came out, if it is was not that good, the patient may get a treatment by a specialized doctor.

However, this project focuses mainly on a pathologist who diagnoses and gets a test result from liver biopsy images by using software. The category of biopsy image which pathologist diagnoses is NAFLD (described in Section 1.2). By doing this, the software does the tasks of image processing then calculates the amount of fat in a more precise way i.e. in percentage. It is worth noting that the software does not intend to replace the liver disease diagnosis done by specialists. It only plays a part in giving information regarding the amount of fat in the liver tissue.

1.5 Hierarchy of project activity

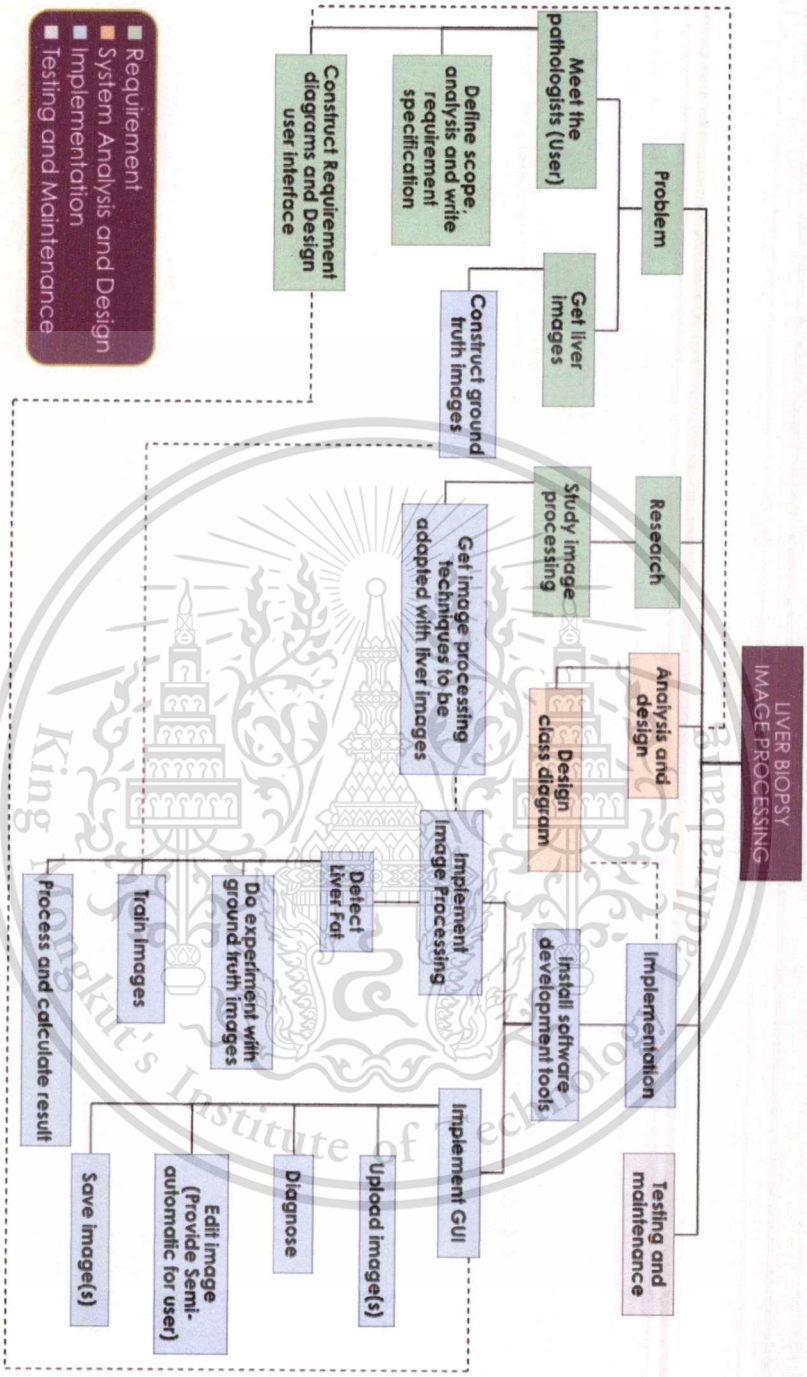


Figure 1-2: Hierarchy of Project Activity

The Hierarchy in Figure 1-2 briefly describes all the main tasks and activities carried out in the project. These can be divided into four phases: Requirements, System Analysis and Design, Implementation, and Testing and Maintenance. The dash lines represent the tasks that technically complement each other.

Starting with problem phase, the issues indicated by the pathologists from Rama Hospital, we obtained the requirements and defined the project scope. We then analyzed the requirements to create use case diagram, activity diagram, and design initial user interface. At the time that we received liver biopsy images from the pathologists, we began to construct ground truths for some of those images to create a set of sample images to be used in the implementation phase. In the meantime after we have obtained the requirements, we did research and study about image processing techniques and adapted the techniques to process sample images in the implementation phase. Next, we continued to design the software by creating class diagram. In the implementation phase, after we installed software development tools, we divided the phase into two main parts: Image Processing and User Interface. After implementation was finished, we did some experiments, test, and maintenance to verify and validate the finishing software.

1.6 Structure of Thesis

The rest of this document is organized as follows.

Chapter 2 describes problem statement and existing work, which are related to this project.

Chapter 3 describes background knowledge, which describes the theories that are relevant to the project. The use of these theories will be proved and demonstrated in development and implementation sections. (Chapter 6 and 7)

Chapter 4 describes system requirements, which describes overall requirements of the system, including use case and activity diagram.

Chapter 5 describes software design, which describes the functionalities and the design of each component of the system, including package and analysis diagram.

Chapter 6 describes system development, which describes techniques, algorithms and tools used throughout the project.

Chapter 7 describes system demonstration and results, which shows the implementation in a flow chart supported by implementation results.

Chapter 8 describes experimentation which shows the experimentation results from optimality test and ground truth image test.

Chapter 9 summarizes what have been done, and describes limitations of the software, lesson-learned and further work.

Chapter 2

Problem Description and Related Work

This section describes the problem description and existing work, which are related to this project.

2.1 Problem description

Our project motivation can be summarized into four major problems. First is the eye-ball technique, diagnosing and patient grouping are not performed in a precise manner (described in Section 1.2). Second, there is a few specialists who are trained to read biopsy slides or images. Third, diagnosis efficiency may be reduced because there are simply too many patients to handle. And fourth for some hospitals, patients have to wait for a long time for the diagnose result.

2.2 Related work

As a matter of fact, we could not find literature or software that are directly related to our requirements which is an effective segmentation of fat area in the liver. Nevertheless, we found a research that is similar to ours in term of image segmentation as follows.

2.2.1 Effective segmentation and classification for HCC biopsy images

Huang and Lai [5] proposed an effective segmentation of nuclei cells for grading the hepatocellular carcinoma (HCC) in liver biopsy images.

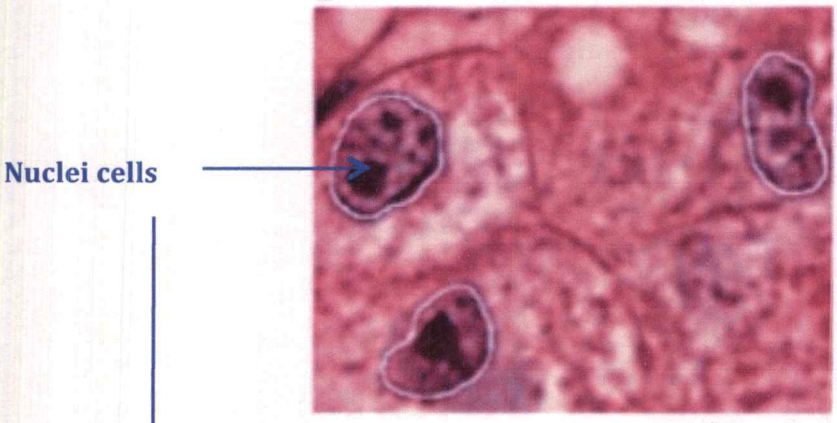


Figure 2-1: Nuclei cells contour and segment

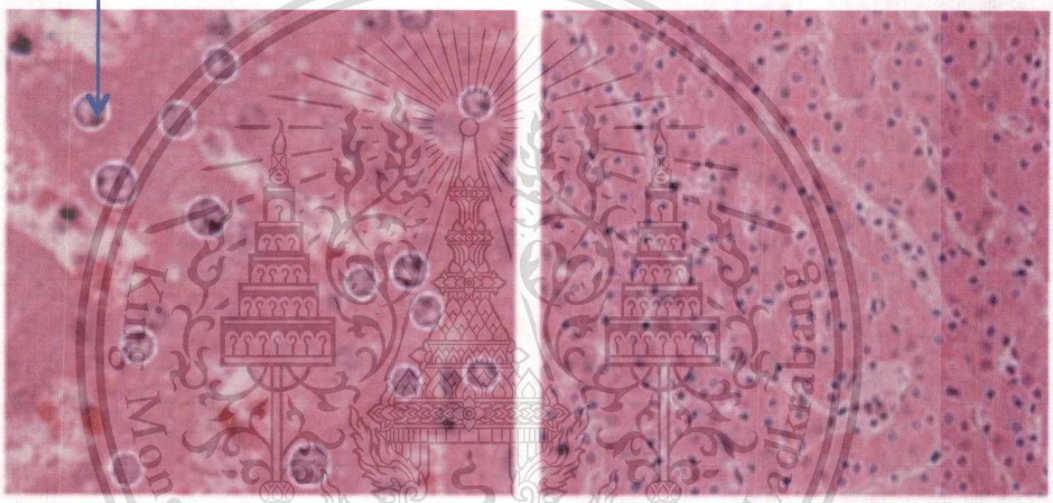


Figure 2-2: Nuclei/cells segmentation results: (a) The original size of 1000x1000 pixels and (b) 2800x2750 pixels

Figures 2-1 and 2-2 show two examples of segmentation results for biopsy images. As shown in Figure 2-2(a) and (b), Both images are shrunk to 256x256 pixels for convenience of displaying.

Although this research relates to ours in term of image segmentation, it focuses on nuclei/cell segmentation, which has a number of features that are different from fat e.g. color and size. They extract 14 features based on nuclei characteristics and use a technique called SVM decision graph for classification.

2.2.2 Bayesian segmentation of hepatic biopsy color images in the JPEG compressed domain

Another approach that related to segmentation in liver biopsy images is from Gordan et al. [4] who proposed a fast and accurate approach for identification of hepatic liver fibrosis in biopsy images that represent in JPEG compressed domain.

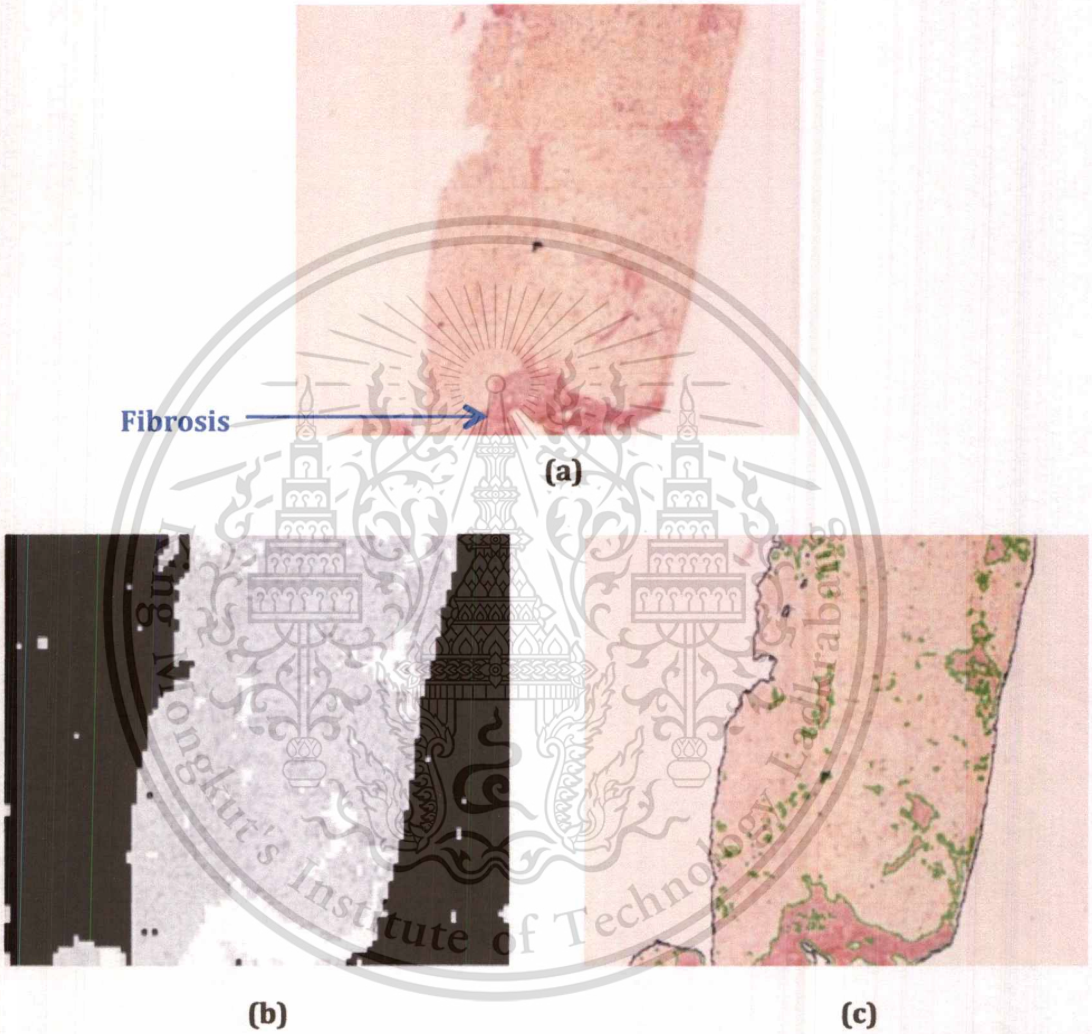
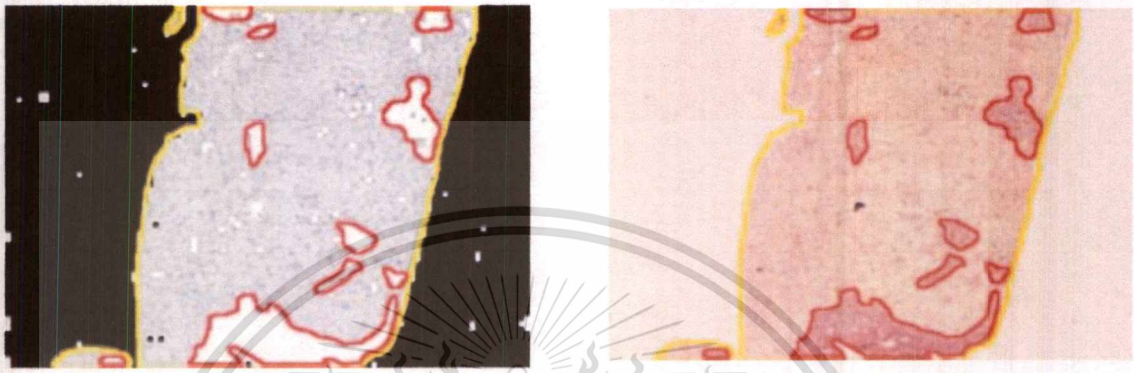


Figure 2-3: Hepatic biopsy image segmentation: (a) Original hepatic biopsy image, (b) Segmentation in JPEG compressed domain data, and (c) Segmentation at pixel level in the RGB color space

As example of segmentation of the fibrosis is shown in Figure 2-3. The features are based on the pixel color and texture. Bayesian classification algorithm on JPEG compressed domain data is used as shown in Figure 2-3(b). The white color and grey color represent fibrosis areas and tissue areas respectively.

In term of comparision, as shown in Figure 2-3(b), the same Baysean method is used on pixel level in the RGB color space.

However, as shown in Figure 2-4, the segmentation performance can be observed between the segmentation results (Figure 2-4(a)) from their method and ground truth (Figure 2-4(b)) which fibrosis areas are manually selected by human.



(a) (b)

Figure 2-4: Difference between: (a) Segmentation results on their method and (b) ground truth image

The method proves a good segmentation and quantification results, both in comparison to pixel level segmentation schemes and in reference to ground truth images. However, it focuses in liver fibrosis image segmentation which is different from liver fat.

Chapter 3

Background Knowledge

In order to achieve the project objectives outlined in Chapter 1, seven techniques are used. The theories behind those techniques are described as follows.

3.1 Grey-scale image

Grey-scale images [3] are 2-D arrays that assign one numerical value to each pixel that is representative of the intensity at this point. The pixel value range is bounded by the bit resolution of the image and such images are stored as N-bit integer images with a given format.

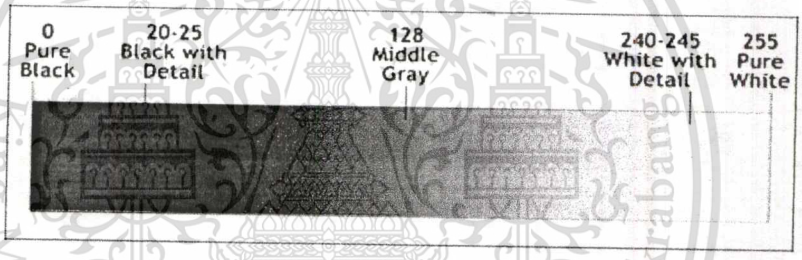


Figure 3-1: Grey-scale value

As shown in Figure 3-1, the scale used is 0-255 where white with detail is at about 240-245. Above that, white will start to lose any true detail as it approaches pure white of 255. On the shadow side, black with detail is at about 20-25. Below 20, black will start to lose the shadow detail. So, pure black is 0 and pure white is 255. Figure 3-2 shows an example of grey scale image conversion algorithm.

The procedure to convert RGB image to grey-scale image is described by a simple algorithm as shown in Figure 3-2. First is to get the red, green, and blue values of a pixel and turn those numbers into a single grey value, then replace the original red, green, and blue values with the new grey value.

```
For Each Pixel in Image {

Red = Pixel.Red
Green = Pixel.Green
Blue = Pixel.Blue

Grey = (Red + Green + Blue) / 3

Pixel.Red = Grey
Pixel.Green = Grey
Pixel.Blue = Grey

}
```

Figure 3-2: Simple grey-scale image conversion algorithm

However, due to the fact that human eyes are not uniform across colors. Humans perceive green more strongly than red, and red more strongly than blue. This may be due to the fact that much of the natural world appears in shades of green, so humans have evolved greater sensitivity to green light. Human eyes then do not perceive all colors equally. The simple algorithm of grey-scale conversion (Figure 3-2) is therefore imprecise. Instead of treating red, green, and blue light equally, a good grey-scale conversion algorithm will weigh each color based on how the human eyes perceive it. The algorithm that will be used in our project that gives result in a more vital grey-scale image is displayed in Eq. 3.1.

$$Grey = (0.299)Red + (0.587)Green + (0.114)Blue \quad (3.1)$$

3.2 Smooth or Low-pass filter

"Smooth or Low-pass filtering is a technique used to replace each pixel by the average value of the pixels around it. By doing this, the rapid intensity variations will be smoothed out and thus replaced by a more gradual transition" [6].

Smoothing filters are used for blurring an image and for reducing noises. Noise reduction can be achieved by using linear filter called moving-average filter [3]. "A filter is said to be linear by replacing a pixel by a weighted sum of neighboring pixels. This is the case of the box filter in which a pixel is replaced by the sum of all pixels in a rectangular neighborhood and divided by the size of this neighborhood to get the average value. The different weights of a filter can be represented using a matrix that shows the multiplying factors associated with each pixel position in the considered neighborhood. The central element of the matrix corresponds to the pixel on which the filter is currently applied" [6].

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

0	1/8	0
1/8	1/2	1/8
0	1/8	0

0	1/5	0
1/5	1/5	1/5
0	1/5	0

Figure 3-3: 3x3 Box filter

As shown in Figure 3-3, there are three different 3x3 masks with different choices of coefficients. The 3x3 group of pixels centered on the given pixel is considered the simplest case to replace the central pixel value by the unweight average of these nine pixels.

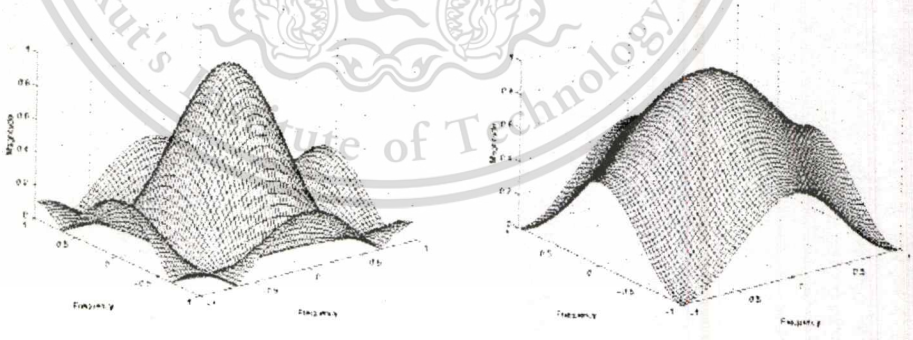


Figure 3-4: Magnitude

As shown in Figure 3-4, the left one has a smaller bandwidth, therefore better noise removal ability.

3.3 Binary image

"Binary images are produced from color images by segmentation. Segmentation is the process of assigning each pixel in the source image to two or more classes. If there are more than two classes then the usual result is several binary images. "The simplest form of segmentation is probably assigns pixels to foreground or background based on grey-scale intensity. Another method is edge detection that creates a binary image with some pixels assigned to edge pixels, and is also a first step in further segmentation" [9].

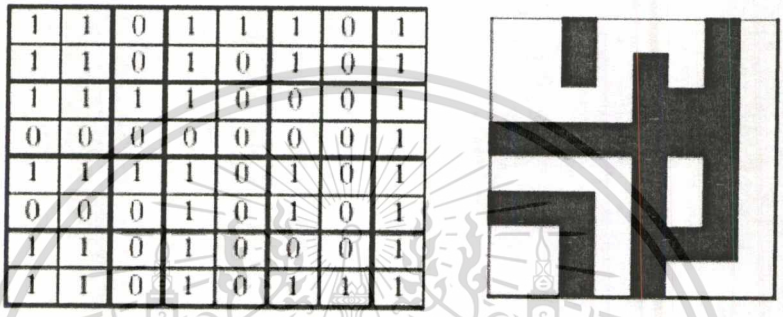


Figure 3-5: Example of binary image labeling

As shown in Figure 3-5, binary images are 2-D arrays that assign one numerical value from the set {0, 1} to each pixel in the image. They are normally displayed as black and white. Numerically, the two values are often 0 for black, and either 1 or 255 for white.

3.4 Morphological opening image

Opening is defined as the dilation and the erosion of an image. It can be used to remove the small blobs introduced by image noise.

The process of erosion starts by placing the center pixel of the structuring element on each foreground pixel. If any of the neighborhood pixels are background pixels, the foreground pixel is switched to background. On the other hand, the dilation process starts by placing the center pixel of the structuring element on each background pixel. If any of the neighborhood pixels are foreground pixels, the background pixel is switched to foreground. Please see in Figure 3-6 illustrates an example of erosion and dilation.

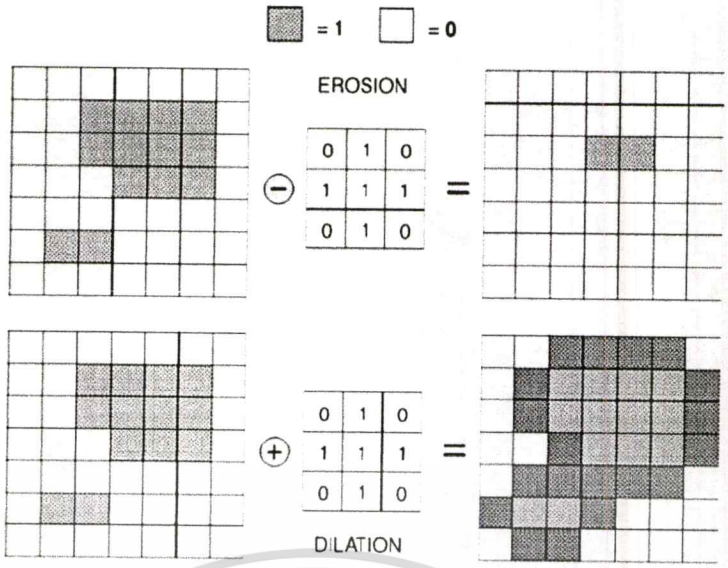
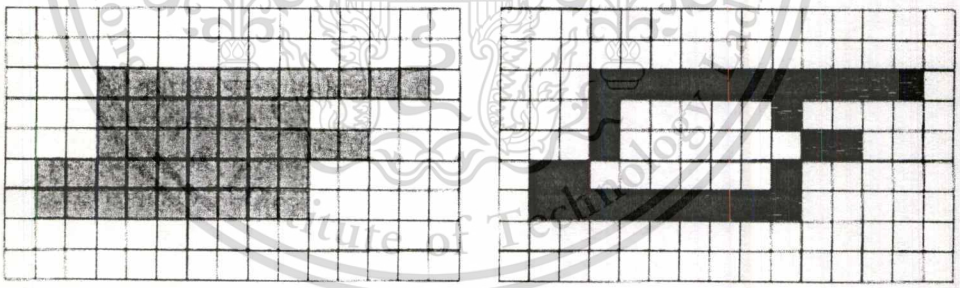


Figure 3-6: Example of performing erosion and dilation

3.5 Contour image

"The contours are extracted by programing that consists of systematically image scanning until a component is hit. From this starting point on the component, its contour is followed, marking the pixels on its border. When the contour is completed, the scanning resumes at the last position until a new component is found" [6].



Binary region *Boundary*

Figure 3-7: Example of extracting contour

Figure 3-7 is an example of how this technique is used to find the connected pixels of a region that are adjacent to the background by selecting a starting pixel and tracking the boundary until it comes back with the starting pixel.

3.6 Area, perimeter, and circularity

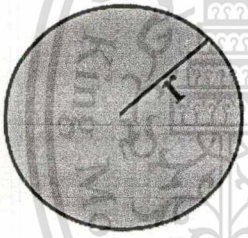
"Circularity is a measure of the roundness of a shape such as blobs. A circle is the most compact shape, so the more compact a shape is, the more closely it resembles a circle. Circularity is a ratio and therefore a dimensionless number" [8].

Circularity of blob formula:

$$C = \frac{4\pi A}{L^2} \tag{3.2}$$

where C describes the circularity of blob, A describes the area which can be simply calculated by counting the number of pixel within each blob, and L describes the perimeter of blob which can be approximated by counting number of pixels on its border.

A perfect circular shape C can be proven by,



A regular circle has area of $A = \pi r^2$

and perimeter of $L = 2\pi r$

From the circularity formula,

$$C = \frac{4\pi(\pi r^2)}{(2\pi r)^2} \tag{3.3}$$

So,

$$C = 1, \text{where } 0 < C \leq 1 \tag{3.4}$$

The circularity C will be greater than 0 and less than or equal to 1 (Eq. 3.4). A long, thin shape has a circularity that approaches 0 and a circle has a circularity of 1.

3.7 *k*-NN algorithm

k-nearest neighbor (*k*-NN) algorithm [2] is the simplest of all machine-learning algorithms: an object is classified by a majority vote of its neighbors, with the object being assigned to the class most common amongst its k nearest neighbors.

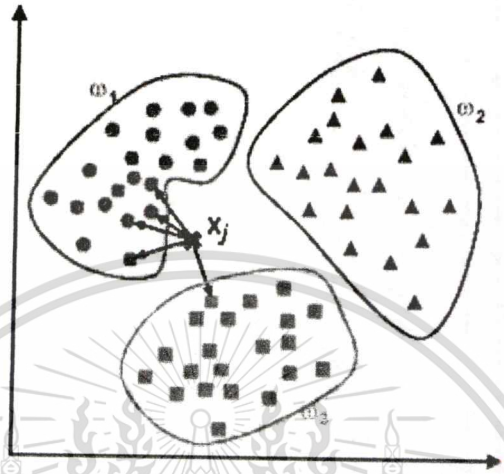


Figure 3-8: K-NN classification example

As shown in Figure 3-8, there are three classes (circle, rectangle and triangle class). The goal is to find a class label for the unknown example x_j. In this case we use a value of k=5 neighbors is used as an example. Of the five closest neighbors, four belong to w_1 and one belongs to w_3, so x_j is assigned to w_1, the triangle class.

"The advantages of K-NN algorithm are analytically tractable, simple implementation, use local information which can yield highly adaptive behavior, and easily to parallel implementations.

On the other hand, *k* -NN algorithm requires large storage of requirements, computationally intensive recall, and highly susceptible to the dimension" [7].

Chapter 4

System Overview

Before system requirements can be put together, we had several meetings with pathologist team from Rama Hospital to discuss about the project objectives and gather project requirements. For more details regarding what was discussed, please refer to Appendix A.

After we obtained all the information, we constructed the user and software requirements which can be further divided into functional and non-functional requirements. The requirements are then translated into diagrams such as use cases and activity diagrams. When we finished the requirement part, we can continue to do the analysis in order to design the software and find the appropriate techniques and tools using our background knowledge described in Chapter 3. The software designs are represented in term of diagrams such as sequence, package and class.

4.1 Overview of requirements

This section describes an overview of the project requirements i.e. what functions the software will be provided. The requirements consist of user requirements and software requirements and both can be divided into functional and non-functional requirements respectively.

4.1.1 User requirements

Functional requirements

R1 Upload original image, user can upload whether in single image or in multiple images in order to start diagnosing. (Digital images are scanned from the tissue slides of a biopsy)

R2 Diagnose from the image;

R2.1 User will get diagnosed result from the input image and the result will be shown in term of percentage of fat proportion.

R2.2 User can click un-highlight/highlight button to preview the undiagnosed image and diagnosed image respectively.

* User is usually a pathologist or a doctor.

R3 Edit image to get a more precise result;

R3.1 User can remove the wrong-highlighted fat areas.

R3.2 User can add unhighlighted fat areas

R3.3 User can undo the editing.

R4 Save image file including diagnosed image and result.

R5 Search history image by patient ID.

Non-functional requirements

R1 User Interface is provided for user to adjust images manually. Program is easy-to-use and has consistent and intuitive user interface.

R2 When uploading an image, it is easy for users to find image through the user's file directory.

R3 Easy to adjust and edit image.

R4 Calculate and show diagnosed result more precisely than the eye-ball technique.

4.1.2 System requirements

Functional requirements

R1 Enable search: display search results and lead user to diagnosis page.

R2 Image processing in segmentation:

R2.1 Segment fat areas, analyze, evaluate and show results in percentage by image processing procedure.

R2.2 Automatically highlights fat areas.

R2.3 Enable user to edit image manually.

Non-Functional requirements

R1 Digital images in.TIFF format type with a size of less than 4MB.

R2 Response time: one upload should be displayed to the user within 5 seconds after issuing the query.

4.2 Software requirements specification

Software requirements specification (SRS) describes behaviors of the software system to be developed. SRS includes use case diagram which describes interaction between user and the software. It also includes activity diagram which is the design of software user interface.

4.2.1 Use case diagram: Liver Biopsy Image Processing <<Application>>

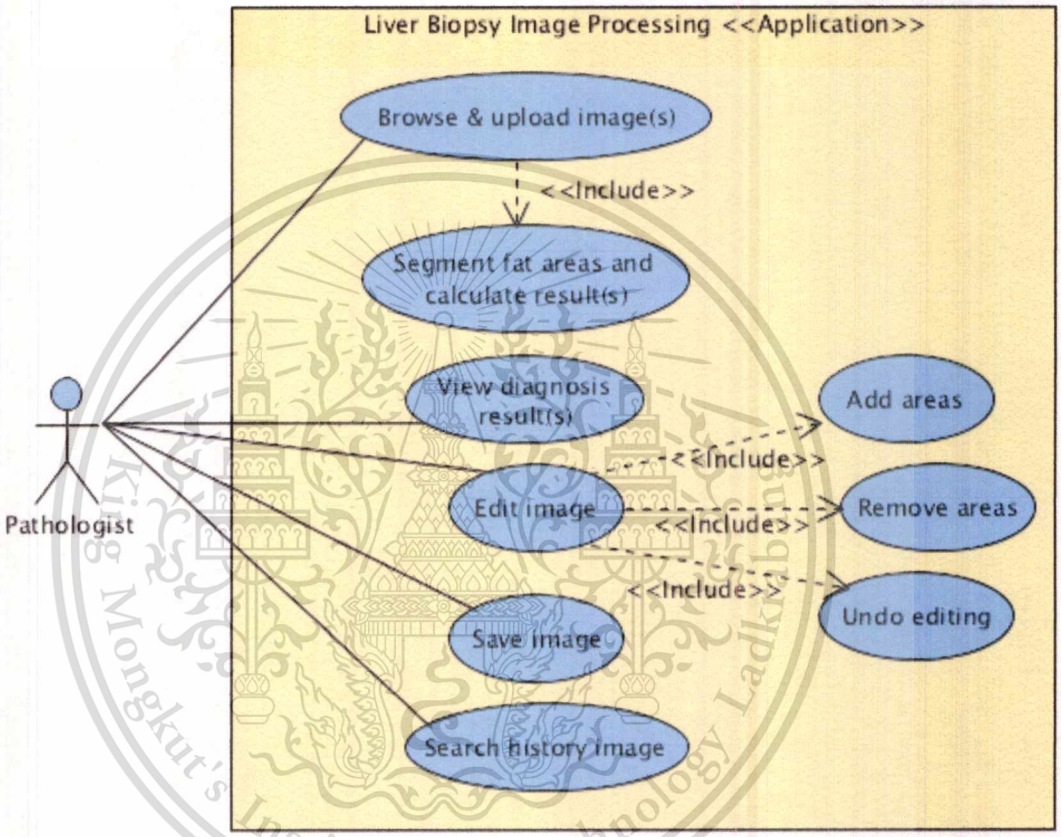


Figure 4-1: Use case diagram

As shown in Figure 4-1, the use case diagram depicts the interaction between system and user that is a pathologist within the system called *Liver Biopsy Image Processing <<Application>>*. This diagram has 8 use cases. Each can be described as requirements with relations between them. The detail for each use case is described in Table 4-1 to Table 4-8.

Table 4-1: Use case of browse & upload image(s)

Use case name	Browse & upload image(s)
Actor	Pathologist
Purpose	For user to browse and upload liver biopsy image(s) and start the diagnosis.
Precondition	User chose "upload single image" or "upload multiple images" from the main menu.
Post condition	The image(s) has been uploaded and diagnosed.
Process	a. User chooses "upload single image" from the main menu. a.1 User browses an image through user's directory a.2 The image will be shown in the diagnosis page. b. User chooses "upload multiple images" from the main menu. b.1 User browses a set of images through user's directory. b.2 All diagnosed images will be shown in a table of original image, diagnosed image, and result respectively. b.3 User clicks one of the images, then the program will lead to the diagnosis page for user to do further edit the individual image to re-calculate the diagnosed result.
Exception	User cancels the image(s) that user has been uploaded.

Table 4-2: Use case of segment fat areas and calculate result(s)

Use case name	Segment fat areas and calculate result(s)
Actor	-
Purpose	1. Use case "Browse & upload image(s)" includes "segment fat areas and calculate result(s)". 2. Image segmentation is a process that the program performs when user wants to diagnose a liver biopsy image. The program will segment fat areas in order to calculate the result (percentage of fat).

Table 4-3: Use case of view diagnose result

Use case name	View diagnosis result
Actor	Pathologist
Purpose	For user to view diagnosed result of fat proportion of the uploaded image.
Precondition	After user has uploaded single image in diagnosis page or has chosen one of the images from multiple uploads.
Post condition	The program has calculated the result.

Table 4-4: Use case of edit image

Use case name	Edit image
Actor	Pathologist
Purpose	For user to edit the processed image by manually adding or removing fat areas in order to re-calculate the result.
Precondition	User has already got the diagnosed image.
Post condition	The result has been re-calculated.
Process	1. User clicks the area to be added or removed. 2. User clicks "re-diagnose" button. 3. The program calculates new result.
Exception	User clears the editing by clicking "undo" button.

Table 4-5: Use case of add areas

Use case name	Add areas
Actor	-
Purpose	1. Use case "edit image" includes "add areas". 2. User could be able to add more fat areas.
Exception	User clears the editing by clicking "undo" button.

Table 4-6: Use case of remove areas

Use case name	Remove areas
Actor	-
Purpose	1. Use case "edit image" includes "remove areas". 2. User could be able to remove fat areas.
Exception	User clears the editing by clicking "undo" button.

Table 4-7: Use case of undo editing

Use case name	Undo editing
Actor	Pathologist
Purpose	1. Use case "edit image" includes "undo editing". 2. After user edited areas, user can also undo the activities.

Table 4-8: Use case of save image

Use case name	Save image
Actor	Pathologist
Purpose	For user to save file after edited and diagnosed the image.
Precondition	User finished the editing and diagnosing the image.
Post condition	The diagnosed image and result are saved to user's directory.
Process	1. User clicks "save" button. 2. The diagnosed image has been saved including diagnosis result to user's directory.
Exception	User can choose not to save image.

Table 4-9: Use case of search history image

Use case name	Search history image
Actor	Pathologist
Purpose	For user to search for a history image (diagnosed image) for perform further editing and re-calculate the result.
Precondition	User chose "search history image" from the main menu.
Post condition	The program has led to the diagnosis page.
Process	1. User types file name into the search bar in the main menu. 2. The program will lead to the diagnosis page with diagnosed image and result. User can then perform further editing and result re-calculation.

4.2.2 Activity diagram

The activity diagram depicts the workflows of stepwise activities and actions with support for choice and iteration of components in the systems. The diagram can be divided into two main parts namely user and liver biopsy image processing (LBIP) system. Please see in Figure 4-2.

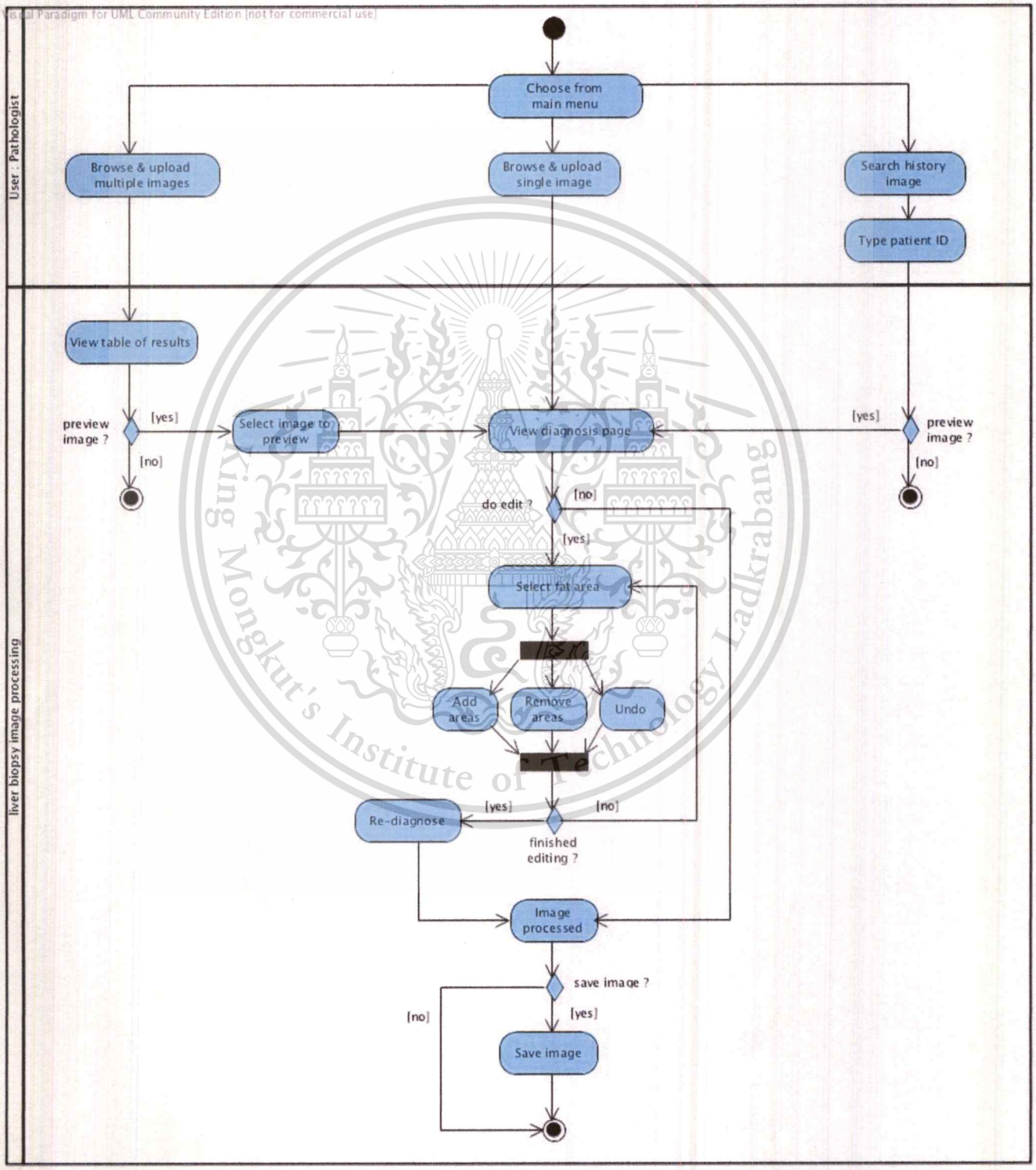


Figure 4-2: Activity diagram

Chapter 5

Software Design

After we have gathered and analyzed requirements, the next step is to design the software by applying the analyzed requirements to construct class diagram and package diagram, which will be used in the development and implementation phase.

5.1 Analysis class diagram

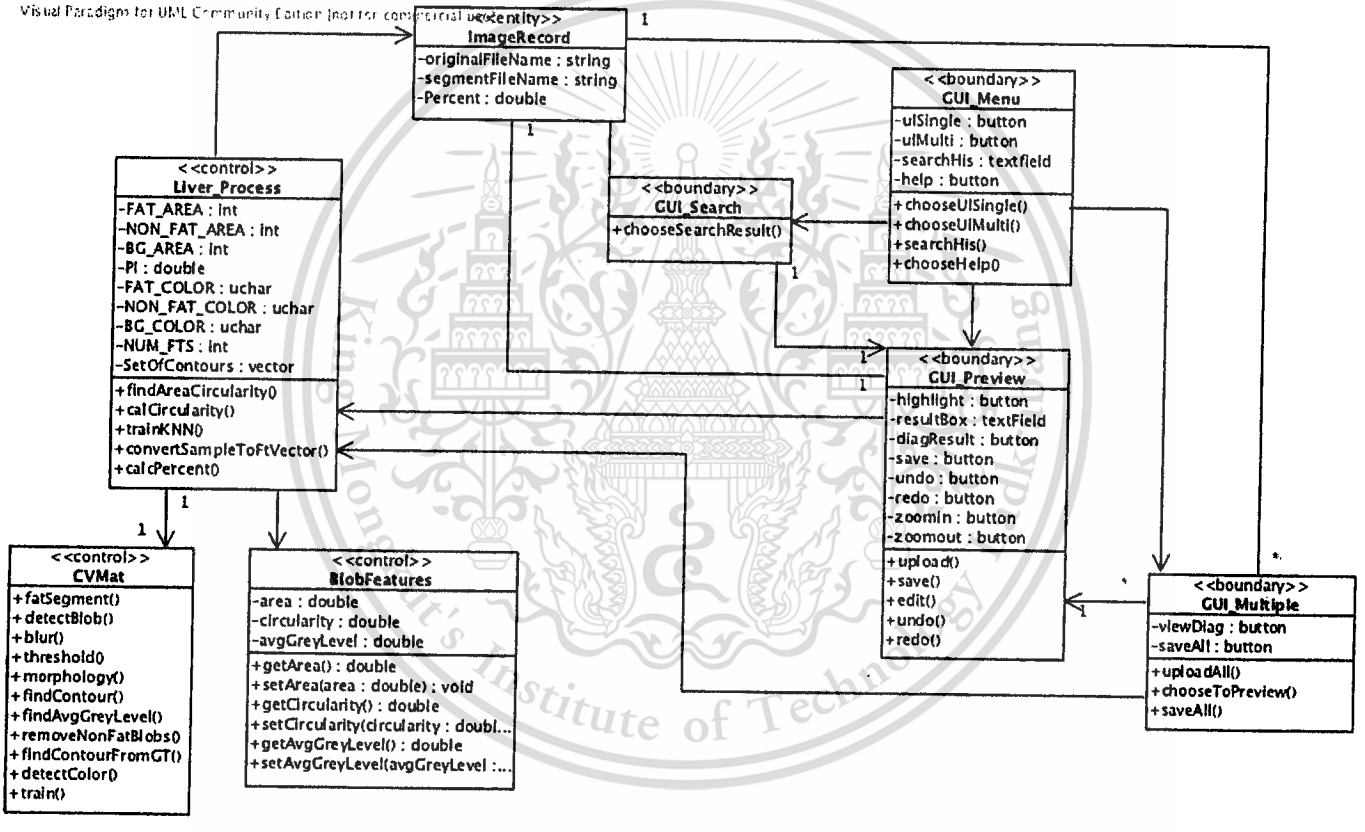


Figure 5-1: Class diagram

The class diagram in Figure 5-1 illustrates an overview of the LBIP system that presented by system classes, attributes and methods in each class. The detail for each use case is described in Table 5-1 to Table 5-8.

Table 5-1: Class for main menu

Class name	GUI_Menu
Attributes	-ulSingle : button -ulMulti : button -searchHis : textfield -help : button
Operations	+chooseUlSingle +chooseUlMulti() +searchHis() +chooseHelp()
Description	This class provides initial interface for user to choose an activity from main menu i.e. upload single image, upload multiple image or search history image.

Table 5-2: Class for single image preview

Class name	GUI_Preview
Attributes	-highlight : button -resultBox : textField -save : button -undo : button
Operations	+upload() +save() +edit() +undo()
Description	This class provides diagnosis page for user to preview the uploaded image with diagnosed image and result. It also allows user to edit the processed image.

Table 5-3: Class for multiple image previews

Class name	GUI_Multiple
Attributes	-viewDiag : button -saveAll : button
Operations	+uploadAll() +chooseToPreview() +saveAll()
Description	This class provides multiple images preview page, which enables user to choose one of the uploaded images to preview. Also, user can save all diagnosed image.

Table 5-4: Class for image search

Class name	GUI_Search
Attributes	-
Operations	+chooseSearchResult()
Description	This class provides search function for user to search for a previously diagnosed image.

Table 5-5: Class for record image

Class name	ImageRecord
Attributes	-originalFileName : string -segmentFileName : string -Percent : double
Operations	-
Description	This class provides a function for user to record data. It will be called after user diagnosed or saved image(s). Also, the class will be called when user search for a previously diagnosed image.

Table 5-6: Class for liver image processing

Class name	Liver_Process
Attributes	-FAT_AREA : int -NON_FAT_AREA : int -BG_AREA : int -PI : double -FAT_COLOR : uchar -NON_FAT_COLOR : uchar -BG_COLOR : uchar -NUM_FTS : int -SetOfContours : vector
Operations	+findAreaCircularity() +calCircularity() +trainKNN() +convertSampleToFtVector() +calcPercent()
Description	This class provides a function to process image from input image, calculate the fat percentage result, and send back to user. Also, it is the class that is used for call function of training image by *k*-NN algorithm. (Described in Chapter 6)

Table 5-7: Class for CV Mat

Class name	CVMat
Attributes	+fatSegment() +detectBlob() +blur() +threshold() +morphology() +findContour() +findAvgGreyLevel() +removeNonFatBlobs() +findContourFromGT() +detectColor() +train()
Operations	-
Description	This class provides image processing function and image training techniques to be called by Liver_Process class. (Described in Chapter 6)

Table 5-8: Class of features training

Class name	BlobFeatures
Attributes	-area : double -circularity : double -avgGreyLevel : double
Operations	+getArea() : double +setArea(area : double) : void +getCircularity() : double +setCircularity(circularity : double) : void +getAvgGreyLevel() : double +setAvgGreyLevel(avgGreyLevel : double) : void
Description	This class provides image training features to be called by Liver_Process class during an image training step. (Described in Chapter 6)

5.2 Package diagram

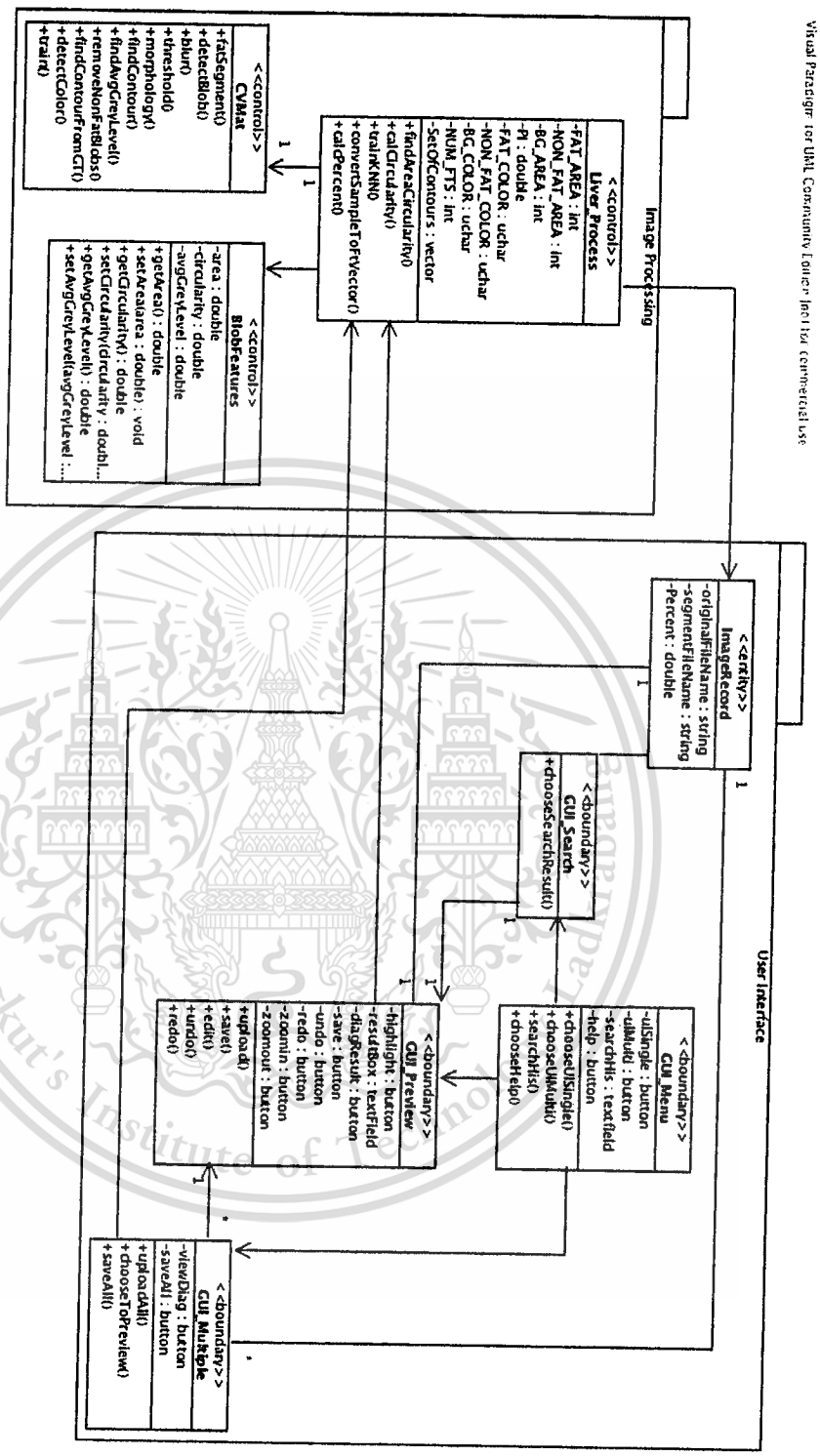


Figure 5-2: Package Diagram

The package diagram in Figure 5-2 illustrates the group of classes that can be divided into two main packages namely user interface package and image processing package.

Chapter 6

Development

After we have gathered requirements and designed the software, the next step is to implement the system by applying the background knowledge or theories of image processing and image training described earlier in Chapter 3 to fulfill the requirements described in Chapter 4.

6.1 Liver fat image processing with related image techniques

This project uses a number image processing techniques to perform liver fat segmentation. In this section, the techniques used will be described in the manner that is specific to our implementation. However, please refer to Chapter 7 for more details regarding the implementation results.

In the following sections, we adopted several image processing techniques to extract the area of candidate fat blobs from an original input image of liver biopsy. Next, background, which is the area of empty slide, will be segmented. Then three blob features, i.e., area, circularity, and average grey-scale value, are calculated for each blob. Finally, a classification technique called *k*-nearest neighbors (*k*-NN) [2] is used to remove blobs that are not the real fat from the list of candidate.

6.1.1 Grey-scale image conversion

In order to extract the candidate fat blobs in an original input image, grey-scale image conversion is required at the very first step of the segmentation process. By considering the attributes of liver fat (described in 7.1.1), we converted an original image of liver biopsy to grey-scale image in order to see the components within the image more clearly. Please see result in Figure 7-3.

6.1.2 Smooth or Low-pass filtering

The simplest low-pass filter is to be used to calculate the average of a pixel and all of its neighbors. The result replaces the original value of the pixel. The process is repeated for every pixel in the image.

1/49	1/49	1/49	1/49	1/49	1/49	1/49
1/49	1/49	1/49	1/49	1/49	1/49	1/49
1/49	1/49	1/49	1/49	1/49	1/49	1/49
1/49	1/49	1/49	1/49	1/49	1/49	1/49
1/49	1/49	1/49	1/49	1/49	1/49	1/49
1/49	1/49	1/49	1/49	1/49	1/49	1/49
1/49	1/49	1/49	1/49	1/49	1/49	1/49

Figure 6-1: 7x7 box filter

From figure 6-1, we set a moving-average filter value to the size of 7x7 [3] for our implementation. Please note that after we experimented with various filter values, the size of 7x7 is considered as optimal to properly remove noises or small details. Please refer to Appendix C for more information on the experiments.

From the 7x7 box, each pixel and its 48 neighbors are multiplied by 1/49 and added together. The pixel in the middle is replaced by the sum. Please see the implementation of smooth filtering in open CV below. Please also see result in Figure 7-3.

```
cv::blur(inputImg, outputImg, cv::Size(7, 7));
```

(Please see full source code in Appendix D)

6.1.3 Binarization (Thresholding)

Binarization is used to change color image into black and white. The color used for objects or blobs in the image is the foreground color while the rest of the image is the background color. Generally the technique separates blobs from the background by selecting threshold value. In our implementation, the threshold value of 220 is considered to be optimal. Please see the implementation of binarization in open CV below. Please also see result in Figure 7-4.

```
cv::threshold(inputImg, outputImg, 220,
                                255,cv::THRESH_BINARY);
```

(Please see full source code in Appendix D)

6.1.4 Morphological opening

Opening which is defined as the erosion and the dilation of an image is used to remove the small blobs introduced by image noises. In our implementation, we chose a 7x7 elliptic structuring element [3] in order to get more apparent filter affect. Please see the implementation of opening in open CV below. Please also see result in Figure 7-5

```
cv::Mat element = getStructuringElement
                    (cv::MORPH_ELLIPSE,cv::Size(7, 7));

cv::morphologyEx(inputImg,outputImg,cv::MORPH_OPEN,
                                                element);
```

(Please see full source code in Appendix D)

6.1.5 Contouring

Liver images generally contain representation of objects, which are blobs. Contouring technique is used to identify or assign label to each blob and extract those blobs from the image background. In our implementation, we chose CV_8U because it is unsigned 8bit per pixel that can have values of 0-255 and it is the normal range for most images. Then we set scalar value to 255 (pure white) in this step of contouring. Please see the implementation of contouring in open CV below. Please also see result in Figure 7-6.

```
cv::findContours(inputImg, contours,CV_RETR_EXTERNAL,
                                    CV_CHAIN_APPROX_NONE);
cv::Mat contourImg (open.size(),CV_8U, cv::Scalar(255));
cv::drawContours(inputImg,outputImg,1,cv::Scalar(0),1);
```

(Please see full source code in Appendix D)

6.1.6 Background empty slide exclusion

The background, or empty slide area, are often included in the list of candidate blobs. Therefore, we need to exclude them by using the opening image and the contour image to check if a blob has the following conditions:

- An area of larger than 3000 pixels
- A length of contour more than 500 pixels
- A region joining to the edge of the image for more than 100 pixels.

Please see the implementation of background empty slide exclusion below. Please also see result in Figure 7-7.

```
const int bgArea = 3000;
const int contourLength = 500;
const int edgePixel = 100;


cv::Mat& input = cv::imread("open.bmp",0);

const int maxX = input.cols-2;
const int maxY = input.rows-2;

for(int i =0; i<contours.size();i++){
     int area = blobFts[i].getArea();
     int length = contours[i].size();
     if(area>bgArea || length>contourLength){
          int countEdgePixel = 0;
          for( int n = 0; n < contours[i].size(); ++n ) {
               int x = contours[i][n].x;
               int y = contours[i][n].y;
               if((x==1)||(x==maxX)||(y==1)||(y==maxY))
                    countEdgePixel++;
          }
          if( countEdgePixel > edgePixel ) {
           cv::floodFill(input, contours[i][len/2], 128 );
               }
```

(Please see full source code in Appendix D)

6.1.7 Calculation of area, perimeter and circularity

After the blobs are contoured and assigned with labels, area and perimeter of each blob are calculated. Area and perimeter are in turn used to calculate the blob's circularity according to Eq. 3.2. Please see the implementation of blob circularity calculation below. Please also see result in Table 7-1.

```
const double PI=3.141592;
double calCircularity(double area,double length){
return 4*PI*area/(length*length);
}
```

(Please see full source code in Appendix D)

6.1.8 Image training and blob classification by *k*-NN

k-nearest neighbors (*k*-NN) [2] is used to classify the candidate fat blobs in an original input image if it is fat or not based on the closest training examples in the feature space.

Before evaluating the *k*-NN performance, ground truth images are needed. We need to construct *a training set* or a set of sample images before sending them to the image training stage (*k*-NN). In constructing ground truth, we manually marked all blobs in two different colors that indicate fat (1) and non-fat (0). They were then converted to *train class (1, 0)*. These markings (ground truth) were sent to the pathologists team at Rama Hospital to verify. Please see the example of training image in Figure 6-2.

Figure 6-2: Training Image Example

Given these ground truth images, training and evaluation of the *k*-NN classifier can be then carried out. The image training stage (*k*-NN) requires ground truth images to know which classification each blob belongs i.e. fat (1) or non-fat (0).

To perform the blob classification, **three training features** are obtained:

1. Area

2. Circularity

3. Average grey-scale

As shown in Figure 6-4, three feature values: *area, circularity,* and *average grey-scale* are calculated for all blobs in each sample image. The three values are then used as the *training data*. The calculation for each feature is explained in Section 7.3. These data are then put into a *long array vector*. An example vector of train data for the image training stage (*k*-NN) would have values such as:

[63.5, 0.8, 248.95, 102.5, 0.87, 247.65,.....,1166.5, 0.86, 222.84]

with each vector element representing a feature value of blobs. The label is *1 or 0,* depending on whether that combination of feature values corresponds to fat or non-fat region respectively according to the ground truth images. Therefore, the vector data which to be sent to the *image training stage (k-NN)* should look like this:

Figure 6-3: Example of vector of train data

Finally, blob classification process is carried out by training the classifier using an original input image that came with data (feature values) and testing it on the *training set* that is unknown to it. All blobs will be classified in a 3-dimentional feature space with features of *area, circularity,* and *average grey-scale* to determine usable fat blobs that are fat areas with *k* value by *k*-NN algorithm. Please see result in Section 7.4.

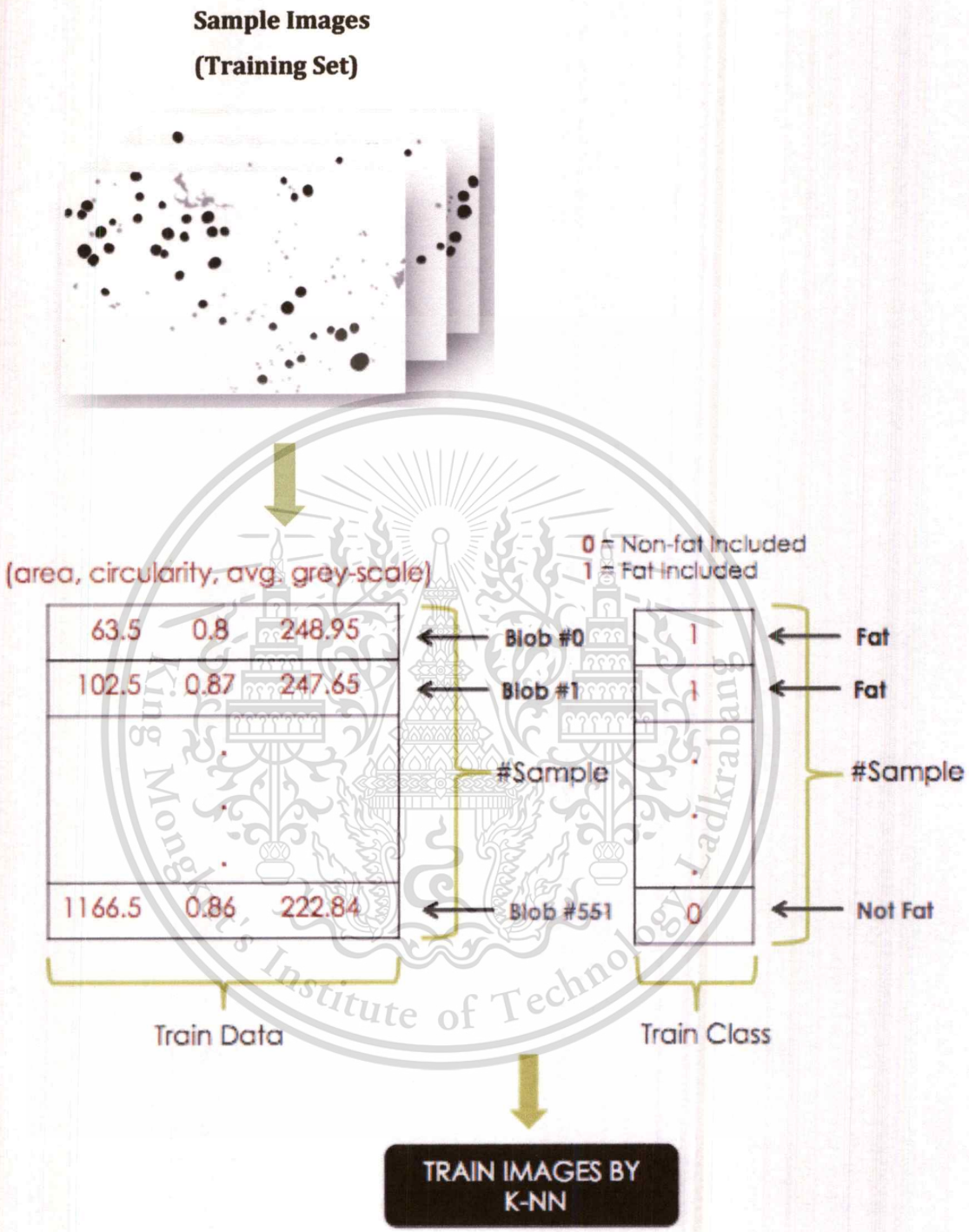


Figure 6-4: Example of image training

6.2 Tools

Software tools that are used in the project can be divided into software development tools consisting of programing tools and design tools, and project management tools which are platforms for planning and sharing work.

6.2.1 Software development tools

6.2.1.1 OpenCV (v. 2.4.2)

OpenCV is an open source library, which is the cross-platform for developing computer vision applications. OpenCV application provides function such as image segmentation and pattern recognition. It also includes a statistical machine-learning library that contains k-nearest neighbor algorithm which is used in this project. All of the libraries, developments and algorithms in OpenCV are developed in the C++ interface and they are manually connected to the code editor, Microsoft Visual Studio. Therefore, C++ language is used to develop our software.

6.2.1.2 Microsoft Visual Studio (2010)

Microsoft Visual Studio is an integrated development environment or IDE which is from Microsoft. It supports different programming languages and allows the code editor and debugger to support nearly any programming language. To be exact, Microsoft Visual Studio is used to implement, integrate with our user interface, and compile our openCV algorithms in C++ language.

6.2.1.3 QtDesigner

QtDesigner is also a cross-platform IDE and application framework that used for developing graphical user interface or GUI from Qt components. We customized our own user interface using widgets and dialogs in a what-you-see-is-what-you-get (WYSIWYG) manner, and tested them using different styles and resolutions. They are then integrated with programmed C++ code that connected to Microsoft Visual Studio afterwards.

6.2.1.4 GIMP (v. 2.6.0)

GIMP has tools for image retouching and editing, free-form drawing, resizing, cropping, and more specialized tasks. GIMP is then used to create the ground truth images which are the data set for image training stage, and also used for experimenting in the implementation part.

6.2.1.4 Matlab

Matlab is a high-level language and interactive environment for numerical computation, visualization, and programming. Therefore, Matlab tool is used with a built-in math functions to analyze data and create models by plotting data which are the feature values to create a 3-D graph of feature space.

6.2.2 Software project management tools

6.2.2.1 Microsoft office project (2007)

Microsoft Office Project is a tool used for our project management which creating a gantt chart or project plan with future works. Our project plan included a list of tasks carried out in our project, estimated assignment work and assigned resource for each task. Each resource can also be assigned to multiple tasks in multiple plans and each task can be assigned multiple resources, which based on resource. The program then calculated overall tasks period from the task level and then summarized to project level.

6.2.2.2 Dropbox

Dropbox is a free service that offers cloud storage. We used Dropbox to share every work files during our project within the group and also with our project advisors. It is real-time and easy to upload.

Chapter 7

Results

In this section, system implementation and demonstration are described in term of flowchart. The results of our software implementation are also shown in this chapter.

The implementation flow chart (Figure 7-1) depicts the workflows of an image that processed from the original input image to the output result. The detail for each process is described in the following sections.

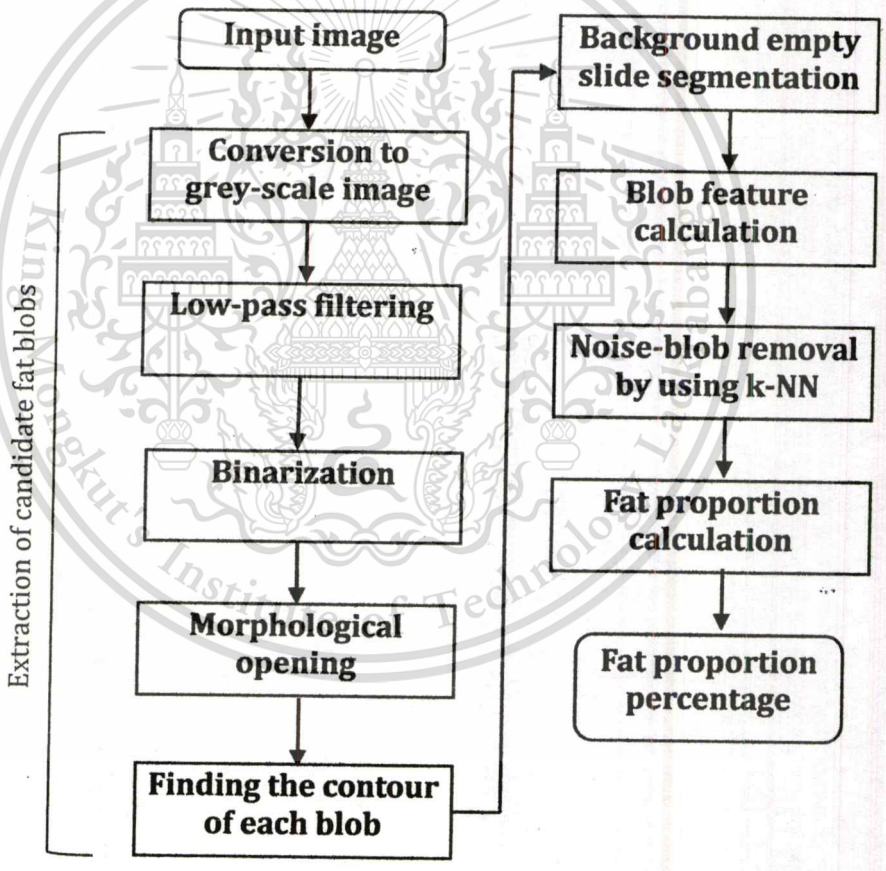


Figure 7-1: Flowchart of implementation

From the workflow in Figure 7-1, the software starting with original input images, generally from user interface. Several pre-processing steps of image processing techniques are required before classification of liver fat can be carried out. First, image segmentation is required to extract all the blobs out from the background of the image. Noises and other components in the image shall be gone and only blobs remain. Then all the blobs will be classified. Finally, the program can calculate the amount of liver fat in the image.

7.1 Extraction of candidate fat blobs results

As shown in Fig. 1, liver fat typically has white color and round shape while the color of liver tissue area is magenta. However, a liver image usually contains other objects such as vessel in liver or background area, which have the same color as fat. Moreover, small white blobs distributed over the entire tissue would not be counted as the liver fat as well because they are too small to consider as a fat cell and not somewhat affect to a fatty liver (said by a pathologist). Therefore, a fat detection algorithm for liver biopsy images must be able to distinguish between fat and other white objects.

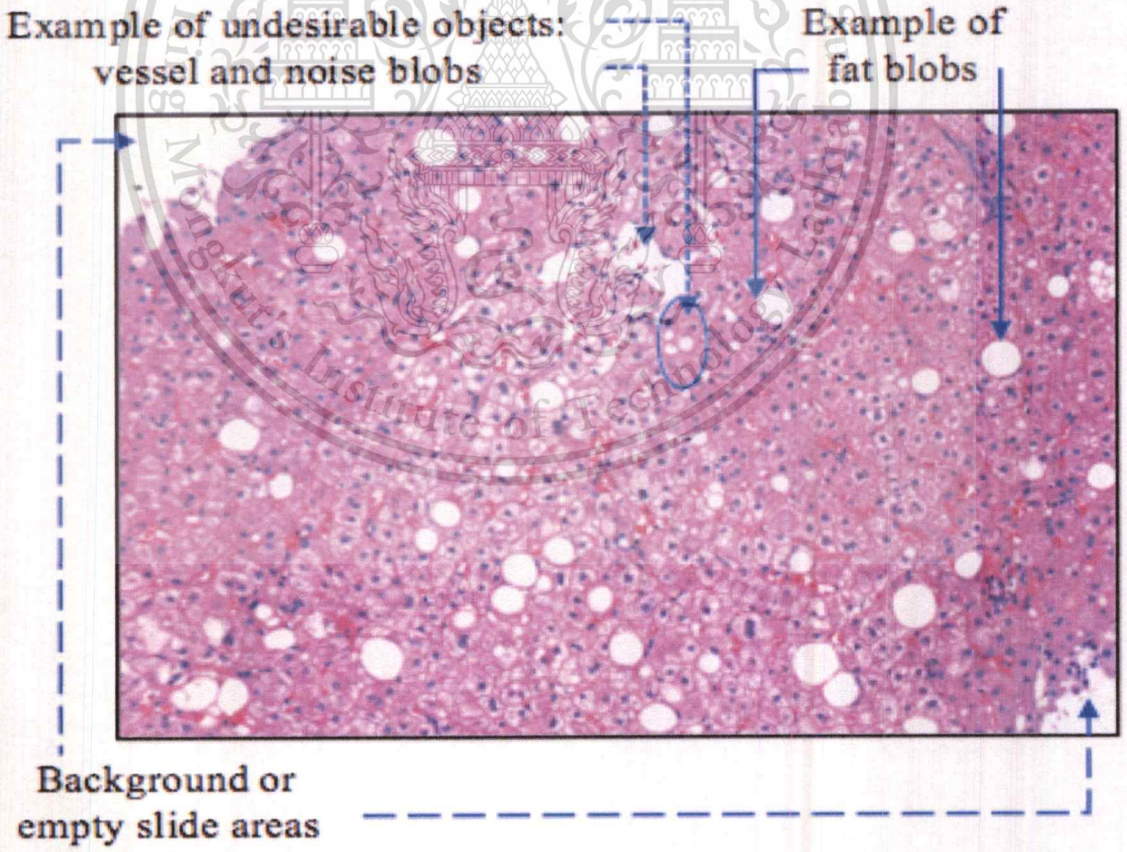


Figure 7-2: Original image of liver biopsy
(file name - S12-15745 A HE-5_p0.tif)

7.1.1 Grey-scale image conversion + smooth filtering result

One can easily see that a liver fat blob in an original image as shown in Figure 7-2 is usually represented as a white big circle, while the liver tissue is represented in magenta color. Our program then starts with converting the *original image to grey-scale image* in order to see all components within the tissue more clearly. Generally, this is to simplify edge, regions or blobs. Then, the program will perform a technique called *smooth filtering* to remove noises or small details. Please see result in Figure 7-3.

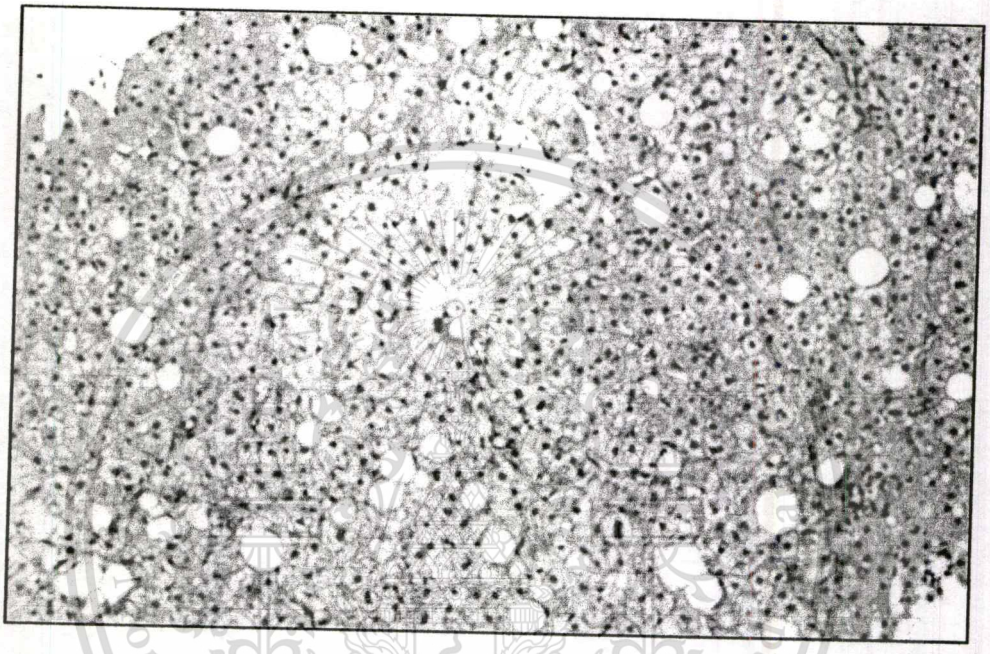


Figure 7-3: Grey-scale conversion + smooth filtering result

7.1.2 Binarization (Thresholding) result

In this step, the program will perform binarization to change the color image into black and white in order to separate objects, which are blobs, out from the background. Please see result in Figure 7-4.

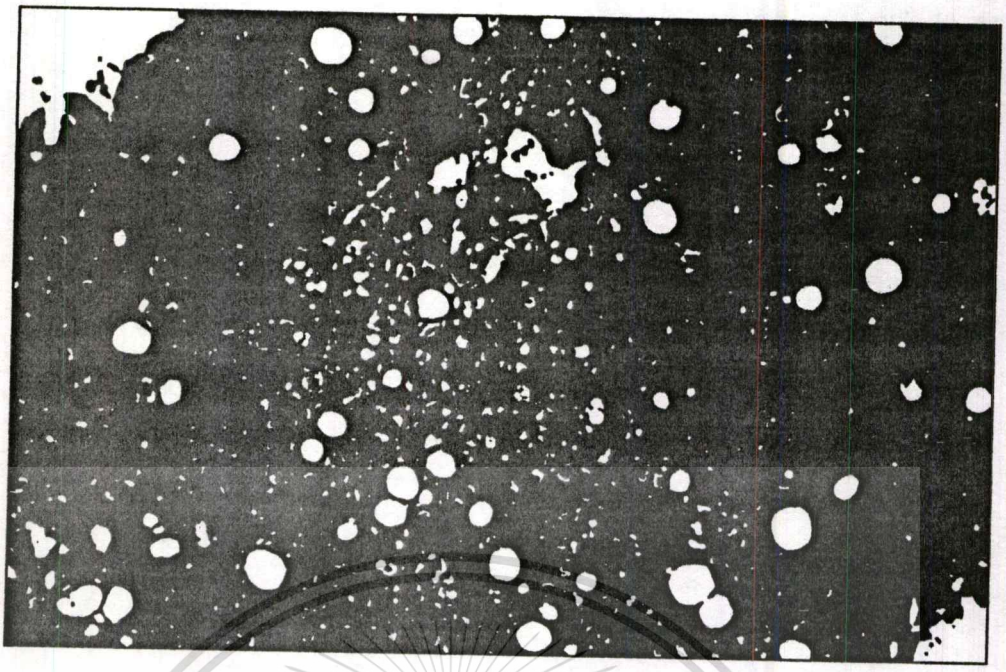


Figure 7-4: Binarization or thresholding result

7.1.3 Morphological opening result

In this step, the program will perform morphological opening to break apart the joining fat regions. The process is done by erosion followed by dilation. Please see result in Figure 7-5.

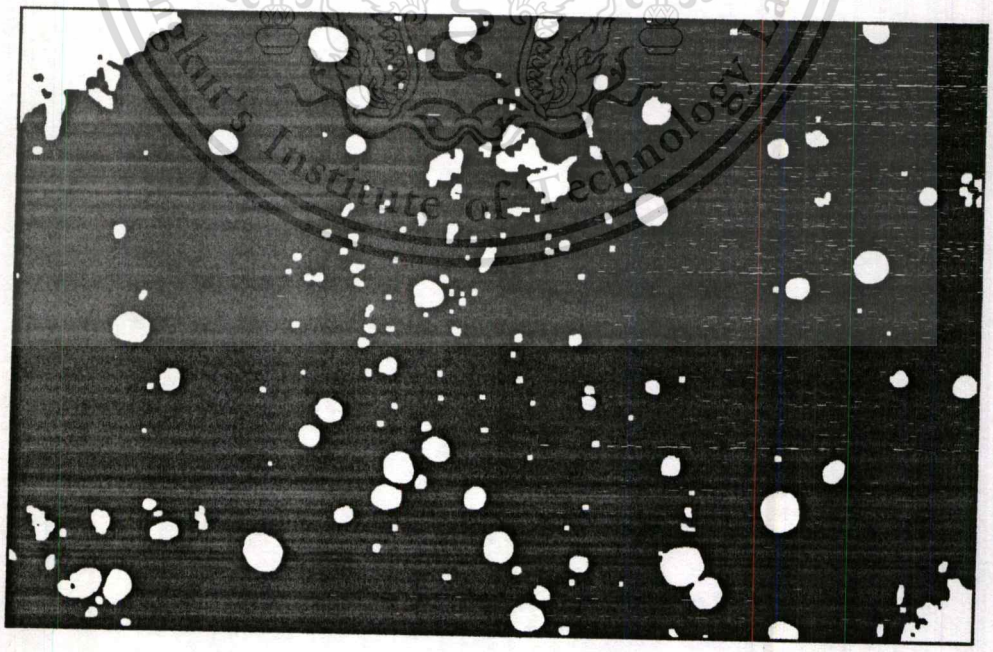


Figure 7-5: Opening result

7.1.4 Contouring and assignment of label to each blob result

In this step, the program will perform contouring to segment the blob regions even more, assign label to each object (fat), and extract those objects to be used in calculation of circularity. Please see result in Figure 7-6.

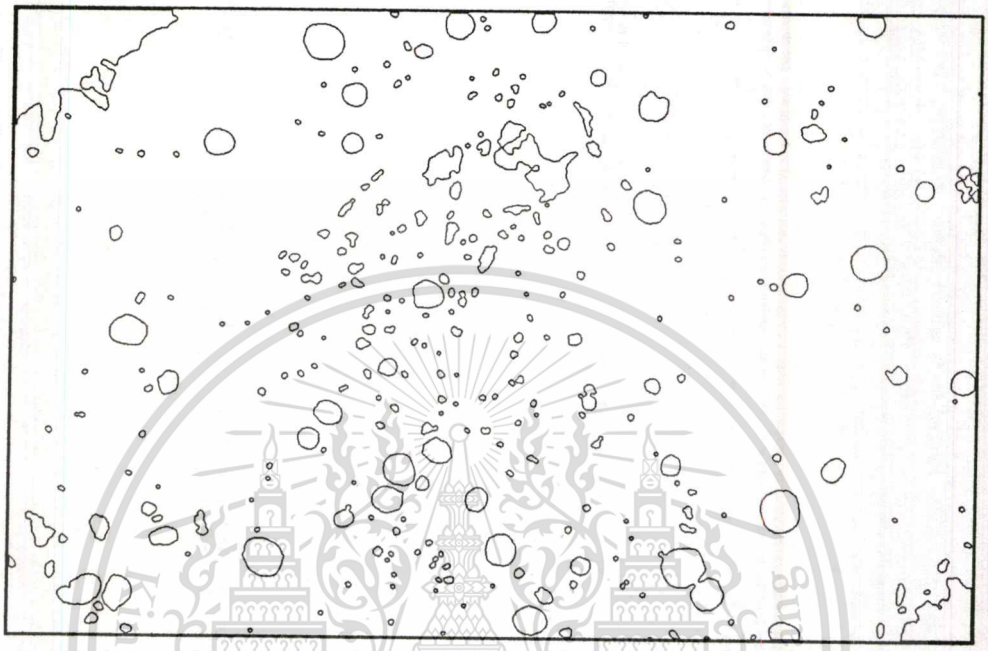


Figure 7-6: Contouring result

7.2 Background empty slide exclusion result

Program will use the opening image and contour image to exclude the background empty slide areas as explained in Section 6.1.6. Please see result in Figure 7-7. Note that the areas of background empty slide that are segmented by the program are represented in grey color.

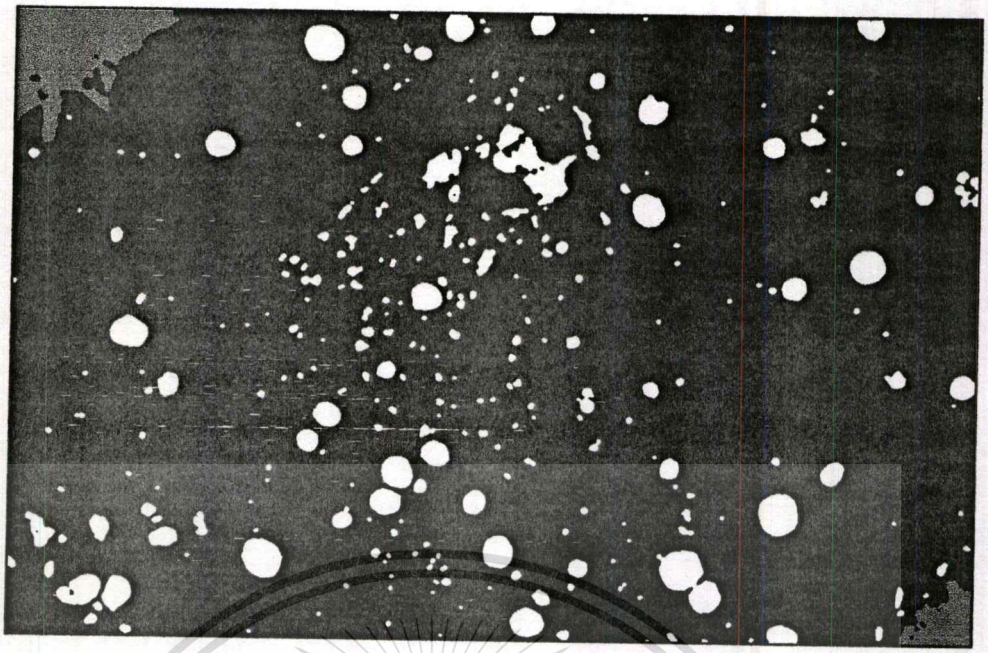


Figure 7-7: Background empty slide exclusion result

7.3 Calculation of blob features result

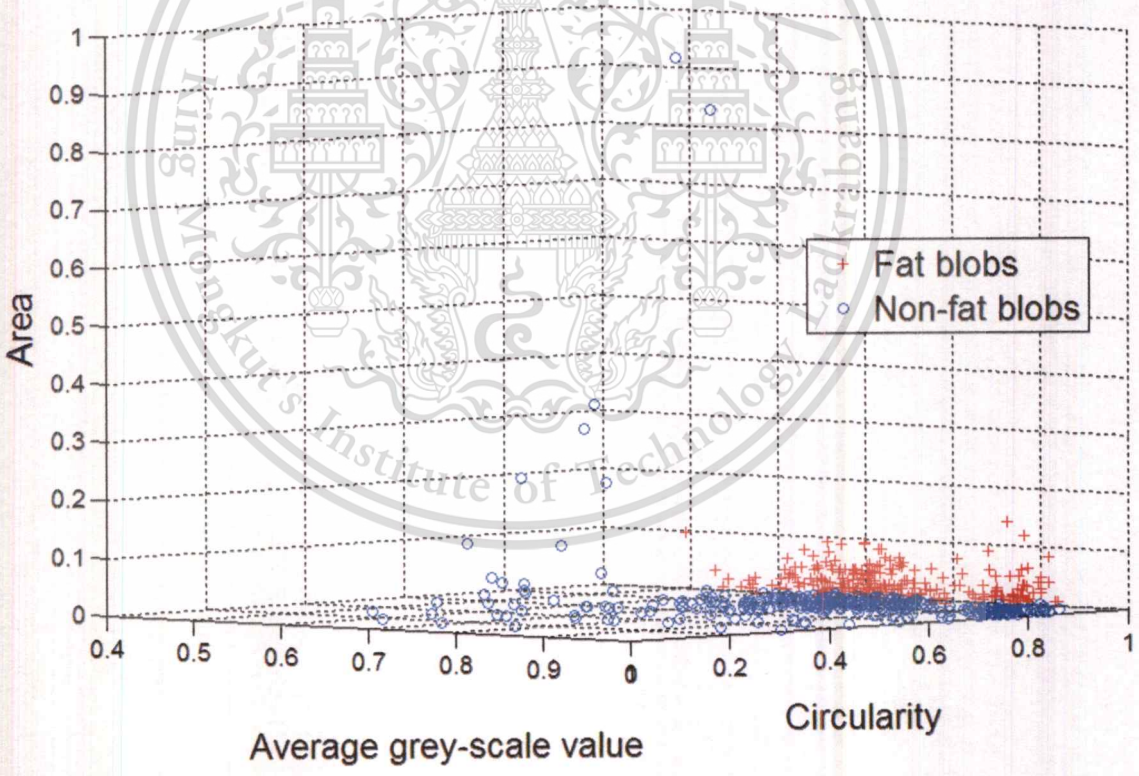
Since the classification of blobs requires a *set of features* to be calculated and given as input to the *k*-NN classifier. The next process is to use the contour image to calculate blob feature values. In this project, three characteristics of a blob are used as blob features. The first feature is area A of blob, which can be simply calculated by counting the number of pixels within each blob. The second feature is circularity, which is a measure of roundness of a blob. The calculation for circularity C is described in Eq. 3.2. The third feature is the average grey-scale value of all pixels in a blob. The opening image (Figure 7-5) is used to calculate average of grey-scale color, which can be extracted by calculating the sum of grey-scale value for every pixel in each blob divided by its area.

After we obtained *area, circularity,* and *average grey-scale values,* the next step is to send these data to train with *k*-NN to remove the noise blob or non-fat blob. From our set of sample images, the system used *k*-NN algorithm to define feature space in a three-dimensional *k*-NN based decision graph: *circularity, area and average grey-scale.* The system then classifies all blobs in the feature space as shown in Figure 7-8 and determines usable ones that are fat by *k*-NN algorithm. Please see result in Table 7-1.

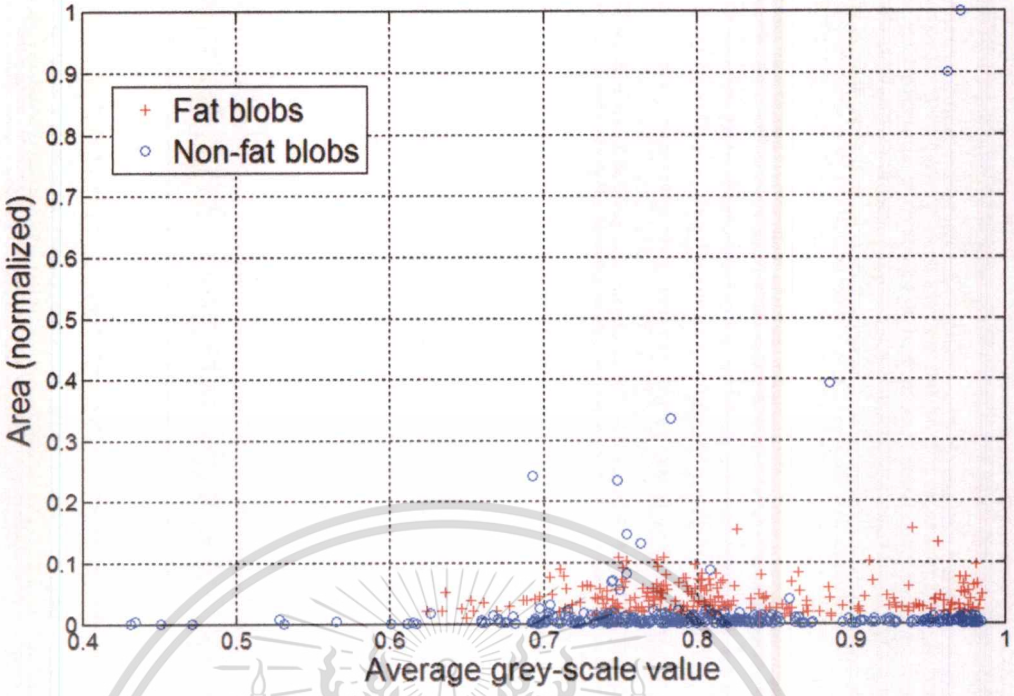
7.4 Noise blob removal by using *k*-NN result

As explained in Section 6.1.8, after we used the *k*-NN method to train images with seven ground truth images or sample images, the evaluation of the *k*-NN classifier was carried out. The result of the classification shows which blobs belong to fat class and which blobs belong to non-fat class. In our system, after experimenting with various odd values for *k*, the value 3 was selected because it gave a better accuracy rate for our sample images compared to $k = 5, 7, 9$ or 11.

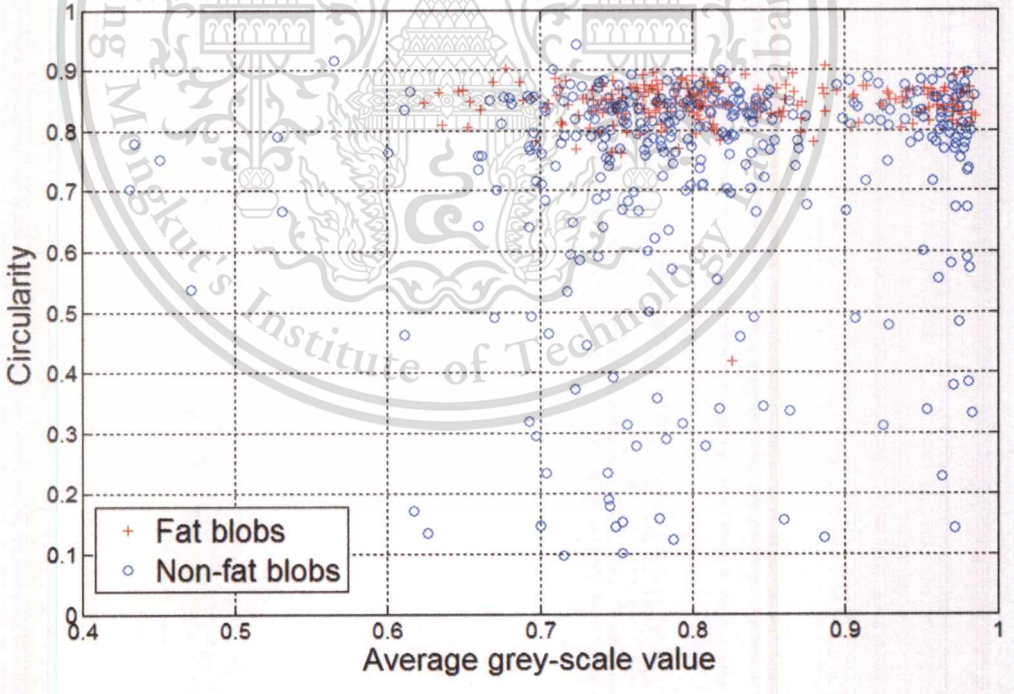
The result of blob classification shows a 3-dimensional feature space with features of *area, circularity,* and *average grey-scale* as shown in Figure 7-8(a) where "+" and "o" represent members of the two classes, which are fat blobs and non-fat blobs respectively. The area and average grey-scale values have been normalized into the range of [0, 1]. Please see result in Table 7-1.



(a)



(b)



(c)

Figure 7-8: Graph of feature space: (a) 3-D feature space, (b) 2-D feature space of area and average grey-scale, and (c) 2-D feature space of circularity and average grey-scale.

From the figure, it appears that the fat blobs (red cross) often have high circularity, high average grey-scale value and a moderate value of area (around hundreds to a few thousand pixels). Noise blobs, however, have a wide-range of circularity and average grey-scale value. Moreover, noise blobs often have a relatively small value of area, less than a hundred, although some of them are very large (several thousand pixels or larger)

Table 7-1: Blob classifying result

Blob no.	Area	Circularity	Avg. grey-scale	Classifying (1 = fat ,0 = non-fat)
0	6	0.4	227.66	0
1	22	0.54	226.26	0
2	37	0.85	246.07	0
3	24	0.8	236.03	0
4	24	0.8	242.9	0
5	117.5	0.69	231.15	0
6	198	0.83	245.27	0
7	158.5	0.83	245.06	0
8	49.5	0.81	234.77	0
9	984	0.76	246.74	1
10	183	0.71	249.10	0
11	50.5	0.83	236.56	0
12	220	0.81	242.11	0
13	30	0.82	241.68	0
14	1664	0.85	248.61	1
. . .	. . .	. . .	. . .	. . .
295	832.5	0.84	245.74	1
296	1319.5	0.88	244.4	1
297	62.5	0.65	245.01	0
298	23555.5	0.24	249.52	0

As shown in Table 7-1, if a blob is classified as *0*, it is considered as noise blobs, and will be removed by *k*-NN algorithm.

7.5 Calculation of fat areas result (%)

After the program processed the image and trained by k-NN algorithm, fat areas still remain white, where non-fat areas and background (empty slide areas) are in green and grey respectively. Please see result in Figure 7-9.

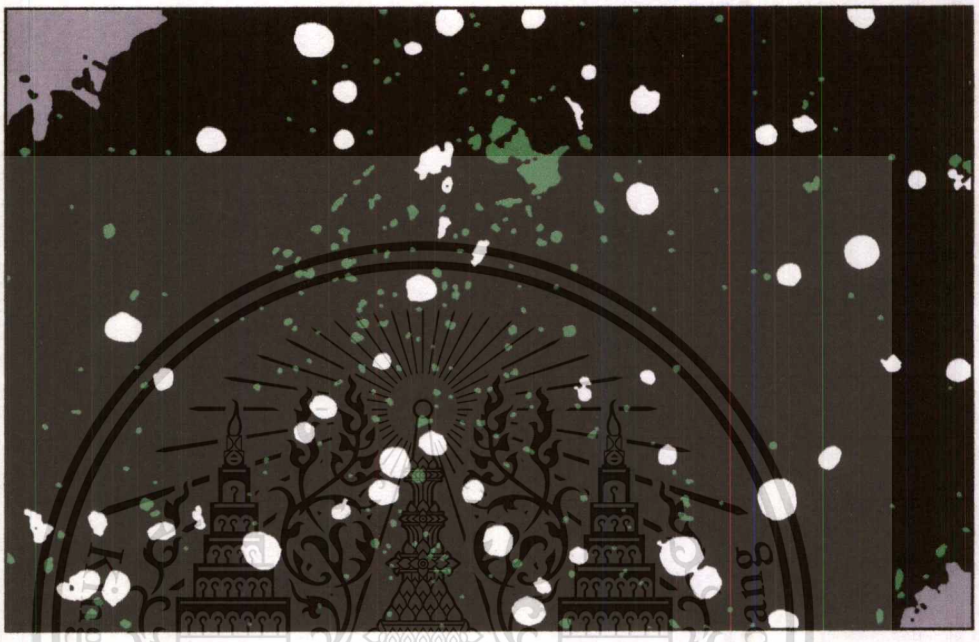


Figure 7-9: Processed image result

The program will show the percentage of all fat areas by using the following formula,

$$\frac{n_{fat}}{n_{fat}+n_{tissue}} \times 100\% \qquad (7.1)$$

where n_{fat} is a number of counted pixels in white color which is fat areas and n_{tissue} is a number of counted pixels of the liver tissue. Please see result in Table 7-4.

Table 7-2: Result of fat proportion

n_{tissue}	1200020
n_{fat}	63767
Result	**5.18%**

Chapter 8

Experiment

In this chapter, we have conducted two experiments in order to measure the effectiveness of the proposed method: evaluation of segmentation accuracy and result comparison between processed and ground truth images. In both experiments, ten liver biopsy images are used as a test set. Please note that these ten images are not the same as those we used for *k*-NN training.

8.1 Evaluation of segmentation accuracy

In the first experiment, we aim to evaluate the accuracy of fat segmentation. Three indicators will be used in this experiment namely true positive rate (TPR), false positive rate (FPR) and segmentation accuracy (ACC). The following equations describe the three indicators:

$$TPR = \frac{TP}{TP + FN} \times 100\% \tag{8.1}$$

$$FPR = \frac{FP}{FP + TN} \times 100\% \tag{8.2}$$

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% \tag{8.3}$$

where TP is the number of fat pixel that is correctly classified as fat, FP is the number of non-fat pixel that is incorrectly classified as fat, FN is the number of fat pixel that is incorrectly classified as non-fat, and TN is the number of non-fat pixel that is correctly classified as non-fat. The experiment results are shown in Table 8-1.

Table 8-1: Accuracy of fat segmentation

Image no.	TPR (%)	FPR (%)	ACC (%)
1	55.55	0.69	96.4
2	87.69	0.35	99
3	76.66	1.63	96.81
4	73.65	0.35	98.87
5	79.17	0.99	97.89
6	62.71	2.55	94.47
7	84.22	0.15	99.52
8	79.69	0.28	99.22
9	81.81	0.12	99.57
10	77	0.09	99.39
Average	**72.92**	**0.71**	**98.12**

From the Table, it indicates that the proposed fat segmentation algorithm can detect fat with the accuracy of 98.12%.

8.2 Result comparison between processed and ground truth images.

In the second experiment, we aim to confirm whether the fat proportion computed by the proposed method is accurate. It is compared with the fat proportion computed from the corresponding ground truth image, which is manually marked by hand. The experiment results are shown in Table 8-2.

Table 8-2: Comparison of fat proportion

Image no.	Percentage of fat proportion		Absolute difference
	Processed image (%)	Ground truth image (%)	
1	4.54	6.58	2.04
2	5.15	5.23	0.08
3	7.64	7.25	0.39
4	2.62	3.03	0.41
5	5.65	5.57	0.08
6	8.34	8.56	0.21
7	2.05	2.19	0.13
8	2.33	2.51	0.17
9	1.56	1.72	0.15
10	1.88	2.27	0.39
Average	**4.17**	**4.49**	**0.32**

From the Table, the average difference between the fat proportion computed by the proposed method and ground truth is 0.32 points. The result shows that the proposed method can calculate the fat proportion in each test image with a very small error, except the image no.1 (absolute difference of 2.04). The reason is that, there are some connected fat blobs in the image which cannot be separated by the pre-processing step. Consequently, *k*-NN classifier may consider those blobs to be noise blobs because their area are too large or their shapes are not round. Using a large filter size in morphological opening step might be able to separate these blobs; however, small fat blobs might be removed as well. In the future, a more accurate blob separation method should be researched in order to improve the accuracy.

Chapter 9

Conclusions

Our software system can be further developed and could be used by more hospitals or medical research and developmet insitutes because nowadays there is still no specific software used to analyze and segment fat in liver tissue from liver biopsy images. At least the pathologists or doctors could use this software to help them estimate the fat proportion in a more precise manner.

9.1 Limitation

9.1.1 Software possible mistakes in choosing fat areas

Whenever user clicks to diagnose, the program then highlights the fat areas which occurred in the image. But occasionally, the software made mistakes regarding the chosen fat areas. These mistakes can be divided into two main groups.

Group1 – Fat is not included: The program did not highlight the areas that *are fat.*

Group2 – Non-fat is included: The program highlighted the areas that *are actually* ***not*** *fat.*

Since these mistakes can happen, our software will provides a semi-automatic function for users to "Edit Image" (as described in Chapter 4) i.e. users can edit images manually and let the program re-calculate the result.

9.1.2 Image property requirements

In this project, the biopsy images used to develop the software have specific properties are follows. As a result, the biopsy images to be analysed by the software needs to have similar properties.

Table 9-1: Image property

Magnification	400x
Format	.TIFF (Tagged Image Format), detailed and adaptable format.
Height x Weight (pixels)	1435 x 924
Bit Depth	24

9.1.3 Number of images

After we discussed initial user requirements with the pathologists in the first semester, they sent us a set of liver biopsy images. As soon as we received the liver biopsy images, we then constructed the ground truths for some of the images to be used in our implementation phase. Unfortunately in the next semester, we got a chance to meet the pathologists again and we took our set of ground truth images to be confirmed by the pathologists. It turned out beyond their expectation; our ground truth images were too detailed. Consequently, we had to re-construct the ground truth images again according to their new requirements. This is part of the reasons that we had a limited number of ground truth images used in our implementation especially in image training stage with *k*-NN. Image training requires a lot of images to be trained and we had only seven. To be specific, *k*-NN feature space requires a lot of train data to be plotted to the graph to provide an accurate categorization. Also, we had a limited number of images in our experiment which is only one.

9.2 Lessons-learned

From the beginning of project after we had obtained the user and software requirements, we found that the techniques that we are going to use in the implementation phase such as image filters, image processing, image training stage and other related algorithms were new things to us. Therefore, we needed to spend a lot time on research and study so that our project plan duration was overlapped.

9.3 Further Work

In future work, we could include more features in image training stage (*k*-NN) to improve the diagnosing result. Different classifiers could be used to determine if the classification of blob regions can be improved. The manual segmentation of blob regions could be fully automated without user editing, so that the missing fat regions could be found.

Lastly, we could use develop some image processing techniques from this project to continue developong other liver image diagnosis, such as liver CT scan or MRI scan.

Bibliography

[1] L. A. Adams, P. Angulo, "Treatment of non-alcoholic fatty liver disease," 2006. Available from URL: http://en.wikipedia.org/wiki/Nonalcoholic_fatty_liver_disease

[2] R. O. Duda, P. E. Hart, and D. G. Stork, Pattern Classification (2nd edition), Wiley-Interscience, 2010.

[3] R. C. Gonzalez and R. E. Woods, Digital Image Processing (3rd edition), Prentice Hall, 2007.

[4] M. Gordan et al., "Bayesian segmentation of hepatic biopsy color images in the JPEG compressed domain," 8th WSEAS Conference, 2008, pp.140-145.

[5] P. W. Huang and Y. H. Lai, "Effective segmentation and classification for HCC biopsy images," Pattern Recognition, Vol. 43, 2010, pp.1550-1563.

[6] R. Laganière, OpenCV 2 Computer Vision Application Programming Cookbook, Packt Publishing, 2011.

[7] A. P. Paplinski. Lecture Notes, Dept. of Computer Sciences, and Software Eng. Manash University, Clayton-AUSTRALIA

[8] M. Swan and J. Ridgway, "Creating Measures Compactness". Available from URL: http://www.flaguide.org/tools/math/measures/compactness.php

[9] Conversion of a Color Image to a Binary Image. Retrieved: 2008-06-11. Available from URL: http://www.codersource.net/csharp_color_image_to_binary.aspx

Appendix A: Requirement questions

Questions for getting project requirements from RAMA hospital. We had studied and researched image processing, medical images, and tests for liver cancer. In order to obtain clear project requirements from the actual hospital, we prepared some questions to ask and discuss with the doctors and our project advisors.

1. Common questions

1.1 How many times a patient can have a liver biopsy?

1.2 Is a CT scan used to guide the placement of the needle?

1.3 Is the next biopsy done in the same area?

1.4 How is the tissue cut in the microscopic procedure? What does the tissue look like?

1.5 How long does a pathologist get the training?

1.6 Have to use a blood test result to support with the biopsy procedure?

1.7 Is a blood test result needed to support the biopsy procedure?

1.7 Hospital state-of-the-art?

2. Images,

2.1 What kinds of features are used to determine fat areas e.g. texture or color?

2.2 Environmental wise, does lighting affect the images? Which is the standard lighting used?

2.3 What is the standard to adjust the slide under the microscope e.g. lens, light sources?

2.4 In the image, is it already segmented to the focusing area?

2.5 What is ground truth image?

2.6 How to distinguish fat area and vessel in the image?

What if these two areas are connected?

3. Functions within the software,

3.1 Are Patients Information, Patient's ID number, and Date enough?

3.2 Does a user need to login to use the software?

3.3 What is the file format to be uploaded (image file, JPEG?)? Will it be from your own directory?

4. Evaluating results,

4.1 How to determine the percentage of fat in a biopsy slide?

How many percent of fat in the particular tissue indicates that the patient tends to have cirrhosis?

4.2 How to define the percentage?

4.3 How is the fat percentage calculated?

4.4 Result descriptions?

4.5 For a patient, will results of each tests need to be compared?

4.6 Do we need to save an image including diagnosis result?

Figure A-1: Questions for getting project requirements

Appendix B: Software screenshots

All screenshots of the software are shown in this section. For more details regarding what was discussed in the software requirements, please refer to Chapter 4.

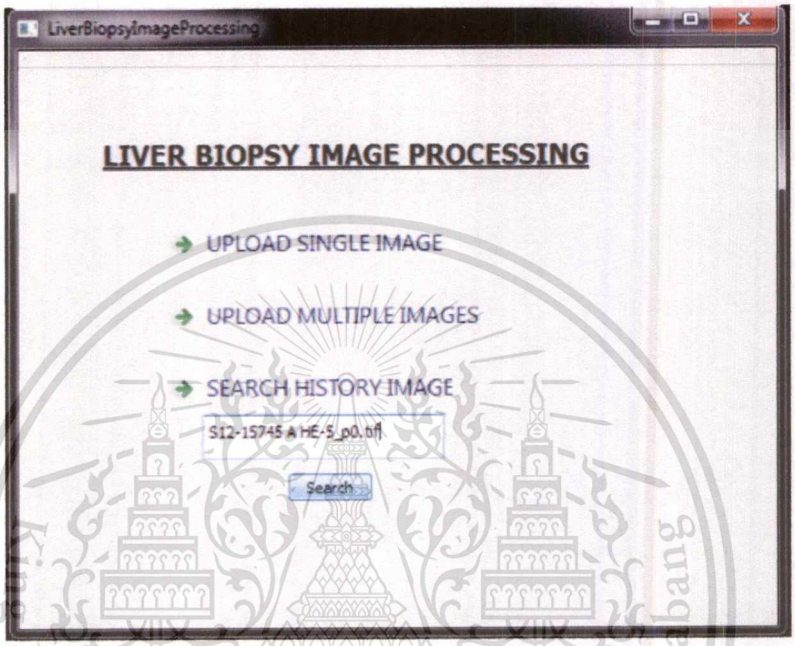
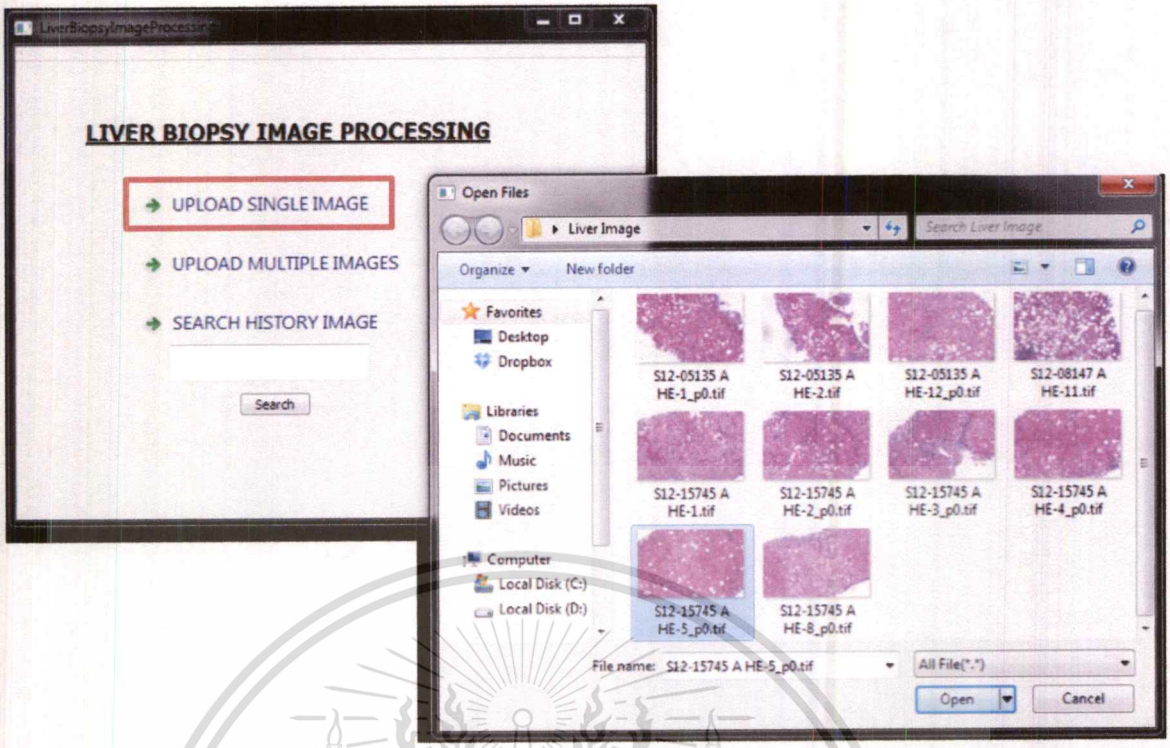
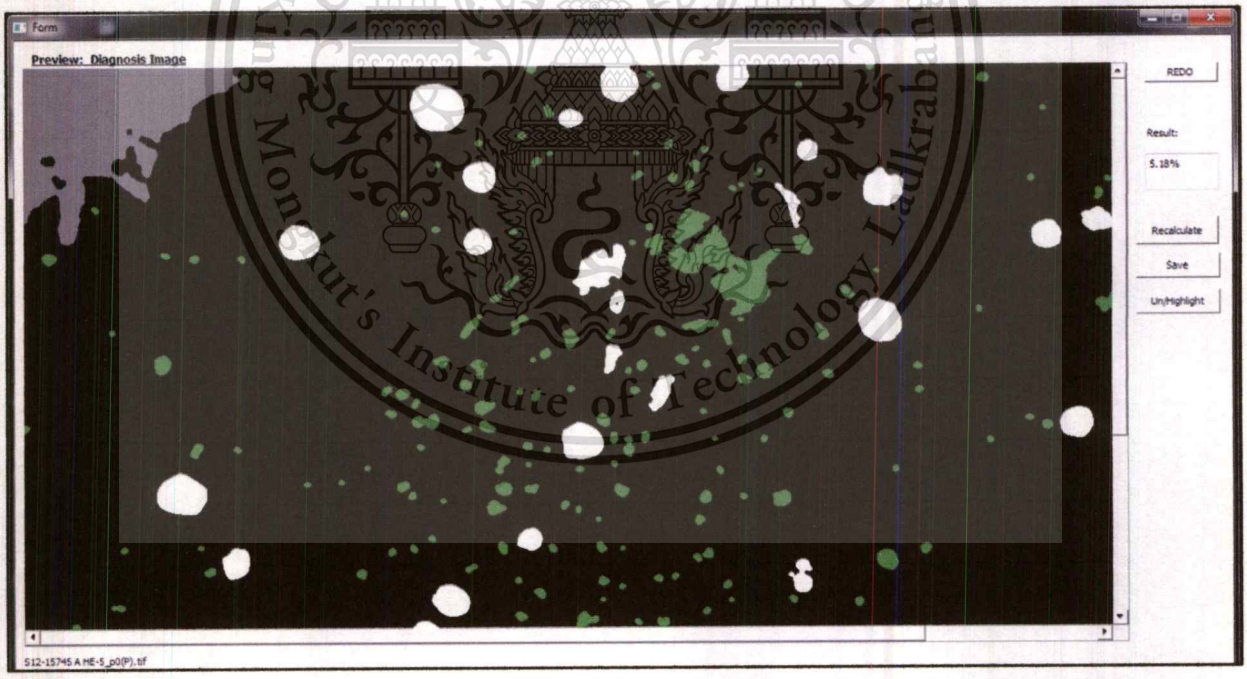


Figure B-1: Main menu

Figure B-1 shows the initial screen after user opens the program. User may choose from the main menu to start the activities. User can upload an original image of liver biopsy or multiple images for diagnosis. User can also search for an image by file name.



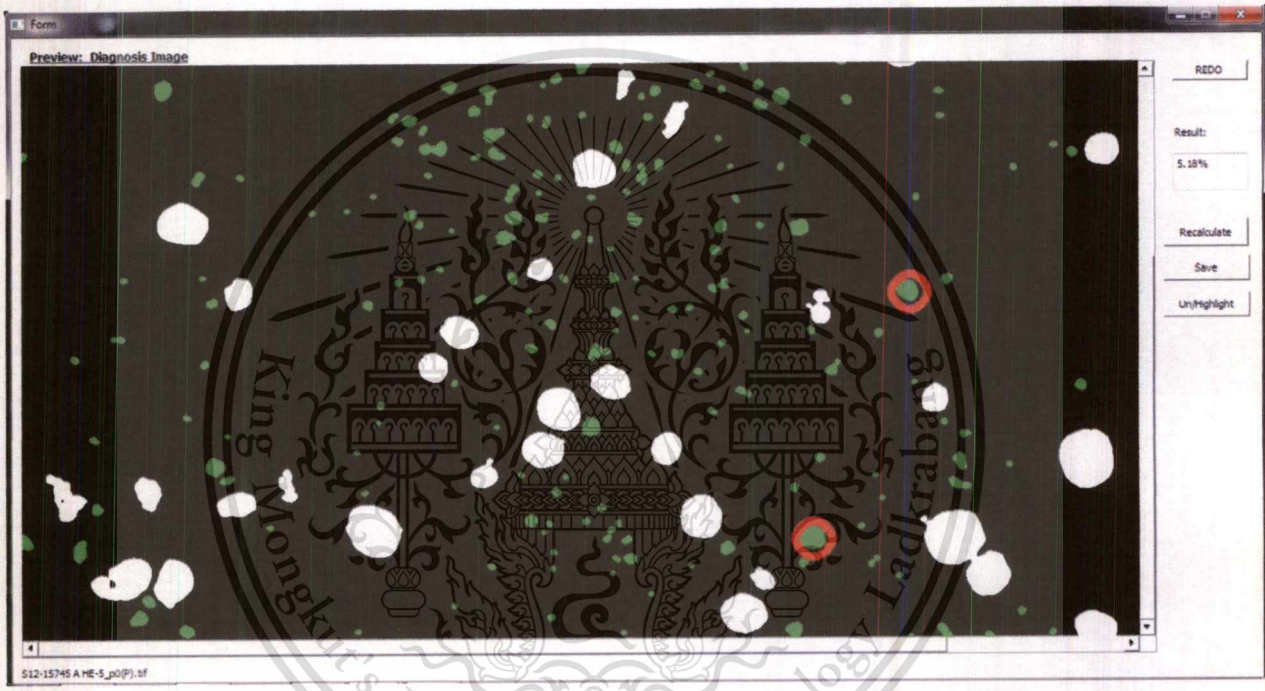
(a)



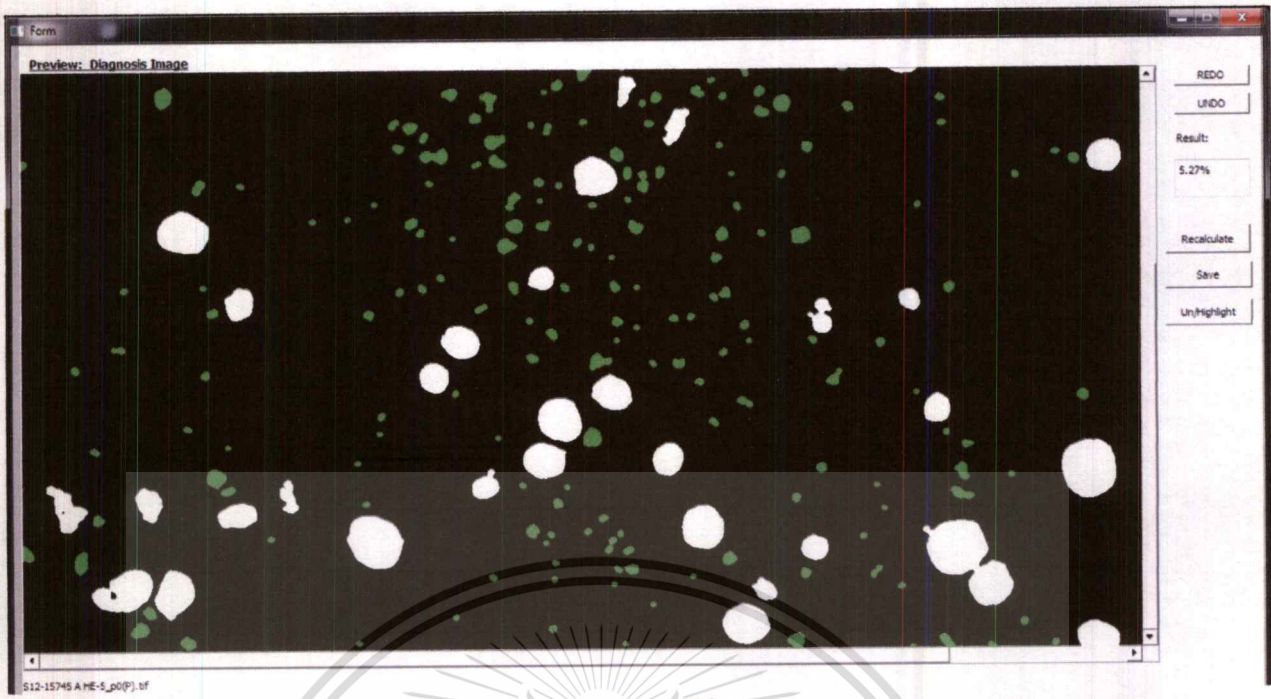
(b)

Figure B-2: Single image upload: (a) browsing one image and (b) diagnosis page

When user chose to upload a single image, user will browse one image through user's directory (Figure B-2(a)). The image will be processed, calculated and the result will be shown in the diagnosis page (Figure B-2(b)). In diagnosed image, fat areas, non-fat areas and background empty slide areas are indicated in white, green and grey respectively. The diagnosed result is calculated from fat areas and shown in percentage of fat proportion. User can click "Un/Highlight" button to preview undiagnosed (original) image or diagnosed image. Please note that after the image was processed or diagnosed, the image file name shall be changed to "filename(P).tiff".



(a)



(b)

Figure B-3: Edit image: (a) adding fat areas and (b) after image is edited

In case of image editing, user can add or remove fat areas by clicking any areas on the diagnosed image (Figure B-3(a)). User can also undo the activity by clicking "UNDO" button. After user finished editing, user should click "Recalculate" button to re-calculate the result. The diagnosed result may be increased or decreased depending on what user has edited (Figure B-3(b)). Finally, user can save the diagnosed image to any directory in the user's computer.

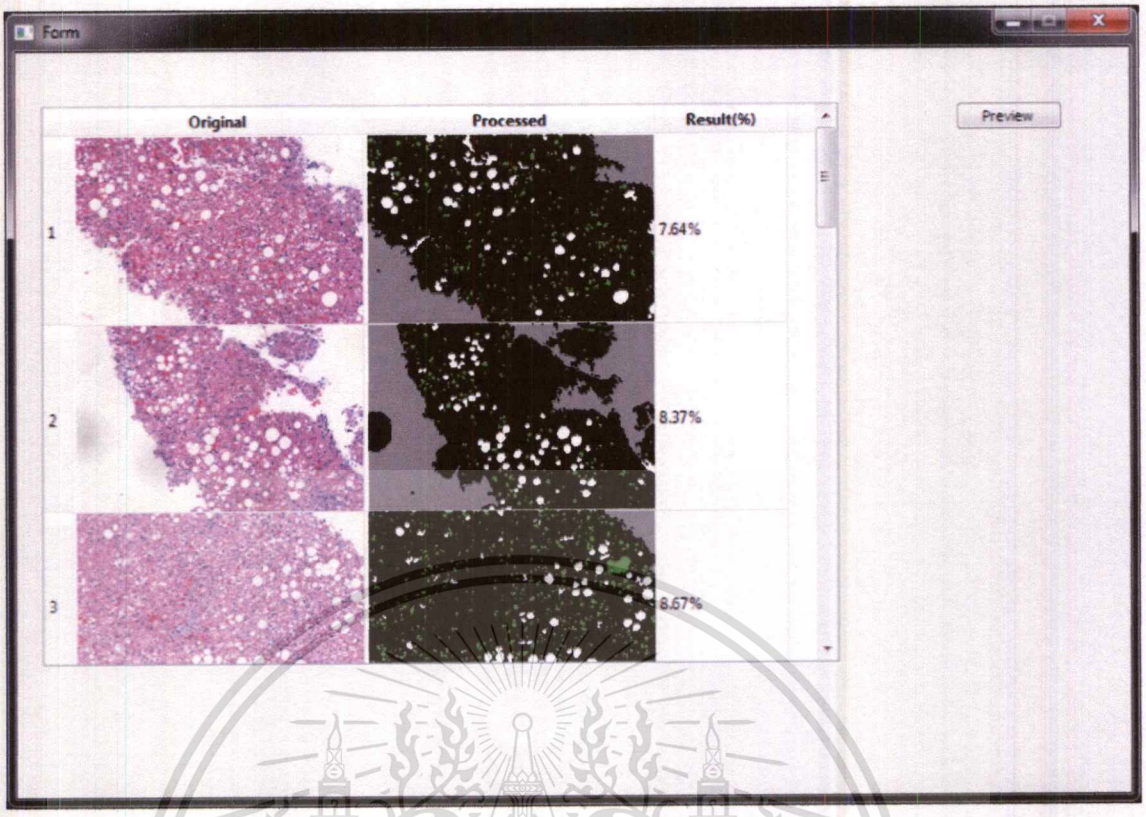


Figure B-4: Multiple images upload

User can also upload more than one image to be diagnosed by choosing to upload multiple images. User will browse the images through user's directory in the same way as in single image upload mode. All uploaded images will be processed and shown with the result in a form of table (Figure B-4). User may choose any image from the list and click "Preview" button that will lead to the diagnosis page in case the user wants to edit the image.

Appendix C: Comparison of moving-average filter values in low-pass filtering

In order to find the best moving-average filter value to be used for low-pass filtering, we conducted a small experiment to test various values as shown in the following tables. As one can see, a 7x7 filter yielded the best results for the tested images. Therefore, we chose 7x7 moving-average filter value in our implementation.

Table C-1: 5x5 moving-average filter results

Image no.	Percentage of fat proportion		Absolute difference
	Processed image (%)	Ground truth image (%)	
1	4.67	6.58	1.91
2	4.96	5.23	0.27
3	7.31	7.25	0.05
4	2.61	3.03	0.42
5	5.36	5.57	0.2
6	7.57	8.56	0.98
7	2.13	2.19	0.06
8	2.21	2.51	0.29
9	1.51	1.72	0.21
10	1.84	2.27	0.43
Average	**4.01**	**4.49**	**0.48**

Table C-2: 7x7 moving-average filter results

Image no.	Percentage of fat proportion		Absolute difference
	Processed image (%)	Ground truth image (%)	
1	4.54	6.58	2.04
2	5.15	5.23	0.08
3	7.64	7.25	0.39
4	2.62	3.03	0.41
5	5.65	5.57	0.08
6	8.34	8.56	0.21
7	2.05	2.19	0.13
8	2.33	2.51	0.17
9	1.56	1.72	0.15
10	1.88	2.27	0.39
Average	**4.17**	**4.49**	**0.32**

Table C-3: 9x9 moving-average filter results

Image no.	Percentage of fat proportion		Absolute difference
	Processed image (%)	Ground truth image (%)	
1	4.93	6.58	1.65
2	4.93	5.23	0.3
3	7.66	7.25	0.41
4	2.58	3.03	0.45
5	5.61	5.57	0.03
6	8.11	8.56	0.44
7	2.01	2.19	0.18
8	2.36	2.51	0.14
9	1.53	1.72	0.18
10	1.81	2.27	0.47
Average	**4.15**	**4.49**	**0.42**

Table C-4: 11x11 moving-average filter results

Image no.	Percentage of fat proportion		Absolute difference
	Processed image (%)	Ground truth image (%)	
1	4.86	6.58	1.72
2	4.93	5.23	0.31
3	8.21	7.25	0.95
4	2.56	3.03	0.47
5	5.96	5.57	0.39
6	8.85	8.56	0.29
7	1.99	2.19	0.19
8	2.81	2.51	0.29
9	1.59	1.72	0.12
10	1.9	2.27	0.37
Average	**4.36**	**4.49**	**0.51**

Table C-5: 13x13 moving-average filter results

Image no.	Percentage of fat proportion		Absolute difference
	Processed image (%)	Ground truth image (%)	
1	4.86	6.58	1.72
2	5.3	5.23	0.06
3	8.88	7.25	1.62
4	3.05	3.03	0.01
5	6.15	5.57	0.58
6	9.77	8.56	1.21
7	2.08	2.19	0.11
8	2.75	2.51	0.24
9	1.69	1.72	0.03
10	2.13	2.27	0.14
Average	**4.66**	**4.49**	**0.57**

Table C-6: 15x15 moving-average filter results

Image no.	Percentage of fat proportion		Absolute difference
	Processed image (%)	Ground truth image (%)	
1	5.11	6.58	1.72
2	5.13	5.23	0.06
3	8.23	7.25	1.62
4	3.21	3.03	0.01
5	6.07	5.57	0.58
6	9.38	8.56	1.21
7	2.05	2.19	0.11
8	2.61	2.51	0.24
9	1.63	1.72	0.03
10	1.99	2.27	0.14
Average	**4.54**	**4.49**	**0.57**

Appendix D: Source Code

Liver_Process.cpp

```
#include "Liver_Process.hpp"
#include <direct.h>
#include "qstringlist.h"
double maxTrainedArea = 0;
void fatSegment( const cv::Mat& input, cv::Mat& output, CvKNearest* trainedKNN ,const string NameImg ) {
cv::Mat blobImg( input.rows, input.cols, input.type() );
detectBlob(  input, blobImg );
SetOfContours contours;
findContour( blobImg, contours );
vector<BlobFeatures> blobFts;
findAreaCircularity( contours, blobFts );
findAvgGreyLevel(input, blobImg, contours, blobFts);
detectBG(contours ,blobFts );
removeNonFatBlobs(blobImg,blobFts,contours,trainedKNN,output,NameImg );
cvtTissue(NameImg);
mergedEditImage(NameImg);
mergedImage(NameImg);
}
void detectBlob( const cv::Mat& input, cv::Mat& blobImg ){
cv::Mat image1, image2;
cv::blur( input, image1, cv::Size( 7, 7 ) );
//Threshold
cv::threshold( image1, image2, 220, 255 ,cv::THRESH_BINARY);
//Openning
cv::Mat element = getStructuringElement(cv::MORPH_ELLIPSE ,cv::Size(7,7));
cv::morphologyEx(image2,blobImg,cv::MORPH_OPEN,element);
cv::imwrite("open.bmp",blobImg);
}
void findContour( const cv::Mat& blobImg, SetOfContours& contours){
cv::findContours(blobImg, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE);
cv::Mat contourImg (blobImg.size(),CV_8U, cv::Scalar(255));
```

```
cv::drawContours(contourImg, contours, -1,cv::Scalar(0),1);
}
void findAreaCircularity( const SetOfContours& contours, vector<BlobFeatures>& blobFts){
double area ;
double perimeter, circularity;
BlobFeatures ft;
for(int i =0 ; i<contours.size() ;i++){
area = cv::contourArea(contours[i], 0);
perimeter = cv::arcLength(contours[i], 1);
circularity = calCircularity(area, perimeter);
ft.setArea( area );
ft.setCircularity( circularity );
blobFts.push_back( ft );
}
}
double calCircularity(double area,double length ){
return 4*PI*area/(length*length);
}
void findAvgGreyLevel( const cv::Mat& input, const cv::Mat& blobImg,const SetOfContours& contours,
vector<BlobFeatures>& blobFts ){
int fill ;
double sum;
int count;
cv::Mat temp;
temp = blobImg.clone();
for(int n =0;n<contours.size();++n){
int cMax = contours[n][0].x;
int cMin = contours[n][0].x;
int rMax = contours[n][0].y;
int rMin = contours[n][0].y;
for(int i = 1 ;i<contours[n].size();++i){
if( cMax <  contours[n][i].x ){
cMax = contours[n][i].x;
}
if( cMin > contours[n][i].x ){
```

```
cMin = contours[n][i].x ;
}
if( rMax < contours[n][i].y){
rMax = contours[n][i].y;
}
if( rMin > contours[n][i].y){
rMin = contours[n][i].y;
}
}
fill = cv::floodFill(temp, contours[n][0], 128);
sum = 0;
count =0;
for(int i = rMin; i<rMax; ++i){
for(int j = cMin; j<cMax; ++j)
if(temp.at<uchar>(i, j) == 128 ){
sum += input.at<uchar>(i,j);
count++;
}
}
sum = sum/count;
blobFts[n].setAvgGreyLevel( sum );
cout << sum << endl;
fill = cv::floodFill(temp, contours[n][0], 255);
}
}
void removeNonFatBlobs(const cv::Mat& blobImg, const vector<BlobFeatures>& blobFts,const SetOfContours&
contours,CvKNearest* trainedKNN, cv::Mat& output,const string NameImg ){
ofstream outFile;
outFile.open( "Result Image.txt" );
cv::Mat& input = cv::imread("open.bmp",1);
cv::Mat a_ftVector( 1, NUM_FTS, CV_32FC1 );
float response;
int filling ;
int i = 0;
for( i = 0; i < blobFts.size(); i++) {
```

```
a_ftVector.at<float>(0,0) = blobFts[i].getArea();
a_ftVector.at<float>(0,1) = blobFts[i].getCircularity();
a_ftVector.at<float>(0,2) = blobFts[i].getAvgGreyLevel();
response = trainedKNN->find_nearest(a_ftVector,K);
if(response == 0 ){      /*1 = fat , 0 = non-fat*/
filling = cv::floodFill(input, contours[i][1], cv::Scalar(0, 255.0, 0));
}
}
cv::imwrite("Highlight.bmp",input);
cv::imwrite("Highlight2.bmp",input);
}

void trainKNN( const string dbFileName, CvKNearest*& trainedKNN ){
string gtImg;
int numImg;
string fileExt, fileName, end1;
ifstream inFile("train_image_list.txt");
inFile >> numImg;
inFile.ignore(1);
getline( inFile, fileExt );
cv::Mat gt ;
cv::Mat input;
cv::Mat ftVector,labelVector;
cv::Mat trainGT;
SetOfContours contours;
vector<int> labels;
vector<BlobFeatures> blobFts;
string path = "C:/Users/DeTonic/Documents/Visual Studio 2010/Projects/LiverOpenCV";
if(numImg > 0){
for(int i=0; i<numImg; i++){
getline( inFile, fileName );
fileName += fileExt;
gtImg = "(BG)"+fileName;
cout << gtImg << endl ;
gt = cv::imread(gtImg, 0);
```

```
input =cv::imread(fileName,1);
findContourFromGT(gt, contours, labels);
trainGT = gt.clone();
calculateTrainingFeatures(trainGT,input,blobFts,labels);
}
}
convertSampleToFtVector(blobFts, labels, ftVector, labelVector );
train(ftVector,labelVector, trainedKNN );
}
void findContourFromGT( const cv::Mat& blobImg, SetOfContours& contours, vector<int>& labels ) {
SetOfContours tempContours;
vector<BlobFeatures> blobFts;
vector<int> tempLabels;
cv::Mat targetArea( blobImg.rows, blobImg.cols, blobImg.type() );        tempContours.clear();
tempLabels.clear();
detectColor( blobImg, NON_FAT_COLOR, targetArea );
cv::findContours(targetArea, tempContours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE);
contours.insert( contours.end(), tempContours.begin(), tempContours.end() );
tempLabels = vector<int>( tempContours.size(), NON_FAT_AREA );
labels.insert( labels.end(), tempLabels.begin(), tempLabels.end() );
detectColor( blobImg, FAT_COLOR, targetArea );
cv::findContours(targetArea, tempContours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_NONE);
cv::Mat contourImg (targetArea.size(),CV_8U, cv::Scalar(255));
contours.insert( contours.end(), tempContours.begin(), tempContours.end() );
tempLabels = vector<int>( tempContours.size(), FAT_AREA );
labels.insert( labels.end(), tempLabels.begin(), tempLabels.end() );
}
void detectColor( const cv::Mat& img, const int color, cv::Mat& out ) {
for( int i = 0; i < img.rows; i++ )
for( int j = 0; j < img.cols; j++ ){
cout <<(int)img.at<uchar>(i, j)<< "  " << color << endl;*/
int val = (int)img.at<uchar>(i, j);
if( val == color )
out.at<uchar>(i, j) = 255;
else
```

```
out.at<uchar>(i, j) = 0;
}
}
void calculateTrainingFeatures(const cv::Mat& trainGT, const cv::Mat& input, vector<BlobFeatures>& blobFts, vector<int>& labels){
vector<int> tempLabels;
cv::Mat fatImg,nonfatImg;
nonfatImg = trainGT.clone();
fatImg = trainGT.clone();
for(int i = 0; i<nonfatImg.rows; ++i){
for(int j = 0; j<nonfatImg.cols; ++j)
if(nonfatImg.at<uchar>(i, j) == 128 ){
nonfatImg.at<uchar>(i, j) = 255;
}
else{
nonfatImg.at<uchar>(i,j) = 0;
}
}
for(int i = 0; i<fatImg.rows; ++i){
for(int j = 0; j<fatImg.cols; ++j)
if(fatImg.at<uchar>(i, j) == 0 ){
fatImg.at<uchar>(i,j) = 255;
}
else{
fatImg.at<uchar>(i,j) = 0;
}
}
cv::Mat blobImg1 = nonfatImg.clone();
SetOfContours contours1;
findContour(blobImg1, contours1);
vector<BlobFeatures> blobFts1;
findAreaCircularity(contours1, blobFts1);
findAvgGreyLevel(input,nonfatImg,contours1, blobFts1);
vector<int> tempLabels1 = vector<int>( blobFts1.size(), NON_FAT_AREA );
cv::Mat blobImg2 = fatImg.clone();
```

```
SetOfContours contours2;
findContour(blobImg2, contours2);
vector<BlobFeatures> blobFts2;
findAreaCircularity(contours2, blobFts2);
findAvgGreyLevel(input,fatImg,contours2, blobFts2);
vector<int> tempLabels2 = vector<int>( blobFts2.size(), FAT_AREA );
blobFts.insert( blobFts.end(), blobFts1.begin(), blobFts1.end() );
blobFts.insert( blobFts.end(), blobFts2.begin(), blobFts2.end() );
}
void convertSampleToFtVector( const vector<BlobFeatures>& blobFts, const vector<int>& labels, cv::Mat& ftVector,
cv::Mat& labelVector ){
ftVector = cv::Mat( blobFts.size(), NUM_FTS, CV_32FC1 );
labelVector = cv::Mat( labels.size(), 1, CV_32FC1 );
maxTrainedArea = 0.0;
for(int i = 0; i < blobFts.size(); i++)
if(blobFts[i].getArea() > maxTrainedArea)
maxTrainedArea = blobFts[i].getArea();
for(int i = 0; i < blobFts.size(); i++) {
ftVector.at<float>(i,0) = blobFts[i].getArea();
ftVector.at<float>(i,1) = blobFts[i].getCircularity();
ftVector.at<float>(i,2) = blobFts[i].getAvgGreyLevel();
labelVector.at<float>(i,0) = labels[i];
}
}
void train( const cv::Mat& ftVector, const cv::Mat& labelVector, CvKNearest*& trainedKNN ){
trainedKNN = new CvKNearest( ftVector, labelVector );
}
void detectBG(SetOfContours& contours , vector<BlobFeatures>& blobFts ){
const int bgAreaTH = 3000;
const int contourLenTH = 500;
const int edgePixelTh = 100;
cv::Mat& input = cv::imread("open.bmp",0);
const int maxX = input.cols-2;
const int maxY = input.rows-2;
for(int i =0; i<contours.size();i++){
```

```
int area = blobFts[i].getArea();
int len = contours[i].size();
if(area>bgAreaTH || len>contourLenTH){
int countEdgePixel = 0;
for( int n = 0; n < contours[i].size(); ++n ) {
int x = contours[i][n].x;
int y = contours[i][n].y;
if( (x == 1) || (x == maxX) || (y == 1) || (y == maxY) )
 countEdgePixel++;
}
if( countEdgePixel > edgePixelTh ) {
 cv::floodFill( input, contours[i][ len/2 ], 128 );
}
cv::imwrite("detectBgImg.bmp",input);
}
}
}
void cvtTissue(const string NameImg){
cv::Mat Highlight = cv::imread("Highlight.bmp",1);
cv::Mat tissueImg = cv::imread( NameImg, 1);
cv::Mat mergedImg = tissueImg.clone();
cv::imwrite("mergedImg.bmp",mergedImg);
for( int i = 0; i < Highlight.rows; i++ )
for( int j = 0; j < Highlight.cols; j++ ) {
if( Highlight.at<cv::Vec3b>(i,j)[0] == 255 &&
Highlight.at<cv::Vec3b>(i,j)[1] == 255 &&
Highlight.at<cv::Vec3b>(i,j)[2] == 255 ) {
tissueImg.at<cv::Vec3b>(i,j)[0] = tissueImg.at<cv::Vec3b>(i,j)[0] / 2;//B
tissueImg.at<cv::Vec3b>(i,j)[1] = (tissueImg.at<cv::Vec3b>(i,j)[1] +255)/2;//G
tissueImg.at<cv::Vec3b>(i,j)[2] = tissueImg.at<cv::Vec3b>(i,j)[2] / 2;//R}
}cv::imwrite("Highlight.bmp",tissueImg);
}void mergedImage(const string NameImg){
cv::Mat CutBG; // Cut Background Image
cv::Mat afterProcessImg; // Highlight Image After knn
cv::Mat mergedImg;
```

```
CutBG = cv::imread("detectBgImg.bmp",0);
afterProcessImg = cv::imread("Highlight.bmp",1);
mergedImg = afterProcessImg.clone();
cv::imwrite("mergedImg.bmp",mergedImg);
for( int i = 0; i < CutBG.rows; i++ )
for( int j = 0; j < CutBG.cols; j++ ) {
if( CutBG.at<uchar>(i,j) == 128 ) {
mergedImg.at<cv::Vec3b>(i,j)[0] = 128;          //B
mergedImg.at<cv::Vec3b>(i,j)[1] = 128;          //G
mergedImg.at<cv::Vec3b>(i,j)[2] = 128;          //R
}
else if( CutBG.at<uchar>(i,j) == 128 ){
mergedImg.at<cv::Vec3b>(i,j)[0] = 0;
mergedImg.at<cv::Vec3b>(i,j)[1] = 0;
mergedImg.at<cv::Vec3b>(i,j)[2] = 0;
}
}
cv::imwrite("FatImage.bmp",mergedImg);
 int num =0 ;
 for(int i =0;i<NameImg.size();i++){
 if(NameImg[i] == 46){
 num = i;
 }
 }
 string temp,temp2,flodername;
 for(int j = NameImg.size()-1;j>=0;j--){
 if(NameImg[j] == 47){
 for(int k = j+1; k<=NameImg.size()-1;k++){
 if(NameImg[k] != 46){
temp += NameImg[k];
 }else{
break;
 }
 }
 flodername = NameImg.substr(0,j+1)+temp;
```

```
mkdir(flodername.c_str());
temp2 = NameImg.substr(0,j+1)+temp+"/"+temp+".tif";
cv::imwrite(temp2,mergedImg);
break;
}
}
for(int i =0;i<NameImg.size();i++){
if(NameImg[i] == 46){
cv::imwrite(NameImg.substr(0,i)+"(P)"+NameImg.substr(i,NameImg.size()-1),mergedImg);
}
}
}
void mergedEditImage(const string NameImg){
cv::Mat CutBG;
cv::Mat afterProcessImg;
cv::Mat mergedImg;
CutBG = cv::imread("detectBgImg.bmp",0);
afterProcessImg = cv::imread("Highlight2.bmp",1);
mergedImg = afterProcessImg.clone();
cv::imwrite("mergedImg2.bmp",mergedImg);
for( int i = 0; i < CutBG.rows; i++ )
for( int j = 0; j < CutBG.cols; j++ ) {
if( CutBG.at<uchar>(i,j) == 128 ) {
mergedImg.at<cv::Vec3b>(i,j)[0] = 128;          //B
mergedImg.at<cv::Vec3b>(i,j)[1] = 128;          //G
mergedImg.at<cv::Vec3b>(i,j)[2] = 128;          //R
}
else if( CutBG.at<uchar>(i,j) == 128 ){
mergedImg.at<cv::Vec3b>(i,j)[0] = 0;
mergedImg.at<cv::Vec3b>(i,j)[1] = 0;
mergedImg.at<cv::Vec3b>(i,j)[2] = 0;
}
}
cv::imwrite("FatImage2.bmp",mergedImg);
int num =0 ;
```

```
for(int i =0;i<NameImg.size();i++){
if(NameImg[i] == 46){
num = i;
}
}
string temp,temp2,flodername;
for(int j = NameImg.size()-1;j>=0;j--){
if(NameImg[j] == 47){
for(int k = j+1; k<=NameImg.size()-1;k++){
if(NameImg[k] != 46){
temp += NameImg[k];
}
else{
break;
}
}
flodername = NameImg.substr(0,j+1)+temp;
mkdir(flodername.c_str());
temp2 = NameImg.substr(0,j+1)+temp+"/"+temp+".tif";
cv::imwrite(temp2,mergedImg);
break;
}
}
for(int i =0;i<NameImg.size();i++){
if(NameImg[i] == 46){
cv::imwrite(NameImg.substr(0,i)+"(P)"+NameImg.substr(i,NameImg.size()-1),mergedImg);
}
}
}
```

Liver_Process.hpp

```
#ifndef LIVER_PROCESS_HPP
#define LIVER_PROCESS_HPP
#include <iostream>
#include <string>
#include <vector>
#include <cstdio>
#include <fstream>
#include <opencv2\core\core.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\imgproc\imgproc.hpp>
#include <opencv2\ml\ml.hpp>
using namespace std;
const double PI = 3.141592;
const int FAT_AREA = 1;
const int NON_FAT_AREA = 0;
const int BG_AREA = 2;
const uchar FAT_COLOR = 0;
const uchar NON_FAT_COLOR = 128;
const uchar BG_COLOR = 255;
const int NUM_FTS = 3;
typedef vector<vector<cv::Point>> SetOfContours;
const int K = 3;
class BlobFeatures {
public:
double getArea() const { return area; }
double getCircularity() const { return circularity; }
double getAvgGreyLevel() const { return avgGreyLevel; }
void setArea( double val ) { area = val; }
void setCircularity( double val ) { circularity = val; }
void setAvgGreyLevel( double val ) { avgGreyLevel = val; }
private:
double area;
double circularity;
double avgGreyLevel;
```

```
};

void fatSegment( const cv::Mat& input, cv::Mat& output, CvKNearest* trainedKNNm, const string NameImg);

void detectBlob( const cv::Mat& input, cv::Mat& blobImg );

void findContour( const cv::Mat& blobImg, SetOfContours& contours );

void findAreaCircularity( const SetOfContours& contours, vector<BlobFeatures>& blobFts );

double calCircularity(double area,double length);

void findAvgGreyLevel(  const cv::Mat& input,  const cv::Mat& blobImg,const SetOfContours& contours,
vector<BlobFeatures>& blobFts );

void removeNonFatBlobs(const cv::Mat& blobImg, const vector<BlobFeatures>& blobFts,const SetOfContours&
contours,CvKNearest* trainedKNN, cv::Mat& output,const string NameImg );

void trainKNN( const string dbFileName, CvKNearest*& trainedKNN );

void findContourFromGT( const cv::Mat& blobImg, SetOfContours& contours, vector<int>& labels );

void detectColor( const cv::Mat& img, const int color, cv::Mat& out );

void calculateTrainingFeatures(const cv::Mat& trainGT, const cv::Mat& input, vector<BlobFeatures>& blobFts, vector<int>&
labels);

void convertSampleToFtVector( const vector<BlobFeatures>& blobFts, const vector<int>& labels, cv::Mat& ftVector,
cv::Mat& labelVector );

void train( const cv::Mat& ftVector, const cv::Mat& labelVector, CvKNearest*& trainedKNN );

void detectBG(SetOfContours& contours , vector<BlobFeatures>& blobFts );

void mergedImage(const string NameImg);

void cvtTissue(const string NameImg);

void mergedEditImage(const string NameImg);

#endif
```