IMAGE PROCESSING ON MEDICAL APPLICATION

AUTOMATIC METHODS TO CALCULATE THE AREA OF AN ARTICULAR CARTILAGE ON A MAGNETIC RESONANCE IMAGE

By

Ngo Quang Long

Submitted in the fulfilment of the
Requirements for the Degree of Master of Engineering
School of Engineering, University of Tasmania (July, 2011)

School Of Engineering

University Of Tasmania

Statement of Originality

The work contained in this thesis has not been previously submitted for a degree or diploma at any higher education institution. To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made.

Signed : _____

Date : _____

Statement of Authority of Access

This thesis may be made available for loan and limited copying in accordance with the *Copyright Act 1968.*

Signed : _____

Date : _____

Publication Associated This Research

In this research, some works have been accepted to publish in International Workshop on Advanced Computational Intelligence (IWACI) Conference 2010.

- Ngo, Q and Jiang, D and Ding, C, Application of Artificial Neural Networks in Automatic Cartilage Segmentation', *Proceedings of IWACI2010*, 25-27 August 2010, Suzhou, China.

ABSTRACT

Digital image processing plays a more and more important role in the medical diagnoses. Most widely used method for visualizing internal anatomical features of human body is magnetic resonance imaging (MRI). MRI can be used for the detection of osteoarthritis (OA) which affects the cartilage in the joints. Several techniques for the segmentation of the cartilage in MRI scans of the knee have recently been developed. One goal of segmentation is to automatically determine area and volume of the cartilage. Due to noise, however, none of these approaches is satisfactory in fully automated segmentation of articular cartilage.

In our research, we attempt to study and develop new automatic methods to extract the cartilage from MRI knee scans. From cartilage images, cartilage area and volume are then computed. Three automatic methods are applied for cartilage extraction.

The first method is bi-directional scanning segmentations method (BSSM), which is based on two basic properties of intensity value: discontinuity (edge detection algorithms) and similarity (thresholding algorithms). It is also based on statistical analysis (curve fitting algorithms and average weight calculation).

The second method is neural network classifier method (NNCM), which is based on artificial neural network. For each pixel on an input image, through a neural network classifier, it is classified as a cartilage pixel if network output value is 1. Alternatively, it is classified as a background pixel if network output value is 0.

The third method is active contour models method (ACMM), which uses an initial contour that approximates the boundary of a cartilage to find the "actual" boundary. This is an innovative method because we apply BSSM to define the initial contour. We also apply NNCM to compute the external energy in active contour models algorithms.

Our three methods have succeeded in automatically extracting the cartilage from input image. The cartilage area and volume obtained from cartilage image, which is extracted by BSSM, NNCM, and ACMM, are highly correlation with the results obtained from reference cartilage image (correlation value in each case $p \approx 1$, $R^2 \approx 1$). Thus, cartilage area and volume assessments are precise, reliable and acceptable. Among those methods, the ACMM provides the best results. Considering noise and complexity of the image, each method has both advantages and disadvantages. BSSM work well where there is significant high contrast between cartilage and background

regions. It often fails where the contrast is low. NNCM can work well in low contrast. However, this method does not work accurately if the cartilage pixels have similar features of background pixels. Those pixels are classified as background pixels. As a result, this method reduces number of cartilage pixels. The third method, ACMM demonstrates higher accuracy in extracting cartilage from an input image. ACMM not only can take advantages of BSSM and NNCM but also can solve the problems existing in BSSM and NNCM. Therefore, results obtained from ACMM are the most precise and acceptable.

ACKNOWLEDMENT

I would like to thank my supervisor, Dr Danchi Jiang, for his enthusiasm and guidance, and for providing his image processing and computer vision expertise. I would also like to thank Dr. Changhai Ding for providing MRI images as experiment samples in my research as well as knowledge about cartilage on MR image.

This research was partly supported by the Menzies Research Institute, and by the School of Engineering, Tasmania of University.

CONTENTS

LIST OF TABLES AND FIGURES

List of tables ..... Pages

List of figures ..... Pages

Abbreviation and Nomenclature

MRI	Magnetic Resonance Imaging
MR	Magnetic Resonance
OA	Osteoarthritis
BSSM	Bi-directional Scanning Segmentation Method
NNC	Neural Network Classifier
NNCM	Neural Network Classifier Method
ACMM	Active Contour Models Method
MLP	Multilayer Perceptron
RF	Radio Frequency
FID	Free Induction Decay
BFGS	Broyden, Fletcher, Goldfard, and Shanno Algorithm
ms	milli-second
mm	milli-meter
SCG	Scaled Conjugate Gradient Algorithm
OSS	One Step Secant Algorithm
CPU	Central Processing Unit
GHz	Giga Hertz
MB	Mega Byte
RAM	Random Access Memory
MSE	Mean Square Error

Chapter 1:

INTRODUCTION

1.1 Motivation

Magnetic resonance imaging (MRI) is a non-invasive method for producing three-dimensional tomographic images of the human body. MRI is the most often used for detection of tumours, lesions, and other abnormalities in soft tissues, such as articular cartilage in human knee.

Recently, computer-aided techniques for analysing and visualizing magnetic resonance (MR) images have been investigated. Many researchers have focused on invivo morphometry and functional analysis of human articular cartilage with quantitative MRI - from image to data, and from data to theory. Measuring the articular cartilage volume from MR images of the knee is one aspect of "image-to-data" process.

This thesis presents three automatic methods that have been accepted to publish on International Workshop on Advanced Computational Intelligent (IWACI) 2010 [1] to extract the articular cartilage from a MR image for measuring its area and volume. They are the bi-directional scanning segmentations method (BSSM), the neural network classifier method (NNCM), and the active contour models method (ACMM).

1.2 Background

Osteoarthritis (OA) is the most prevalent chronic disease in the elderly, affecting more than 50% of those 65 years and older (Peyron 1986 [2], Felson 1988 [3] 1990 [4]; Felson et al. 1995 [5]). It causes pain and functional deficits, with substantial effects on the quality of life (Guccione et al. 1994 [6]). Cartilage loss in knee is one of important elements for detecting OA. For this reason, MRI is used to detect and track the volume of the cartilage in the knee [7]. Identifying the articular cartilage is an important step in calculating its volume.

Numerous methods have been investigated and applied to determine the cartilage in the knee. MRI provides insufficient contrast for fully automated segmentation of articular cartilage based on the gray value distribution alone. For this reason, volume-growing algorithms (Eckstein et al. 1996 [8]; Piplani et al. 1996 [9]) are sensitive to irregularities at the cartilage surface and often fail in regions where contrast is low. Therefore, a B-spline Snake (deformable contour) algorithm is developed that replies on the interaction of "image forces" (gray value gradient), 'model forces' (stiffness of a parameterized B-spline curve), and 'coupling forces' (segmentation of previous section). This approach can accelerate the interactive segmentation process and increases consistency between observers (Stammberger et al. 1999 [10]). Other groups have employed different algorithms, such as "active shape models" (Solloway et al. 1997 [11]), edge detection (Robson et al. 1995 [12]; Kshirsagar et al. 1998 [13]), fitting of B-spline curves to manually digitized points (Ghosh et al. 2000 [14]), and "live-wire" algorithm (Steines et al. 2000 [15]). However, none of these approaches has succeeded in fully automated segmentation of articular cartilage.

1.3 Objectives and Organization of Thesis

The primary goal of this research is to study, and develop automatic methods for extracting articular cartilage of knee from MR images. Three automatic methods have been investigated, including BSSM which is the combination and adaptation of classical image segmentations (using edge detection, thresholding, curve fitting, and average weight calculating algorithms) (detail in Chapter 3); NNCM which is an application of artificial neural network (detail in Chapter 4) and the innovative method, ACMM which is the combination among active contour models algorithms, NNCM and BSSM (detail in Chapter 5).

Based on the successful segmentation using the developed methods, a further goal can be achieved to calculate the articular cartilage area and volume (detail in Chapter 6), that are important properties of articular cartilage in detecting the osteoarthritic(OA).

Chapter 2:

MRI AND DATA CHARACTERISTICS

2.1 Overview

Three automatic methods that we use in cartilage extraction on a MR knee image presented in this thesis capitalize on several properties of magnetic resonance images. Therefore, some knowledge of magnetic resonance imaging (MRI) and the data produced by MRI scanners is necessary.

This chapter introduces the background of MRI and describes some important characteristics of MR knee image.

2.2 Background of MRI

MRI uses a strong magnetic field and high radio frequency (RF) for obtaining sectional images. The technique has so far been shown to have no adverse effects on health. Other important advantages are its multiplanar capabilities and its superior soft tissue contrast (Peterfy and Genant 1996[16]; Stabler et al 2000[17], Peterfy 2000[18]). In MRI, the tissue contrast can be substantially modulated by choosing different types of pulse sequences, and by changing the specific parameters of these sequences (repetition time, echo time, flip angle, etc.). Therefore, a variety of specific sequences can be selected for optimal delineation of specific tissues, or even for specific aspects of these tissues.

The pixel intensity of a given tissue type depends on the proton density of the tissue; the higher the proton density, the stronger the free-induction decay (FID) response signal. MR image contrast also depends on two other specific parameters:

1. The longitudinal relaxation time, T1, and
2. The transverse relaxation time, T2

T1 measures the time required for the magnetic moment of the displaced nuclei to return to equilibrium. T2 indicates the time required for the FID response signal from a given tissue type to decay.

For the analysis of cartilage macro-morphology (volume, thickness, and surface areas), the bone cartilage interface and the articular surface need to be delineated accurately. In particular, the spatial resolution must be sufficient to permit quantitative measurements throughout its thickness. For these reasons, a high-resolution pulse sequence is required that visualizes cartilage with high contrast to its surrounding tissues. Some investigators have used two different pulse sequences and digital subtraction techniques to improve contrast (Robson et al 1995[19]; Munsterer et al 1996[20]). However, today it is widely accepted that T1-weighted gradient echo sequences with spectral fat suppression are best suited for this purpose (Recht et al 1993[21]). These sequences produce images in which the cartilage appears bright (hyper-intense) compared to all other tissues.

2.3 Characteristics of MR Knee Images

Images that are used in this thesis were MRI scans of the right knees. The following image sequence was used: T1-weighted fat saturation magnetic resonance imaging; repetition time 43ms; echo time 13ms; flip angle 50^{o}; 62 partitions; 512x512 matrix; greyscale formation. Sagittal images were obtained at a partition thickness of 1.5mm and an in-plane resolution of 0.31x0.31mm (512x512 pixels). Fig 2.1 shows an example of a sagittal image.

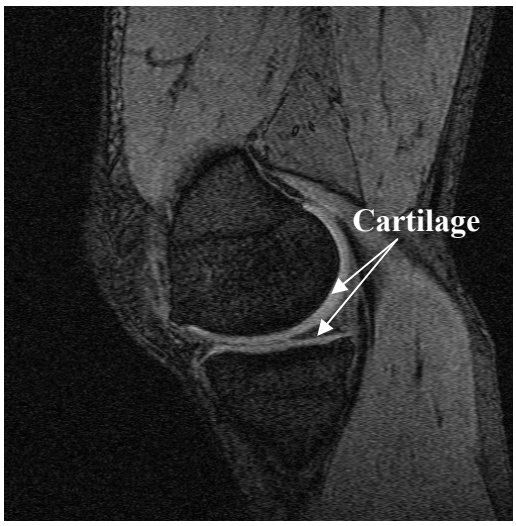


Figure 2.1 Sagittal T1-weighted fat saturation MR image

In order to measure the articular cartilage's area on a sagittal image, we need to extract a cartilage from original image. For doing this, we can use BSSM (Chapter 3), NNCM (Chapter 4), and ACMM (Chapter 5). The volume of a cartilage can be then computed from 62 sequenced sagittal images.

Fig 2.1 shows that cartilage is generally located in the middle region of the image. All the images used in our research had the cartilage located in the middle region. From the image sequence, the first and the last 13 images of the sequence did not reveal the cartilage on those images. This is because MR scanning is performed from the medial side of the knee, over the patella and ends on the lateral side. Fig 2.2 demonstrates MR scanning direction. The first and the last 13 images from the image sequence was obtained heuristically. This assumption was used in a consistent manner for the experiments.

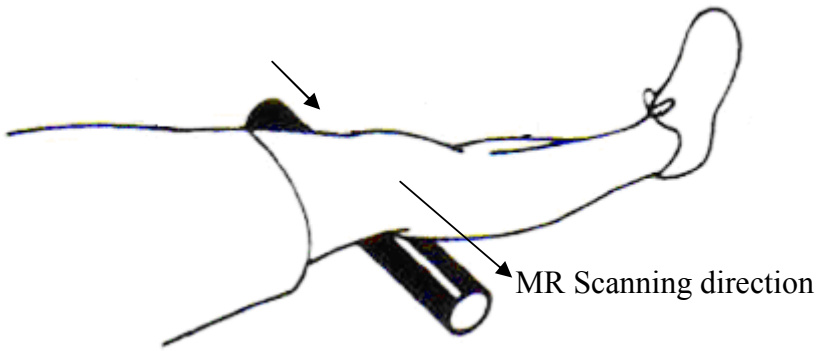
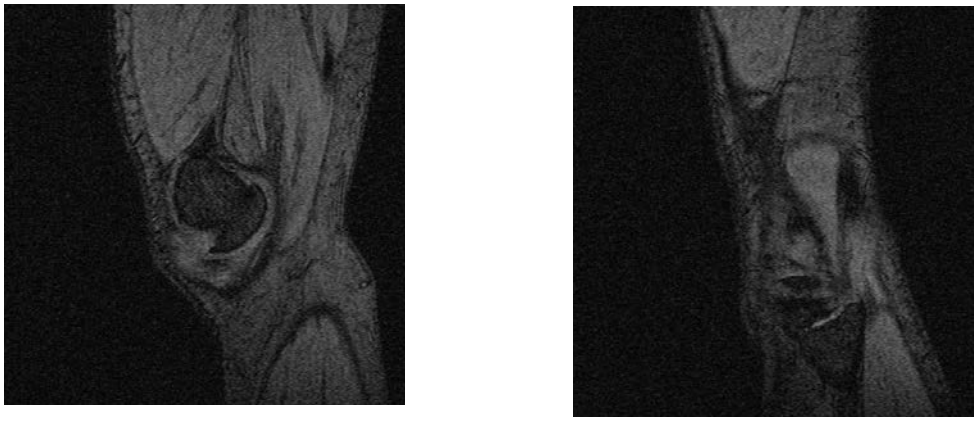


Figure 2.2 MR scanning direction

Fig 2.3 illustrates the disappearance of cartilage on a MR image. Fig 2.3 (a) is a sagittal image on the first 13 image sequence while Fig 2.3 (b) is a sagittal image on the last 13 image sequence.



(a) (b)

Figure 2.3 MR images in that articular cartilage did not appear

(a) A MR image in the first 13 image sequence.

(b) A MR image in the last 13 image sequence.

In study for the easy readability, the cartilage is divided in three components namely femur which is the cartilage is attached to femur, tibia which is the cartilage is attached to tibia, and patella which is the cartilage is attached to patella (Fig 2.4). Depend on image sequence and MR scanning direction, a sagittal image may contain one, two, or all of cartilage components. Therefore, we have three typical types of image according to cartilage components:

1. Images that contain Femur and Tibia
2. Images that contain Femur, Tibia, and Patella
3. Images that contain Femur and Patella

Following table illustrates distribution of image types according to image sequence.

Image Sequence	**Cartilage types**
1 - 13	Do not appear
14-18	Femur and Tibia
19 - 21	Femur , Tibia, and Patella
22 - 37	Femur and Patella
38 - 45	Femur, Tibia, and Patella
46 - 49	Femur, and Tibia
50 - 62	Do not appear

Table 2.1 Image types according to image sequence

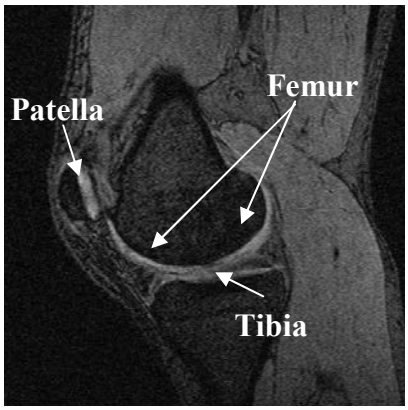


Figure 2.4 A MR image that contains three types of cartilage

In term of computer vision, a MR image, which is formatted in grayscale [22], is a 512x512 matrix. Each matrix's element (or call pixel) have intensity value in range [0, 255]. Because cartilage appears bright colour compared to all other tissues (background), cartilage pixels have high intensity whereas background pixels have lower intensity. Fig 2.5 is an example of this. Fig 2.5 (a) is a sub-image on an image that contains cartilage segment. Fig 2.5 (b) is a gray value profile according to a sub-image. Cartilage pixels have high intensity (greater than 80) compared to background pixels intensity (smaller than 50).

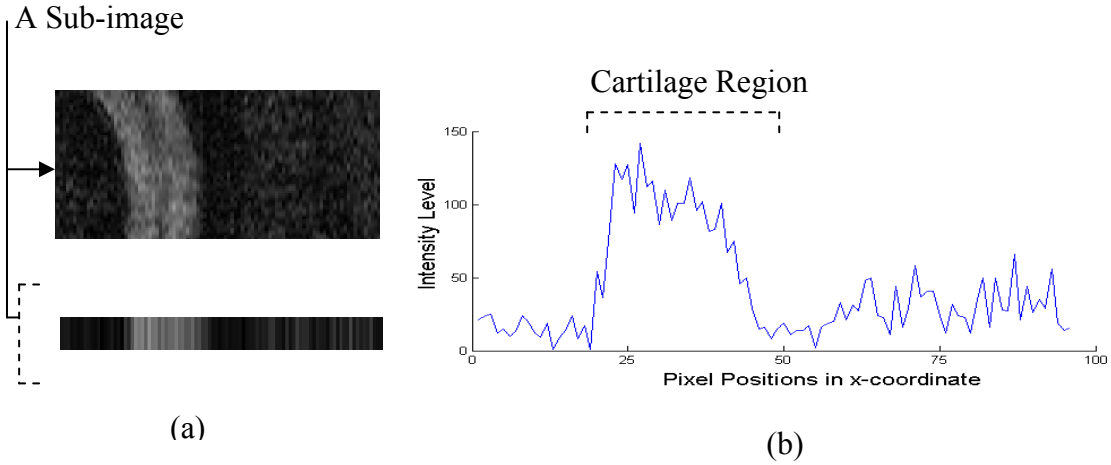


Figure 2.5 A sub-image (a) and its gray-value profile (b).

2.4 Conclusion

This chapter introduced the background of magnetic resonance imaging and illustrated several characteristics of MR knee images. Base on this, we developed three automatic methods for extracting a cartilage from an original image.

Next chapter, we present the first method namely bi-directional scanning segmentation.

Chapter 3:

BI-DIRECTIONAL SCANNING SEGEMENTATIONS

3.1 Overview

Image segmentation is an essential preliminary step in most image pattern (object) recognition process. It subdivides an image into its constituent regions or objects. That is, segmentation should stop when the objects of interest in an application have been isolated.

Generally, image segmentation algorithms are based on one of two basic properties of intensity values: discontinuity and similarity. In the first category, the approach is to partition an image based on abrupt changes in intensity, such as edges in an image. The principal approaches in the second category are based on partitioning an image into regions that are similar according to a set of predefined criteria. Thresholding is an example of methods in this category that is to extract the objects from the background is to select a threshold T that separates object pixels and background pixels.

However, according to complexity in MR knee images that contain unwanted intensity variations (noises), none of individual segmentation algorithms is satisfactory in extracting an articular cartilage from an original image. Therefore, we developed a method that combines various image segmentation algorithms to improve the result. This method is named bi-directional scanning segmentations method (BSSM) [1] and will be presented in this chapter.

3.2 Pre-processing

In order to reduce the noise and other irrelevant parts, a pre-processing step is required at the beginning. Pre-processing's goal is to find an interest region that contains an articular cartilage on an original image. This interest region is used as the input image for BSSM. Furthermore, NNCM (chapter 4) and ACMM (chapter 5) also use it as input image.

An interest region can be defined as a function ***R*** of the form:

$$R(x,y) = \mathrm{R}\,[x, y, f(x,y), g(x,y), p_1(x,y), p_2(x,y)\ldots]$$

where $\boldsymbol{R}(x,y)$ is the interest region an image $f(x,y)$ according to x and y coordinates. $g(x,y)$ denotes feature of a cartilage on this image. p_1, p_2… indicate some other properties relating to cartilage that helps for defining $\boldsymbol{R}(x,y)$.

Figure 3.1 Examples of interest regions according to different images.

Fig 3.1 illustrates interest regions that are defined on different images. The sizes of interested regions are also different.

3.3 Bi-directional Scanning Segmentations

Due to as the complexity of MR knee images, segmentation on local regions (sub-images) of the image is much more effective than segmentation on an entire image. Therefore, our segmentations algorithms are applied to partitioned sub-images to extract cartilage segments. In general, our BSSM algorithms include two processes: left and right process. The left process is to scan the image from an initial position to its left direction and extract the cartilage on partitioned sub-images during the scan. Similarly, right process is to extract the cartilage on partitioned sub-images to the right direction (Fig 3.2(a)).

Consider an input image as a matrix ***I*** that is defined from pre-processing step. Sub-images are columns of matrix ***I.*** Sub-images that contain the cartilage segments are defined as cartilage sub-images. BSSM algorithms are described as:

1. From input image [K rows and L columns] ***I***, find initial cartilage sub-image representing $i^{\text{-th}}$ column of matrix ***I***, where $i = 1, 2, \ldots, L$. Initial cartilage sub-image determination is described in section 3.3.1.

- Apply boundary detection method to determine cartilage boundaries and extract a cartilage segment from an initial sub-image. Boundary detection method is described in section 3.3.2.

2. Left Process: Staring from the initial sub-image, scan to the next cartilage sub-image on the left representing $(i + \Delta)^{-th}$ column, $\Delta = 1, 2, 3\ldots$

- Apply boundary detection method to determine cartilage boundaries and extract a cartilage segment from a sub-image.

3. Right Process: Starting from the initial sub-image, scan to the previous cartilage sub-image representing $(i - \Delta)^{-th}$ column, $\Delta = 1, 2, 3\ldots$

- Apply boundary detection method to determine cartilage boundaries and extract a cartilage segment from a sub-image.

4. Continue apply boundary detection method until all cartilage sub-image on an input image are processed.

BSSM is used to isolate a cartilage from its background. We can then measure the size or area of a cartilage. The success of this operation depends on determining cartilage boundaries on partitioned sub-images. We then develop a method named boundary detection for detecting cartilage boundaries. It is the most important part in BSSM.

According to three different components of a cartilage namely Femur, Tibia, and Patella, BSSM aims to isolate individual cartilage component from its background and other cartilage components. Therefore, boundary detection also tries to detect individual cartilage component's boundaries.

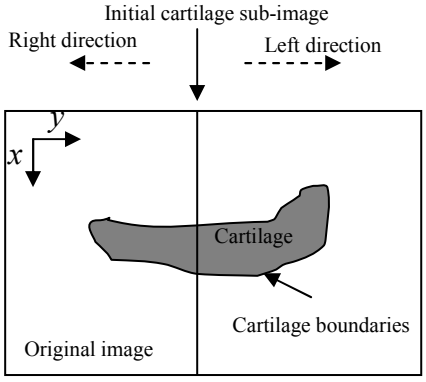


Figure 3.2(a) Right and left direction according to initial sub-image on an input image

Fig 3.2(b) shows flowchart of BSSM algorithms.

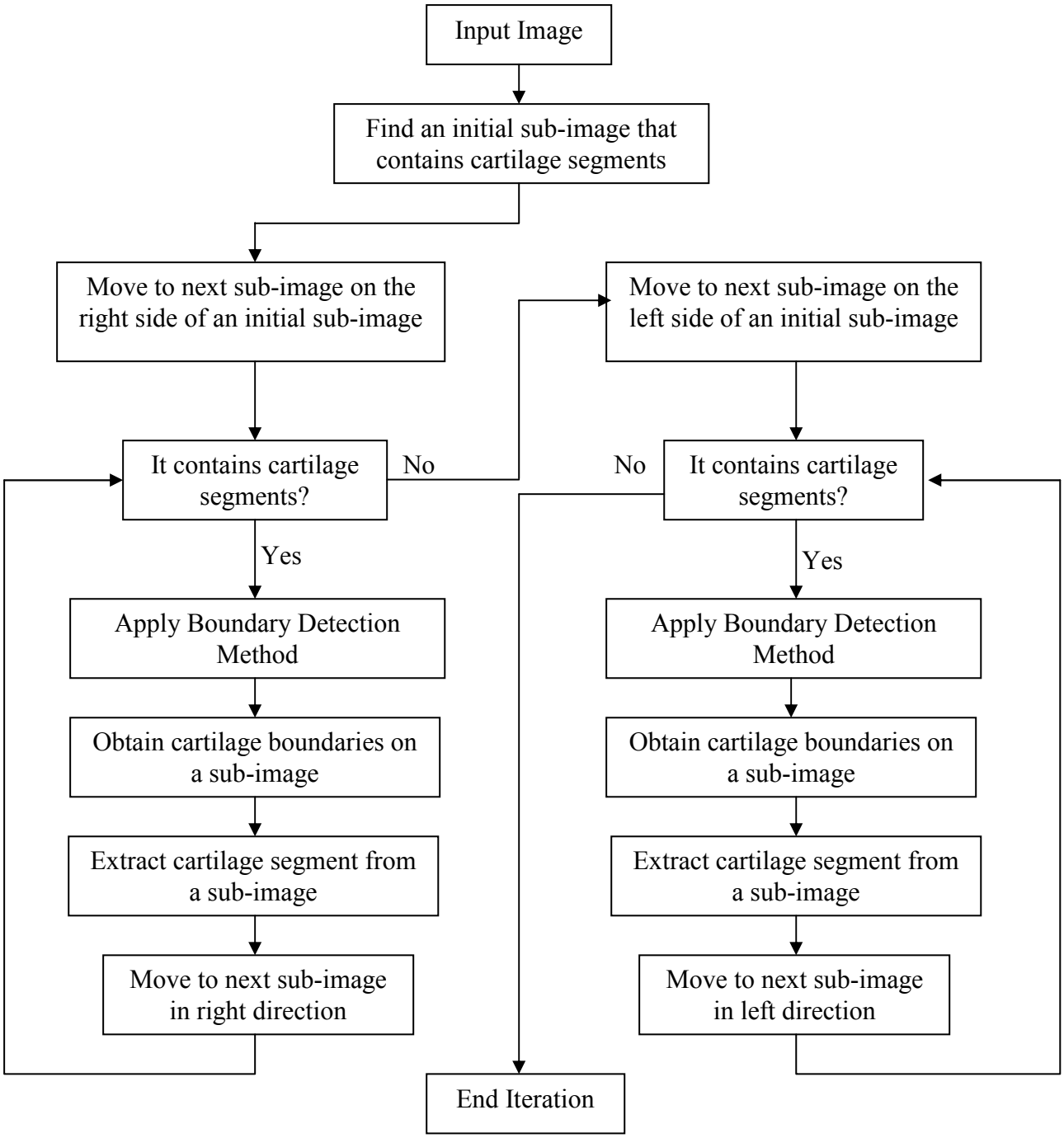
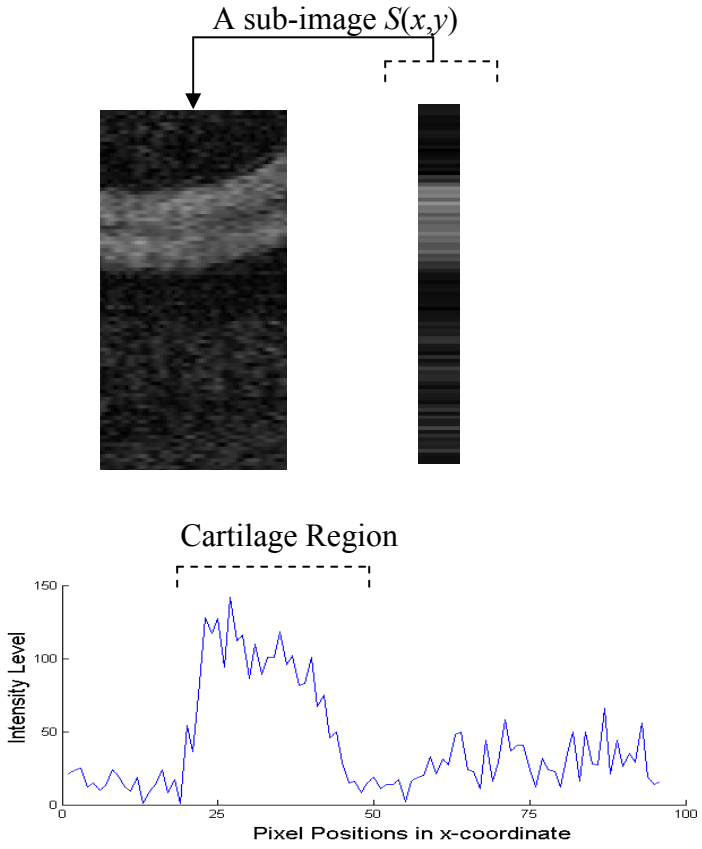


Figure 3.2(b) Operation of BSSM.

3.3.1 Cartilage and Initial Cartilage Sub-image Determination

In an MR knee image, the articular cartilage will appear in high intensity whereas the other tissues (background) appear in low intensity. Consider a sub-image $S(x,y)$ that contains a cartilage segment show in Fig 3.3 (a) and gray-level profile of $S(x,y)$ as we traverse along a vertical line. (Fig 3.3 (b)).



(b)

Figure 3.3 A sub-image (a) and its gray-value profile (b)

From the gray-level, cartilage pixels have high intensity in the range [80, 150] whereas background pixels have very low value in range [0, 50]. Thus, a cartilage sub-image is determined if there are a group of high intensity pixels with mean value is greater than a threshold value V.

A sub-image $S(x,y)$ is defined
- Cartilage sub-image if $g(x,y) >= V$
- Background sub-image if $g(x,y) < V$

where $g(x,y)$ is the mean value of a group of high intensity pixels in sub-image $S(x,y)$.

An initial cartilage sub-image is a cartilage sub-image, which locates in the middle of a cartilage region as well as in the middle of an input image. Consider S_1, S_2,..S_k, ..S_N (k = 1, 2, 3…, N where N is number of cartilage sub-image) are cartilage sub-image in the middle region of an input image. An initial cartilage is defined as:

An sub-image $S_k(x,y)$ is initial sub-image if $g_k(x,y)$ is maximum.

Where $g_k(x,y)$ is the mean value of a group of high intensity pixels in sub-image $S_k(x,y)$.

3.3.2 Boundary Detection

Boundary detection method is used to detect the cartilage segments boundaries on sub-images. Therefore, cartilage segments can be extracted from a sub-image (its background and other cartilage components). Fig 3.4 generally illustrates the goal of this method.

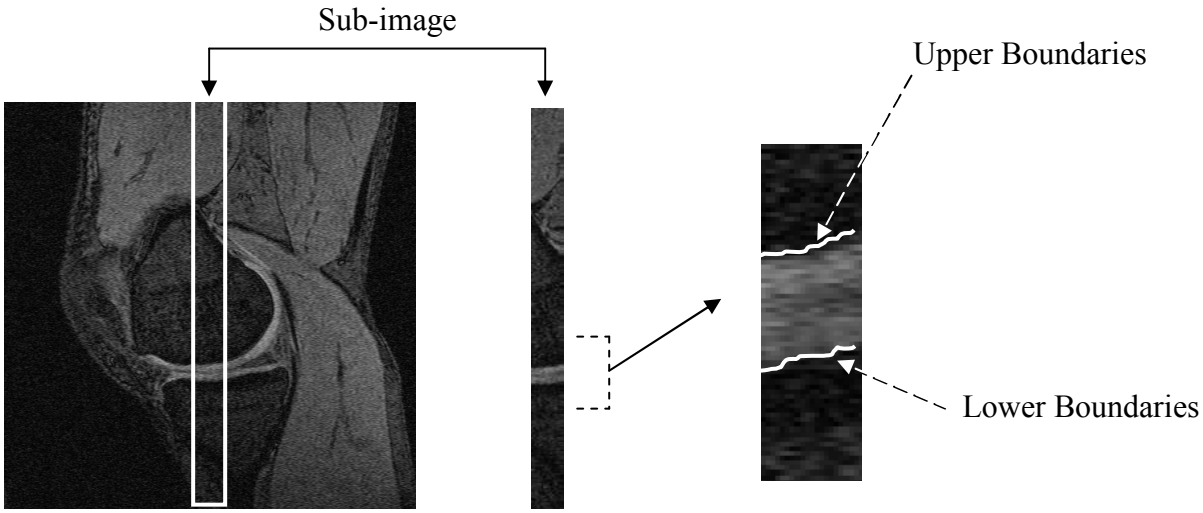


Figure 3.4 Cartilage segment boundaries found on sub-image by using boundary detection.

We consider two types of a cartilage boundary on a sub-image defined as upper and lower boundaries (Fig 3.4). Original boundary detection algorithms are based on edge detection algorithms (detail in section 3.3.2.1) to find cartilage boundaries on a sub-image. However, because the limitation of edge detection, we develop boundary detection method with thresholding method (detail in section 3.3.2.2) and statistical analysis algorithms (detail in section 3.3.2.3).

Consider a cartilage sub-image $S(x,y)$, boundary detection algorithms to find upper and lower boundaries of a cartilage are described as:

1. Apply Edge detection on sub-image $S(x,y)$, we obtain new image $S_{edge}(x,y)$ where:

   Pixel value at location (x,y) is 1 representing edge are found.

   Pixel value at location (x,y) is 0 representing edge are not found.

2. Apply Thresholding on sub-image $S(x,y)$, we obtain new image $S_{threshold}(x,y)$ where:

   Pixel value at location (x,y) is 1 representing cartilage pixel.

   Pixel value at location (x,y) is 0 representing background pixel.

3. Find the highest intensity pixel at location (x_0,y) on $S(x,y)$.
4. For next pixel at location $x_1 = x_0 + \Delta$ where $\Delta = 1, 2, 3 \ldots$

   If Pixel value at location (x_1,y) on S_{edge} and $S_{threshold}$ is 1, upper boundary is found, denoted by B_{up}.

5. For previous pixel at location $x_2 = x_0 - \Delta$ where $\Delta = 1, 2, 3 \ldots$

   If Pixel value at location (x_2,y) on S_{edge} and $S_{threshold}$ is 1, lower boundary is found, denoted by B_{low}.

6. Repeat step 4 and 5 until upper and lower boundaries are found.
7. Apply statistical analysis to obtain approximations of upper and lower boundaries on $S(x,y)$ that is denoted by A_{up} and A_{low} respectively.
8. Compute the error:

$$\delta_1 = \left| B_{up} - A_{up} \right|$$
$$\delta_2 = \left| B_{low} - A_{low} \right|$$

If $\delta_1 < \alpha$, upper boundary is B_{up}, otherwise upper boundary is A_{up}.

If $\delta_2 < \beta$, lower boundary is B_{low}, otherwise lower boundary is A_{low}.

Where parameters α, and β is relative constant of error terms.

Fig 3.5 illustrates flowchart of boundary detection algorithms:

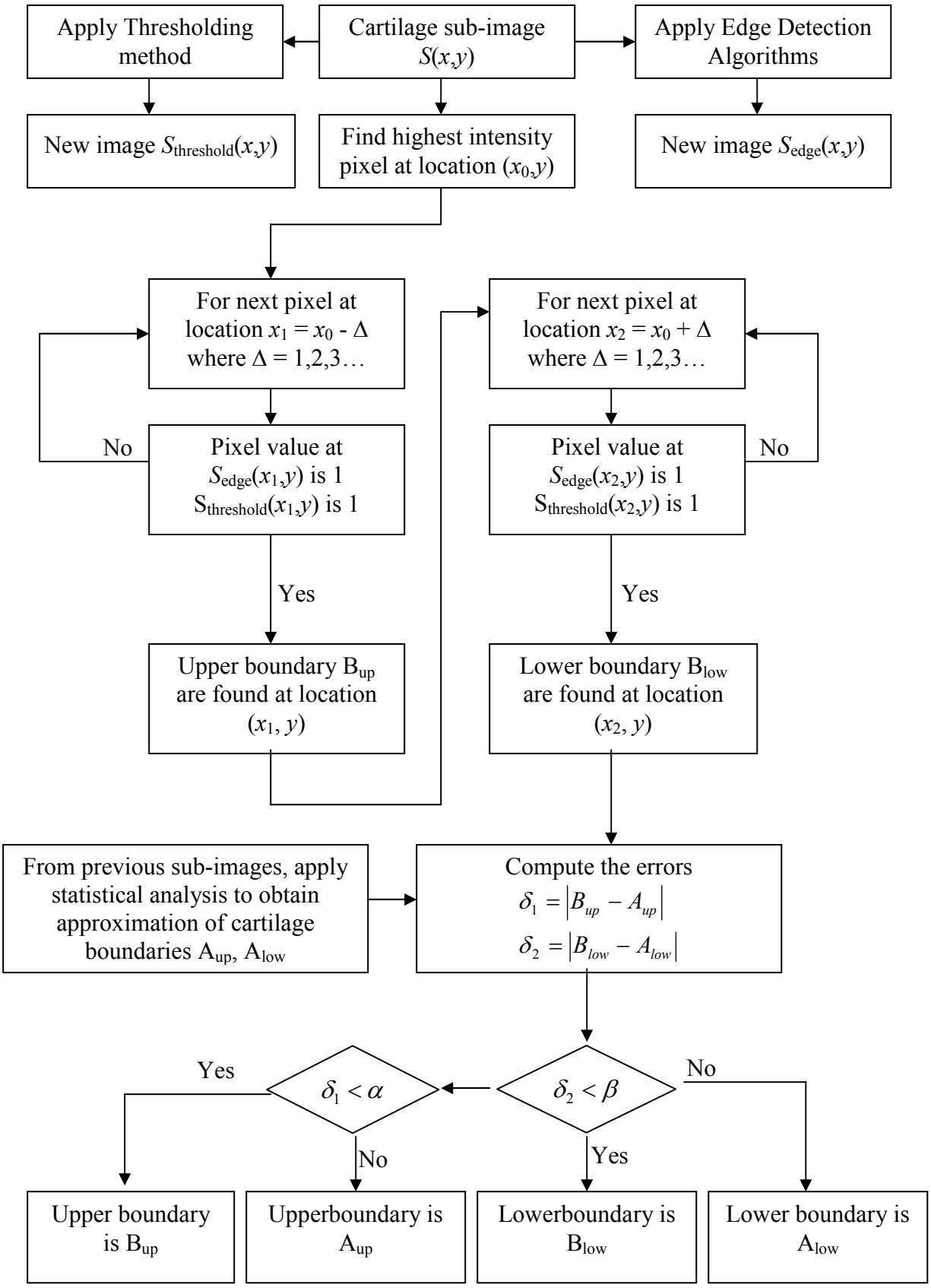
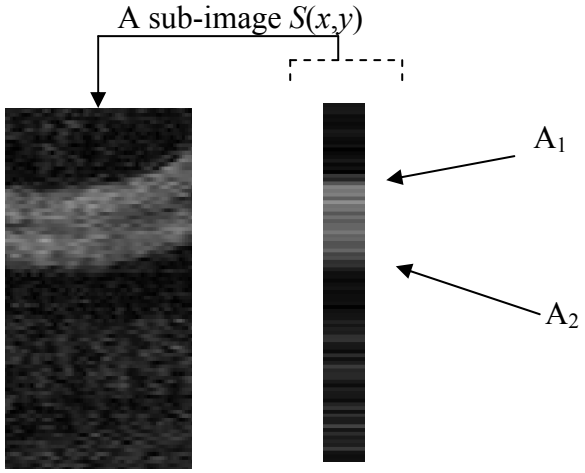


Figure 3.5 Operation of boundary detection.

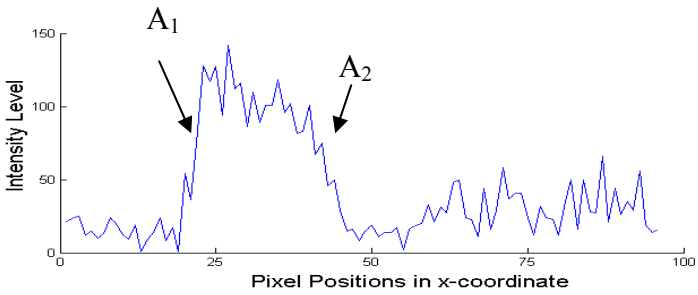
3.3.2.1 Edge Detection

Edge detection is one of image segmentation methods, which is based on the discontinuity of intensity values. It aims to partition an image by investigating the intensity changes.

Consider the sub-image $S(x,y)$ (representing a column of matrix $\boldsymbol{S}$) in Fig 3.6 (a), which is composed of light cartilage on dark background. Suppose we plot the gray values as we traverse the image along a vertical line (Figure 3.6 (b)). There are changes in intensity values in individual region and in between them. Analysing the contrast between high intensity pixels and low intensity pixels, the boundary of a cartilage is found when there is a significant intensity change. To investigate those changes, we take the derivative of pixels in term of intensity. Thus, edges that are considered as cartilage boundaries can be calculated. Intuitively, A_1, A_2 represent actual upper and lower boundaries of a cartilage segment on a sub-image. Edges are expected to close to A_1, A_2.



(a)



(b)

Figure 3.6 (a) A sub-image $S(x,y)$ and its gray-value profile (b).

We consider two types of derivatives: first-order and second-order derivatives. There are many edge-finding algorithms based on that.

First-order Derivative

First-order derivatives of an image are defined as various approximations of the 2-D gradient. The gradient of an image *f(x,y)* at location *(x,y)* is defined as the vector:

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

It is well known from vector analysis that the gradient vector points in the direction of maximum rate of change of f at coordinates (x,y). This gives two important properties: the magnitude and the direction of this vector.

The magnitude of gradient vector is defined as:

$$\nabla f = mag(\nabla f) = \left[G_x^2 + G_y^2\right]^{\frac{1}{2}}$$

This quantity gives the maximum rate of increase of *f(x,y)* per unit distance in the direction of gradient vector.

The direction of gradient vector at point *(x,y)* is represented by $\alpha(x,y)$, defined as:

$$\alpha(x,y) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

There are several ways to approximate G_x and G_y. Let the 3x3 area shown in Fig 3.7 presents the intensity values in a neighbourhood at point P*(x,y)*. The following list some:

$P_{(x-1,y-1)}$	$P_{(x-1,y)}$	$P_{(x-1,y+1)}$
$P_{(x,y-1)}$	$P_{(x,y)}$	$P_{(x,y+1)}$
$P_{(x+1,y-1)}$	$P_{(x+1,y)}$	$P_{(x+1,y+1)}$

Fig 3.7 A 3x3 region of a point *(x,y)* in an image.

Robert cross-gradient method [23]:

$$G_x = P_{(x+1,y+1)} - P_{(x,y)}$$
$$G_y = P_{(x+1,y)} - P_{(x,y+1)}$$

Prewitt method [23]:

$$G_x = \left(P_{(x+1,y-1)} + P_{(x+1,y)} + P_{(x+1,y+1)}\right) - \left(P_{(x-1,y-1)} + P_{(x-1,y)} + P_{(x-1,y+1)}\right)$$
$$G_y = \left(P_{(x-1,y+1)} + P_{(x,y+1)} + P_{(x+1,y+1)}\right) - \left(P_{(x-1,y-1)} + P_{(x,y-1)} + P_{(x+1,y-1)}\right)$$

Sobel method [23]:

$$G_x = \left(P_{(x+1,y-1)} + 2P_{(x+1,y)} + P_{(x+1,y+1)}\right) - \left(P_{(x-1,y-1)} + 2P_{(x-1,y)} + P_{(x-1,y+1)}\right)$$
$$G_y = \left(P_{(x-1,y+1)} + 2P_{(x,y+1)} + P_{(x+1,y+1)}\right) - \left(P_{(x-1,y-1)} + 2P_{(x,y-1)} + P_{(x+1,y-1)}\right)$$

Therefore, edges are found at those points where the gradient is maximum.

Second-order Derivative

Laplacian [24] of a 2-D function *f(x,y)* is a second-order derivative defined as:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

For a 3x3 region (see Fig 3.7), $\nabla^2 f$ can be approximated as:

$$\nabla^2 f = 8P_{(x,y)} - \left(P_{(x-1,y-1)} + P_{(x-1,y)} + P_{(x-1,y+1)} + P_{(x,y-1)} + P_{(x,y+1)} + P_{(x+1,y-1)} + P_{(x+1,y)} + P_{(x+1,y+1)}\right)$$

The Laplacian is combined with smoothing as a pre-processing to finding edges via zero-crossing [24]. Consider the function:

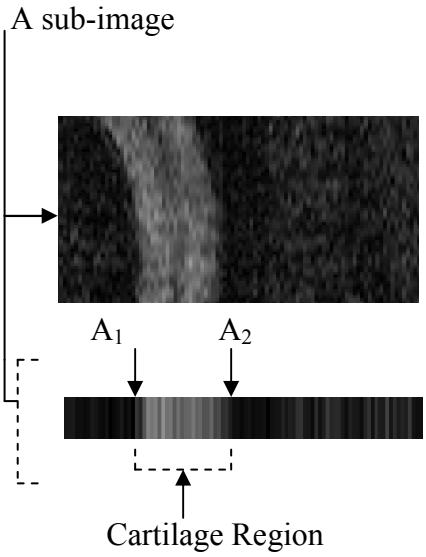
$$h(r) = -e^{-\frac{r^2}{2\sigma^2}}$$

Where $r^2 = x^2 + y^2$ and σ is the standard deviation. The Laplacian of *h* (the second-order derivative of h with respect to r) is:

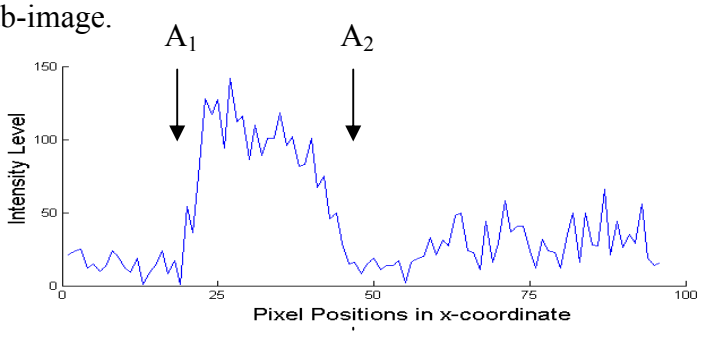
$$\nabla^2 h(r) = -\left[\frac{r^2 - \sigma^2}{\sigma^4}\right] e^{-\frac{r^2}{2\sigma^2}}$$

Discussion on using different edge detections algorithms as boundary detection

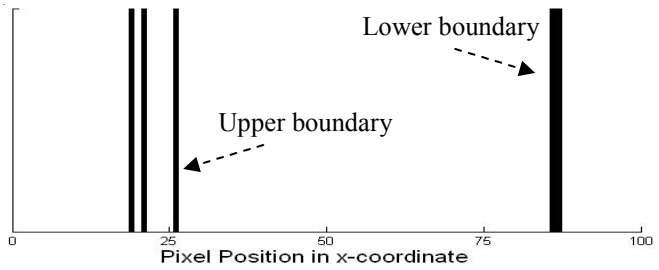
Fig 3.8 demonstrates an example of applying different edge detection techniques to find edges corresponding to a sub-image.



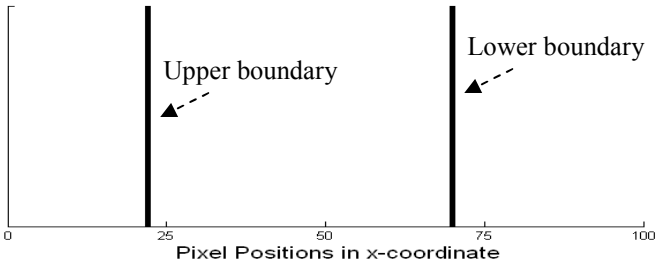
(a)



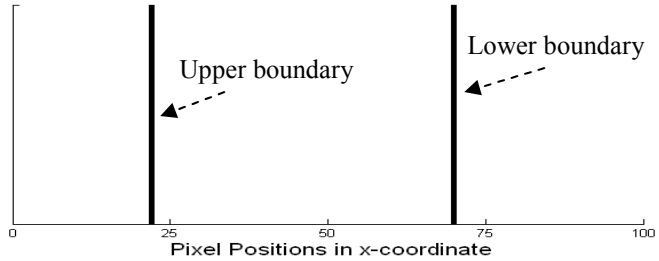
(b)



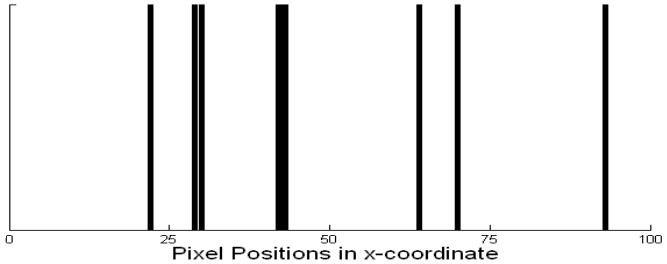
(c)



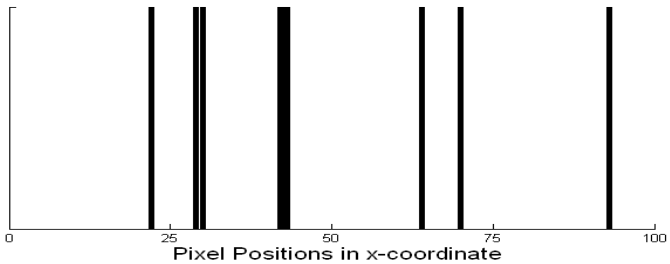
(d)



(e)



(f)



(g)

Figure 3.8
(a) An example sub-image
(b) Gray-level profile corresponds to a sub-image.
Edges found by using Robert method (c), Prewitt method (d), Sobel method (e), Laplacian method (f), and zero-crossing method (g) according to the profile.

Fig 3.8 (a) show a sub-image $S(x,y)$. This sub-image is the same shown in Fig 3.6 (a). In Fig 3.8 (a), we present sub-image $S(x,y)$ in horizontal direction as the results obtained by using various edge detection algorithms. It is compatible with its gray-level profile (Fig 3.8 (b)). A_1, A_2 represent actual upper and lower boundaries of a cartilage segment on a sub-image, respectively.

The Laplacian and zero-crossing methods fail in detecting the cartilage boundaries (Fig 3.8 (f) and (g)). It produces more edges that cause confusion to determine which edges are the cartilage boundaries. The Robert method can detect the lower cartilage boundary (Fig 3.8(c)). However, the difference between actual lower boundary and lower boundary found by Robert method is quite large and unacceptable. Furthermore, Robert method also produces multiple edges that cause complication to detect the upper cartilage boundary.

Sobel and Prewitt methods (Fig 3.8 (d) and (e)) provide the results that are more accurate compared to others. They can detect both upper and lower boundaries of the cartilage. The difference between the edges and actual boundaries is small and acceptable. Prewitt method is simpler to implement than Sobel method, but the later have slightly superior noise-suppression characteristic. Therefore, we apply Sobel algorithms for edge detection. However, since Sobel algorithm is only based on the discontinuity of intensity values, edge detection faces two typical issues:

- The first issue occurs when there is significant intensity change in cartilage region, multiple edges are more likely to be detected. Hence, it causes complication for boundary detection in determining cartilage boundaries.

- The second issue occurs when there is slightly intensity change between cartilage and background (or other cartilage components) regions. Edges cannot be effectively detected or are likely to be detected incorrectly. As a result, boundary detection fails to obtain the correct cartilage boundaries.

Fig 3.9 illustrates the first issue of edge detection. Fig 3.9 (a) is a gray profile of a sub-image. Fig 3.9 (b) is result obtained by using Sobel edge detection method. Because there are significant intensity changes on cartilage region, multiple edges are produced in cartilage region (Fig 3.9 (b)). Those edges also contain two main edges that indicate upper and lower cartilage boundaries. Therefore, boundary detection cannot detect cartilage boundaries on this sub-image.

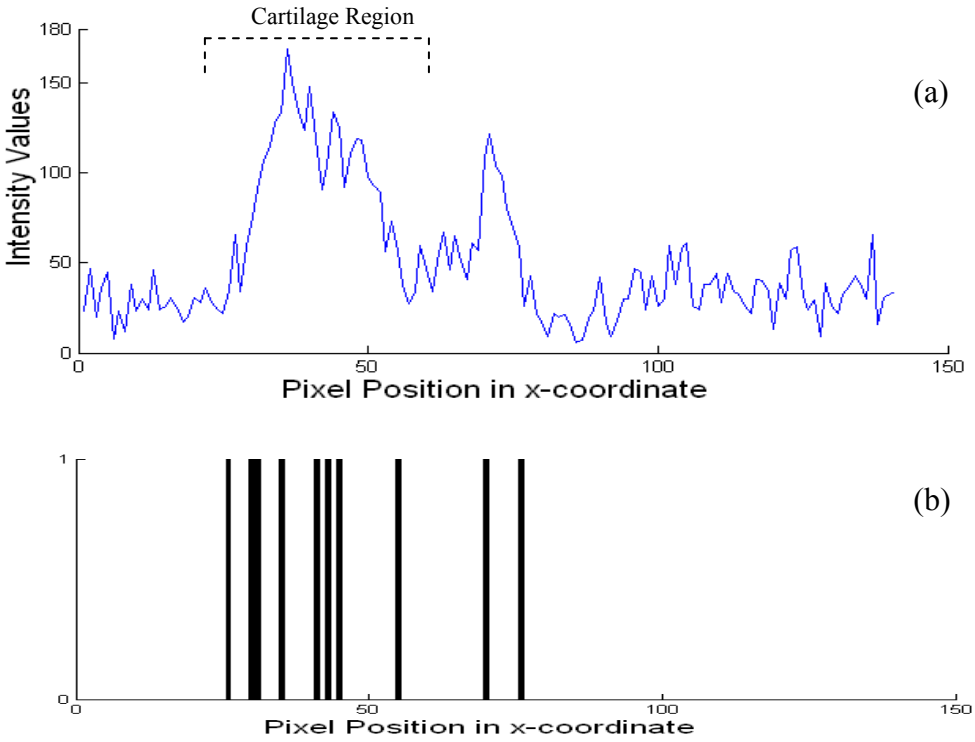


Figure 3.9 (a) The gray-profile of a sub-image
(b) Edges found by using Sobel edge detection method

Fig 3.10 demonstrates second issue of edge detection. Fig 3.10 (a) presents a gray profile of a sub-image. Fig 3.10 (b) shows the result obtained by using Sobel edge detection method. Since the intensity changes between cartilage and background regions in the left side of cartilage region are slight, the edge cannot be detected. In additional, it is likely to be detected incorrectly where the change is significant. As a result, upper boundary is determined far away from actual upper boundary.

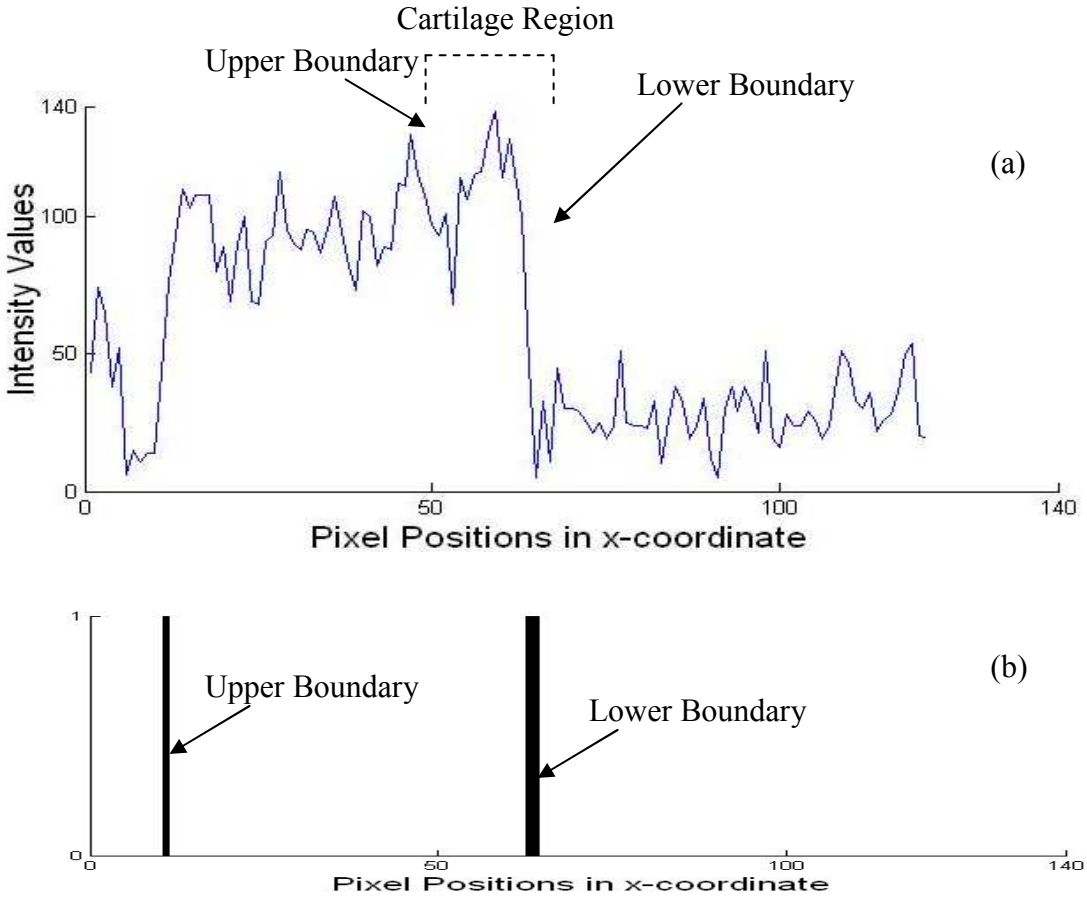


Figure 3.10 (a) The gray-profile of a sub-image

(b) Edges found by using Sobel edge detection method.

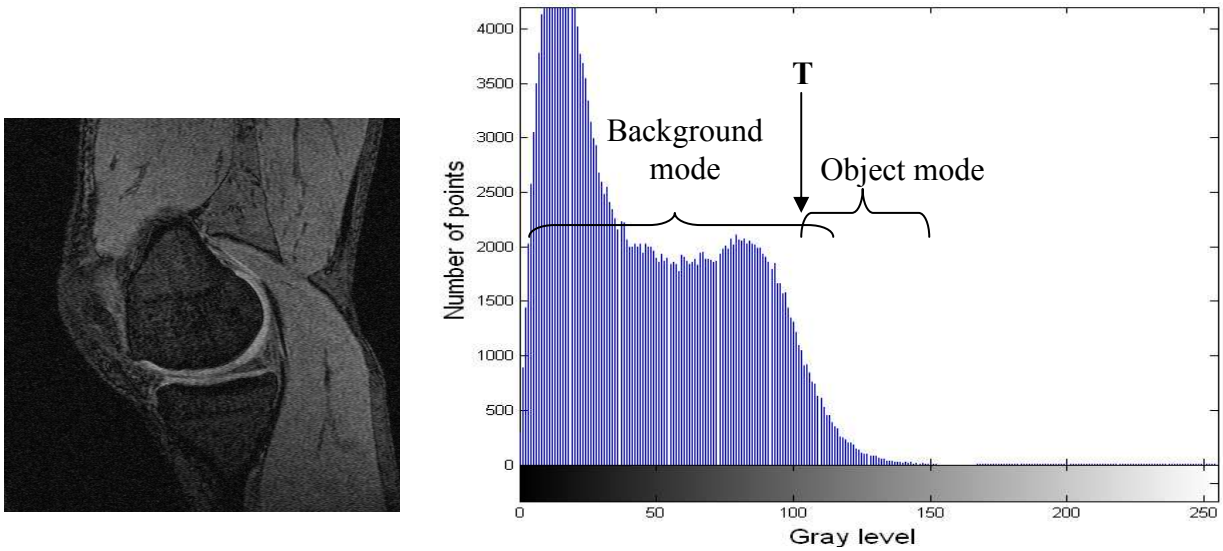
Conclusion

Edges are formed from pixels with derivative values. Intuitively, edges are considered as cartilage boundaries. Boundary detection method is based on Sobel edge detection algorithm is used to determine cartilage boundaries on a sub-image. However, because the limitations of edge detection algorithms mentioned above, we develop boundary detection method by applying thresholding method.

3.3.2.2 Thresholding

Unlikely edge detection, thresholding is based on different properties of intensity values: similarity. The gray-level histogram show in Figure 3.11 (b) corresponding to an MR image is composed of light cartilage on a dark background (other irrelevant tissues), in such a way that object and background pixels have gray levels grouped into two dominant modes. In order to isolate the object from the background, we select a threshold T that separates these modes. Then, each pixel on image is classified as a cartilage pixel or a background pixel according to whether its gray value is greater than or less than threshold value T [25].

$$\text{A pixel is classified} \begin{cases} \text{Cartilage pixel if its gray level is} > T, \\ \text{Background pixel if its gray level is} \leq T. \end{cases}$$



(a) (b)

Figure 3.11 A example of gray level histogram (b) corresponding to an image (a)

However, the trouble is that in general the individual histograms of the objects and background overlap (Figure 3.11 (b)). Hence, it is important in choosing an appropriate threshold level. If we choose a value too high, we may decrease the size of object or reduce its total pixel number. Conversely, if we choose a value too low, we may

include extraneous background material. For this reason, we applied Otsu's method for choosing a best threshold.

Optimal Thresholding Algorithms

Otsu's method (Otsu 1979 [26]) is one of the most popular techniques of finding an optimal threshold. Essentially, Otsu's technique maximises likelihood that the threshold is chosen to split the image between an object and its background. The basis is to use of the normalised histogram where the number of points at each level is divided by the total number of points in the image. As such, this represents a probability distribution for the intensity level as:

$$p_i = \frac{n_i}{N}$$

where n_i is the number of pixels with gray level i, N is the total number of pixels in an image, so that p_i is the probability of a pixel having gray level i. If we threshold at level k, we define:

$$\omega(k) = \sum_{i=0}^{k} p_i$$

$$\mu(k) = \sum_{i=k+1}^{L-1} p_i$$

where L is the number of grayscales, so that L - 1 is the largest. By definition,

$$\omega(k) + \mu(k) = \sum_{i=0}^{L-1} p_i = 1$$

We would like to find k to maximize the difference between $\omega(k)$ and $\mu(k)$. This can be done by first defining the image average intensity as:

$$\mu_T = \sum_{i=0}^{L-1} i.p_i$$

And then finding k, which maximizes:

$$\frac{\left(\mu_T \omega(k) - \mu(k)\right)^2}{\omega(k)\mu(k)}$$

Adaptability of sub-image thresholding

A global optimal threshold can be computed from an entire original image. Due to noise, a cartilage cannot be partitioned effectively by a global threshold. It causes the error between the actual cartilage boundaries and the cartilage boundaries found. For handling such a situation, we need to divide the original image into sub-images and then utilize a different threshold to segment each sub-image. Since the threshold used for each pixel depends on the location of the pixel in terms of the sub-images, this type of thresholding is adaptive.

Fig 3.12 presents the difference between using global and adaptive thresholds in partitioning a cartilage segment on a sub-image. Fig 3.12 (b) is the gray-level profile of a sub-image which is showed in Fig 3.12 (a).

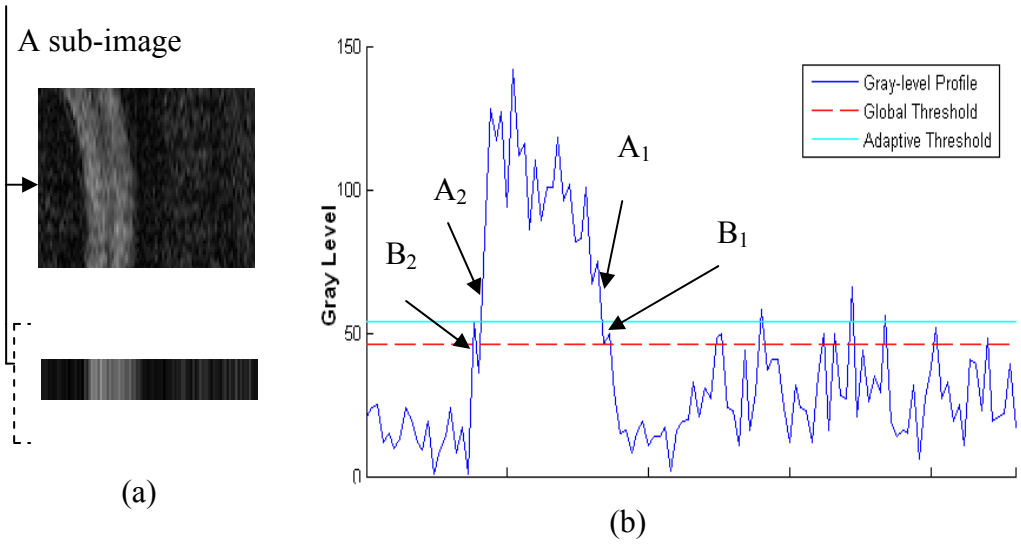


Figure 3.12 An example of using global and adaptive threshold
(a) A sub-image of MR image and (b) Its gray level profile.

From Fig 3.12 (b), A_1, A_2 are lower and upper boundaries of cartilage segment on a sub-image. A_1, A_2 are obtained by using adaptive threshold. Similarly, B_1, B_2 are lower and upper boundaries of a cartilage segment. B_1, B_2 are obtained by using global threshold. A_1, A_2 are closer to actual boundaries of a cartilage compared to B_1, B_2. Therefore, the error between the actual cartilage and the cartilage that is partitioned by adaptive thresholding method is reduced in terms of size and area.

Discussion on boundary detection using Thresholding method

Fig 3.13 illustrates the behaviour of thresholding method corresponding to the first issue of edge detection (failure to detect cartilage boundaries where there are significant intensity changes in cartilage region). Fig 3.13 (a) shows the gray-level profile of a sub-image. It is also a gray profile shown in Fig 3.9 (a). Fig 3.13 (b), and (c) are the results obtained by using thresholding method and edge detection, respectively. Generally, pixels that have value 1 are classified as cartilage pixels while pixels that have value 0 are classified as background pixels. In this case, pixels that have value "1" are grouped into two regions. Obviously, the region that has the largest area is considered as cartilage region whereas other is considered as noise. The cartilage boundaries are then defined.

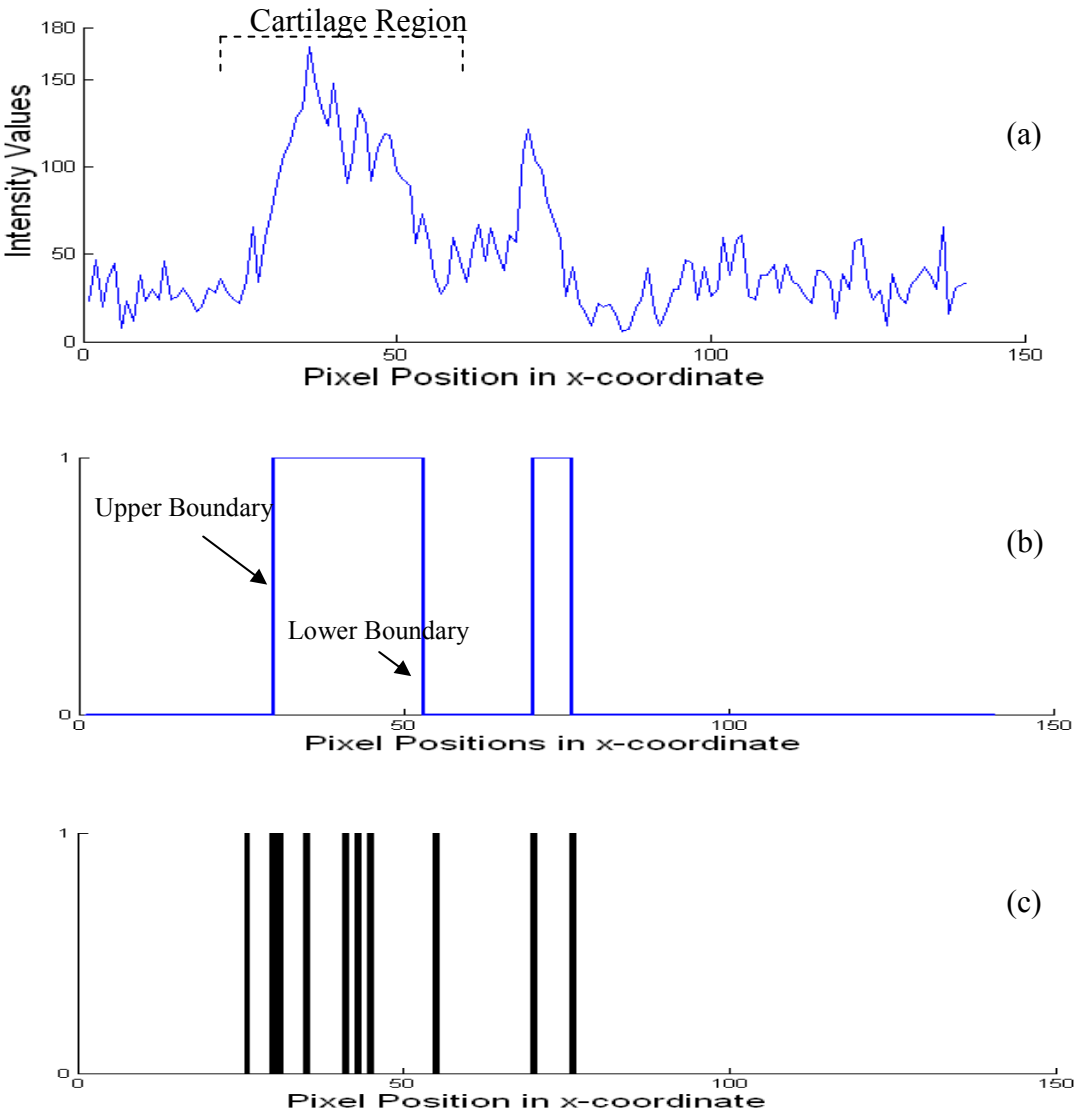


Figure 3.13 (a) Gray profile of a cartilage sub-image

Result obtained by using thresholding method (b) and edge detection (c)

Fig 3.14 demonstrates the behaviour of thresholding method corresponding to the second issue of edge detection (failure to detect cartilage boundaries when there are slight intensity changes between cartilage and background regions). Fig 3.14 (a) shows the gray-level profile of a sub-image. It is also a gray profile shown in Fig 3.10 (a). Fig 3.14 (b), and (c) are the results obtained by using thresholding and edge detection, respectively. Similar to edge detection, the upper cartilage boundary detected by thresholding method also includes background segment. This is due to the similarly of high intensity values of pixels in cartilage and background regions.

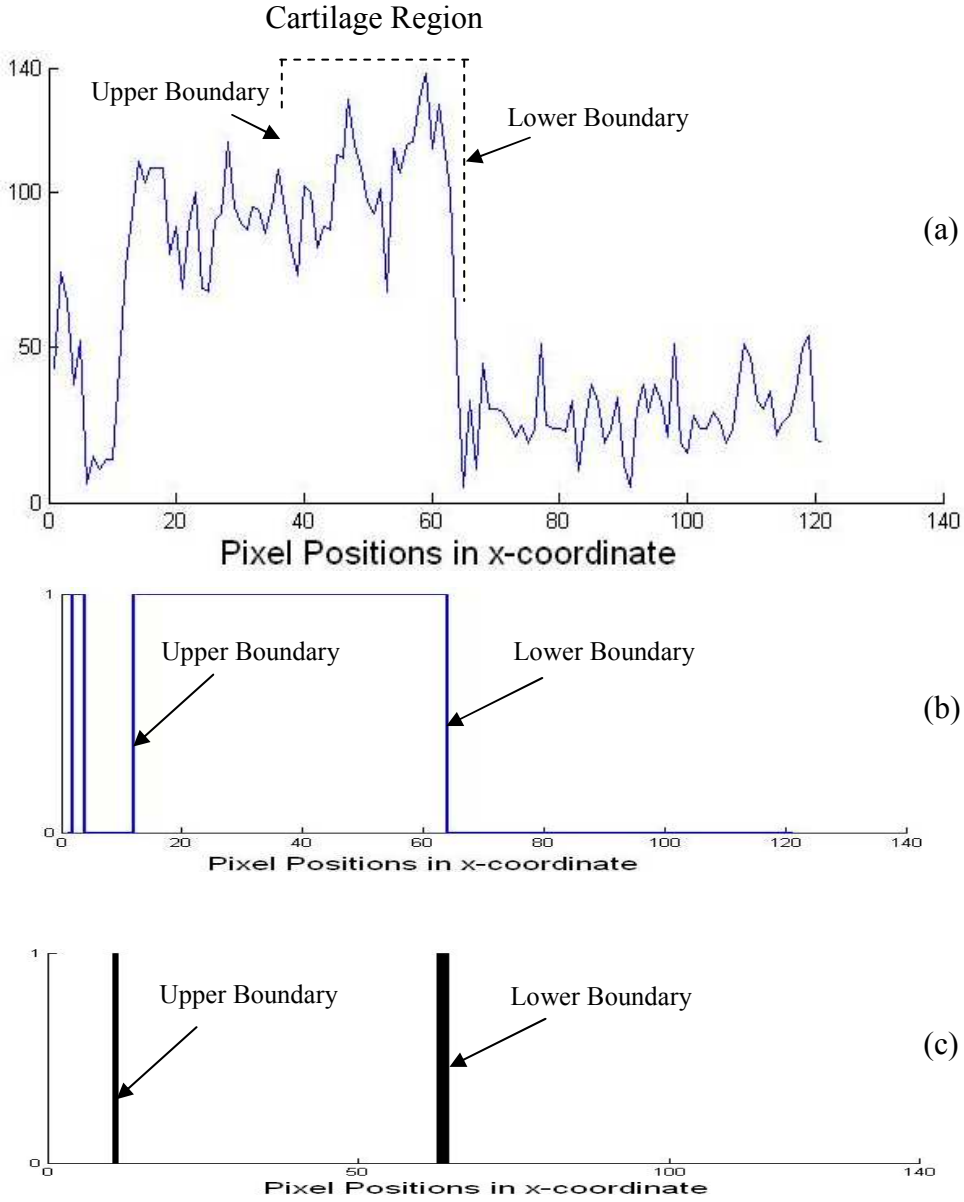


Figure 3.14 (a) Gray-level profile of sub-image, and
Result obtained by using thresholding method (b) and edge detection (c)

Conclusion

Thresholding is a method, which selects pixels that have a particular intensity value, or are within a specified intensity range. It is used to find cartilage within an image if their high intensity value (or range) is defined. Along with edge detection, thresholding improves the performance of boundary detection. However, it still cannot detect cartilage boundaries under condition that is similarity of high intensity value between cartilage and background region (Refer to Fig 3.14 as an example). Therefore, our last attempt to improve the performance of boundary detection method is carried out. That is, using statistical analysis to give an approximation of cartilage boundaries**.**

3.3.2.3 Statistical Analysis

When edge detection and thresholding fail to detect the cartilage boundaries on a sub-image, giving an approximation of a piece of cartilage boundary may be required. Generally, to provide an approximation of cartilage boundaries in a sub-image, we take a statistical analysis on a set of cartilage boundaries that were successfully found on previous sub-images.

For an input image representing matrix *I*, consider the current cartilage sub-image representing j^{th} column of matrix *I* where Boundary Detection fail to detect the cartilage boundaries. Boundaries of the cartilage on a sub-image are defined as:

$$B_{low}(x,y) = \begin{vmatrix} B_{low}(x) \\ B_{low}(y) \end{vmatrix} \quad ; \; B_{up}(x,y) = \begin{vmatrix} B_{up}(x) \\ B_{up}(y) \end{vmatrix}$$

Where $B_{up}(x,y)$ indicates upper cartilage boundary while $B_{low}(x,y)$ indicates lower cartilage boundary at location (x_2, y) according to x and y coordinates.

$$B_{up}(y) = B_{low}(y) = j$$

Suppose a set of cartilage boundaries on previous sub-images:

$$R_{up} = [\, B^1_{up}(x_1, y_1), B^2_{up}(x_2, y_2), \ldots B^n_{up}(x_n, y_n) \,]$$

$$R_{low} = [\, B^1_{low}(x_1, y_1), B^2_{low}(x_2, y_2), \ldots B^n_{u}(x_n, y_n) \,]$$

$y_i = y - i = j - i$ where $i = 1, 2, 3 \ldots n$; n is number of previous sub-images

From R_{up}, R_{low} we can apply statistical analysis to obtain an approximation of upper and lower cartilage boundaries on a current sub-image.

There are two types of statistical analysis that are employed: curve fitting algorithms and average weight calculation.

Curve fitting algorithms

Curve fitting algorithm is using a polynomial curve, which is computed from previous cartilage boundaries (low or up cartilage boundaries) in set *R* to provide an approximation of cartilage boundaries (lower or upper cartilage boundaries) on a current sub-image.

Polynomial curve is a polynomial $p(x)$ of degree n:

$$p(x) = y = p_1 x^n + p_2 x^{n-1} + \ldots + p_n x + p_{n+1}$$

For $n = 3$, it becomes cubic function:

$$p(x) = y = p_1 x^3 + p_2 x^2 + p_3 x + p_4$$

Curve fitting algorithms to compute an approximation of a cartilage boundary, for instance, a lower cartilage boundary on a current sub-image, are described as follows:

Suppose a current sub-image representing j^{th} column of matrix *I.The* lower cartilage boundary is defined as $B_{low}(x,y) = \begin{vmatrix} B_{low}(x) \\ B_{low}(y) \end{vmatrix}$ where $B_{low}(y) = j$

1. Present a set of lower cartilage boundaries on n previous sub-images:

   $$R_{low} = [\, B_1(x_1, y_1)\,, B_2(x_2, y_2)\,, \ldots B_n(x_n, y_n)\,]$$

   Where n is number of previous sub-images
2. Compute the polynomial function at each point:

   $$p(B_i(x)) = B_i(y) = p_1 B_i^3(x) + p_2 B_i^2(x) + p_4 B_i(x) + p_5$$

   where $i = 1, 2, 3 \ldots n$
3. Compute the error between two data point:

   $$\delta_i = p(B_{i+1}(x)) - p(B_i(x))$$

4. Using Least Mean Square [27] to get the coefficients p_1, p_2, p_3, and p_4 to minimum the error δ.
5. An approximation of a lower cartilage boundary on a current sub-image $B_{low}(x)$ is computed from equation:

   $$B_{low}(y) = p_1 B_{low}^3(x) + p_2 B_{low}^2(x) + p_4 B_{low}(x) + p_5$$

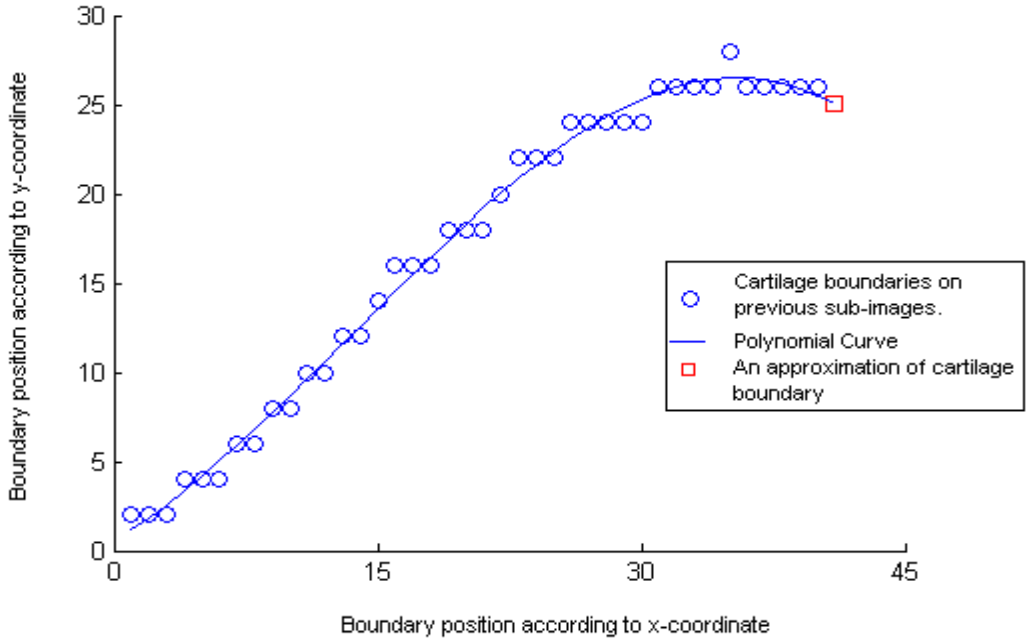


Figure 3.15 Polynomial curve used 40 data points.

Fig 3.15 demonstrates the polynomial curve, which fit all the previous cartilage boundaries. From the polynomial curve, a cartilage boundary on current sub-image is computed.

The polynomial curve shown on Fig 3.15 used 40 previous cartilage boundaries points. The curve is accurate, smooth and fit most points on the plot. Fig 3.16 illustrates a different polynomial curve, which uses the same data as Fig 3.15, but it only uses 6 previous cartilage boundaries points.

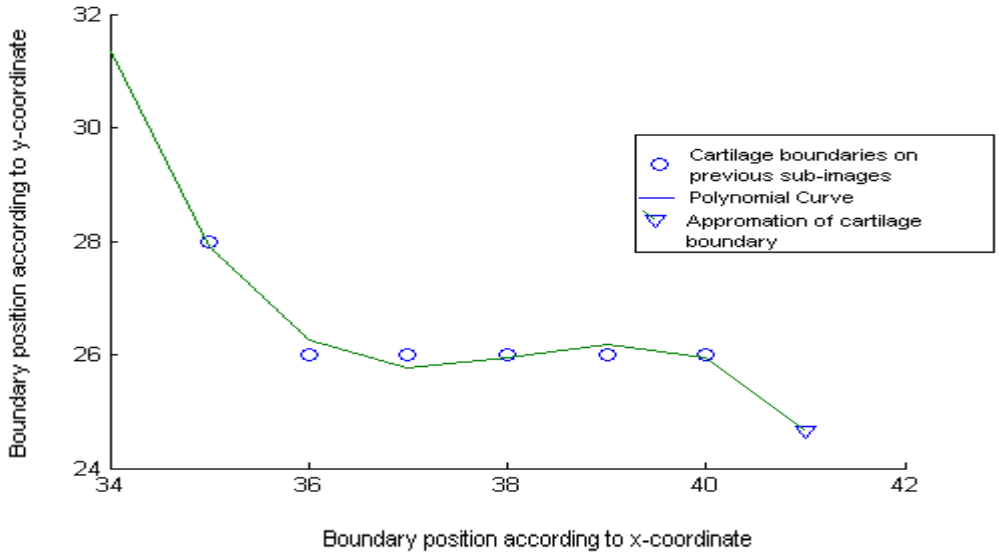


Figure 3.16 Polynomial curve used 6 data points.

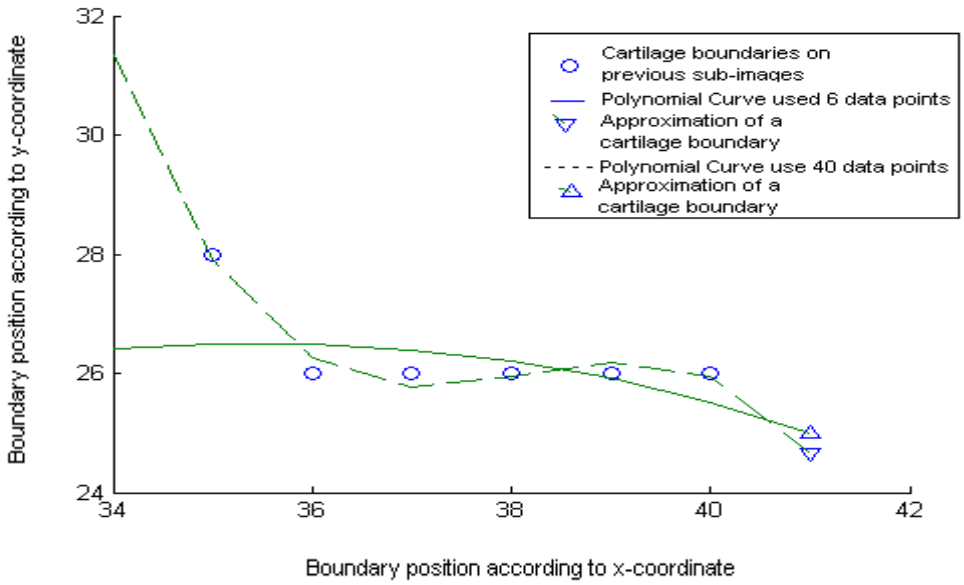


Figure 3.17 Comparison between two approximations of cartilage boundary.

From Fig 3.17, the fitting curve, which uses 40 data points, is smoother than the one, which uses only 6 data points. The slope and direction of the 40-points fitting curve are more consistent compared to of the 6-points fitting curve. Therefore, using a reasonable number of data points will improve the quality of approximation. From experiment results, using 40 data points is the most suitable choice.

Average Weight Calculation

Average weight calculation is simpler implementation than curve fitting algorithms. It computes the average value of set of previous boundary points and uses it as cartilage boundary on current sub-image. Average weight calculation algorithms, for instance, to compute a lower cartilage boundary on a current sub-image, are described as following: Consider upper cartilage boundary $B_{up}(x,y)$ are found by using boundary detection method.

1. Presents sets of cartilage boundaries on *n* previous sub-images:

$$R_{up} = [\, B^1_{up}(x_1, y_1), B^2_{up}(x_2, y_2), \ldots B^n_{up}(x_n, y_n)\,]$$

$$R_{low} = [\, B^1_{low}(x_1, y_1), B^2_{low}(x_2, y_2), \ldots B^n_{low}(x_n, y_n)\,]$$

2. Compute the weights of cartilage boundaries on each previous sub-image :

$w_i = \left| B^i_{up}(x,y) - B^i_{low}(x,y) \right|$ where i = 1, 2, 3,…n; n is number of previous sub-images

Compute the average weight of *n* previous sub-images:

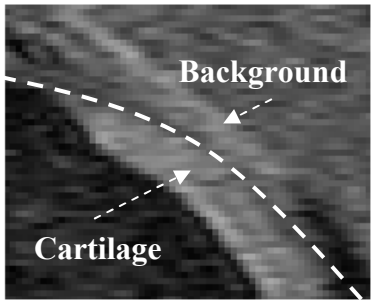
$$\overline{w} = \frac{1}{n}\sum_{i=1}^{n} w_i$$

3. An approximation of a lower cartilage boundary on a current sub-image is computed as:

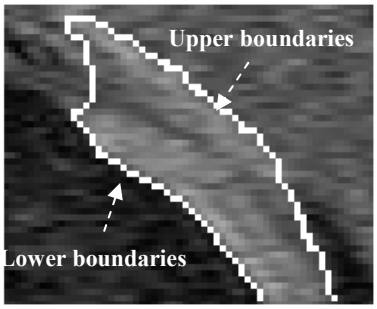
$$B_{low}(x) = B_{up}(x) + \overline{w}$$
$$B_{low}(y) = B_{up}(y) = j$$

Discussion on boundary detection using statistical analysis

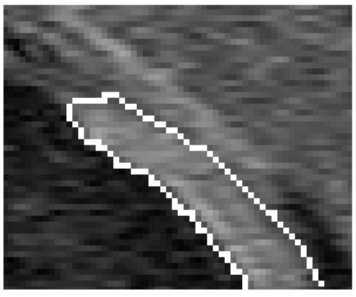
Fig 3.18 illustrates the results obtained by using statistical analysis when there are similarities of high intensity between cartilage and background pixels. Fig 3.18(b) shows the result obtained by using boundary detection without statistical analysis. In this case, the cartilage boundaries (upper boundaries) that are detected including background pixels. On the other hand, with the use of statistical analysis (Fig 3.18 (c)), up cartilage boundaries that can be detected is more accurate compared to actual boundaries.



(a)



(b)

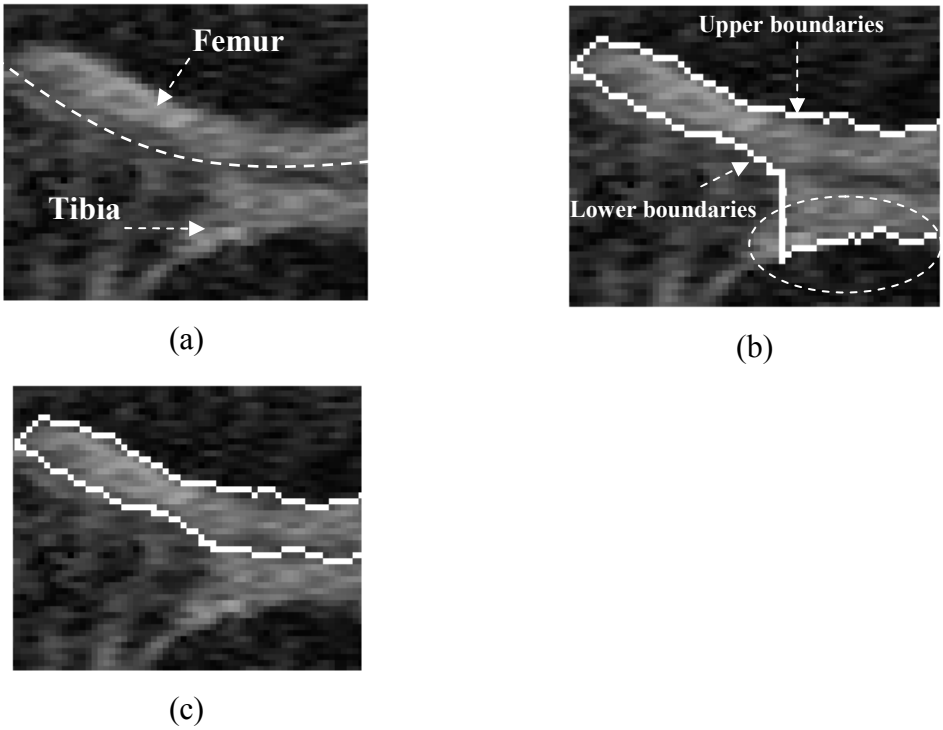


(c)

Figure 3.18 (a) Example region on an image where there are similarities of high intensity between cartilage and background pixels

Cartilage boundaries detected by using boundary detection without statistical analysis (b); and with statistical analysis (c)

Fig 3.19 demonstrates another example of using statistical analysis to detect cartilage component boundaries. Fig 3.19 (a) is an example image that we apply boundary detection to detect a cartilage component (femur) boundaries. Fig 3.19 (b) shows the result obtained by using boundary detection without statistical analysis. In this case, where femur and tibia regions are connected, femur boundaries (lower boundaries) between those regions cannot be detected correctly. Femur boundaries are likely detected wider as tibia boundaries (shown in circle in Fig 3.19 (b)). Therefore, femur boundaries obtained also contain tibia segments. Using statistical analysis, this problem can be addressed (Fig 3.19 (c)). Hence, the lower femur boundaries can be detected more accurately.



(a) (b)

(c)

Figure 3.19 (a) Example region on an image where there are similarities of high intensity between femur and tibia pixels
Femur boundaries detected by using boundary detection without statistical analysis (b); and with statistical analysis (c)

3.4 Application of Bi-directional Scanning Segmentations

An articular cartilage on the MR image includes three components namely femur, tibia, and patella. Because BSSM can be used to extract individual cartilage component from an original image, there are three processes on our application of BSSM to obtain a cartilage from an input image.

- The first process is to obtain a cartilage component: femur from original input image (Refer to Fig 3.20).

- The second process is to obtain a cartilage component: tibia. Input image which is used in the second process is an image in which femur is extracted (Refer to Fig 3.20).

- The final process is to obtain the last cartilage component: patella. Input image in this time is an image in which femur and tibia are extracted (Refer to Fig 3.20).

Fig 3.20 illustrates the algorithms to extract a cartilage from an input image by a flow chart. In this implementation, the decision of the appearance of different cartilage components depends on different types of an input image. As mentioned in Chapter 2, section 2.3, following table illustrates distribution of image types according to image sequence.

Image Sequence	Cartilage types
1 - 13	Do not appear
14-18	Femur and Tibia
19 - 21	Femur , Tibia, and Patella
22 - 37	Femur and Patella
38 - 45	Femur, Tibia, and Patella
46 - 49	Femur, and Tibia
50 - 62	Do not appear

Table 3.1 Image types according to image sequence

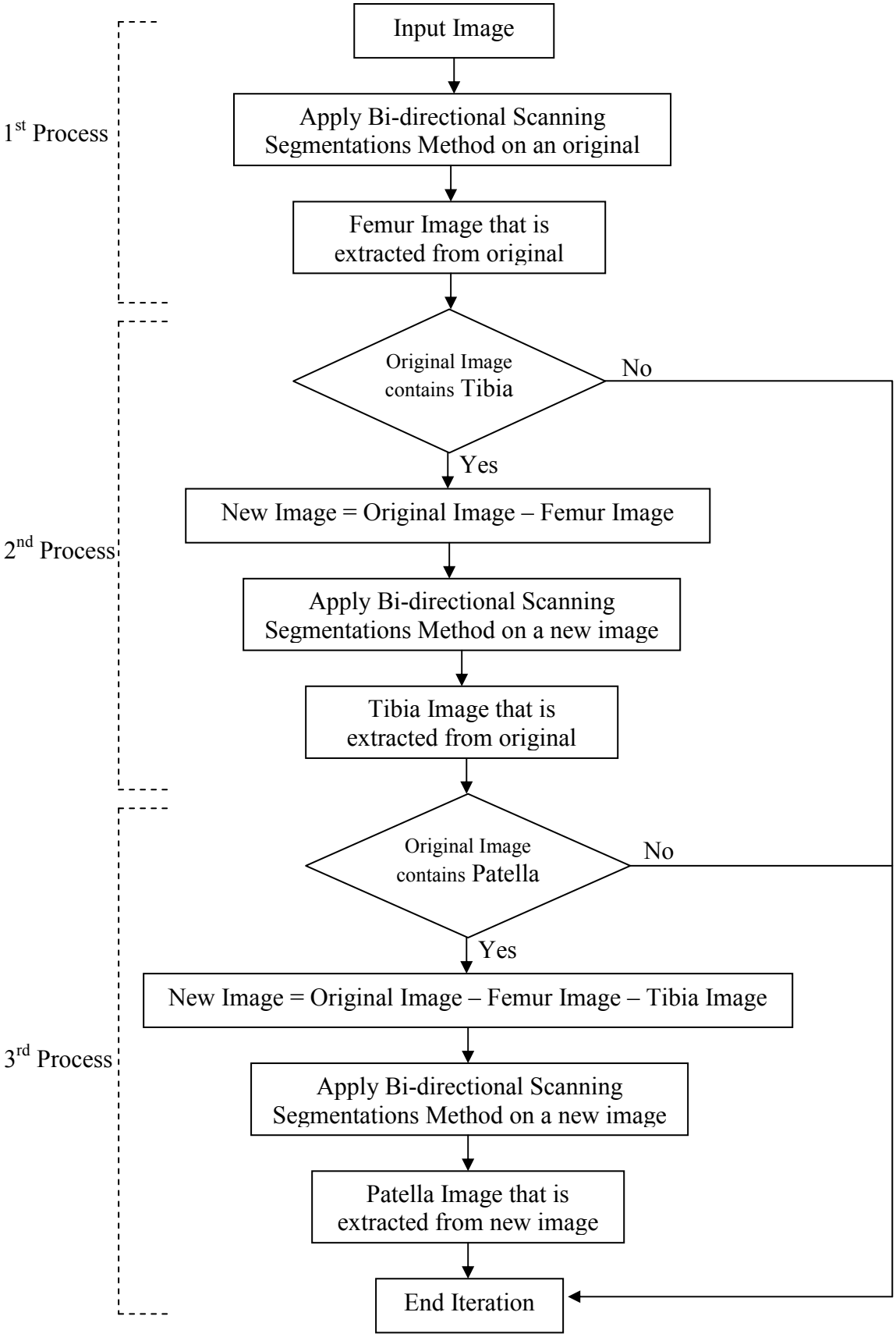


Figure 3.20 Flowchart of application of BSSM to extract individual cartilage parts from an original image.

3.5 Experiment Results

In the thesis, each method will be tested in two types: visual results and quantitative evaluation. Visual results will firstly described in this section as well as in section 4.7 - Chapter 4 (Introduction of method 2), and section 5.5 – Chapter 5 (Introduction of method 3). Quantitative evaluation is then computed and demonstrated in Chapter 6 (Area and Volume Calculation).

This section presents the results obtained by using BSSM to extract cartilage from images sets. According to image type, we have three sets of images for testing as follows:

- Image set 1: Images that contain Femur, Tibia, and Patella.
- Image set 2: Images that contain Femur, and Tibia.
- Image set 3: Images that contain Femur and Patella.

Image Set 1

Fig 3.21 shows the cartilage images extracted from Image Set 1. In general, cartilage image is composed of femur, tibia, and patella images. This is only true in case of image 1 and 2 when patella appears to be isolated from other components on original input image. BSSM cannot distinguish femur from patella when patella and femur are connected vertically. Hence, it does not extract femur from patella separately. Patella is then considered as femur and femur image contains patella. Cartilage image now is composed of femur image and tibia image. In case of tibia and femur, this problem does not exist since they are connected horizontally.

Intuitively, cartilage images that are extracted by using BSSM are reasonably close to actual cartilage on original images except in case of image number 2 and 6 in Fig 3.21. In both cases, several femur segments are not detected. This is because of a very low contrast between those femur segments and background regions.

Image Set 2

Fig 3.22 shows the cartilage images extracted from Image Set 2. In this case, cartilage image is composed of femur image and tibia image. Generally, the results obtained are reasonable, except in case of image number 6 and 8. In both cases, some tibia segments are not detected. The reason is a low contrast between tibia segments and background regions.

Image Set 3

Fig 3.23 shows the cartilage images extracted from Image Set 3. Because femur and patella are connected vertically, so that BSSM cannot extract femur from patella

separately. Therefore, femur image contains tibia. Cartilage images shown in Fig 3.23 are femur images.

BSSM is good at extracting cartilage from original image. Results obtained from image set 3 are specifically accurate than results obtained from image set 1 and 2. This is because contrasts between cartilage and background regions are high on image set 3.

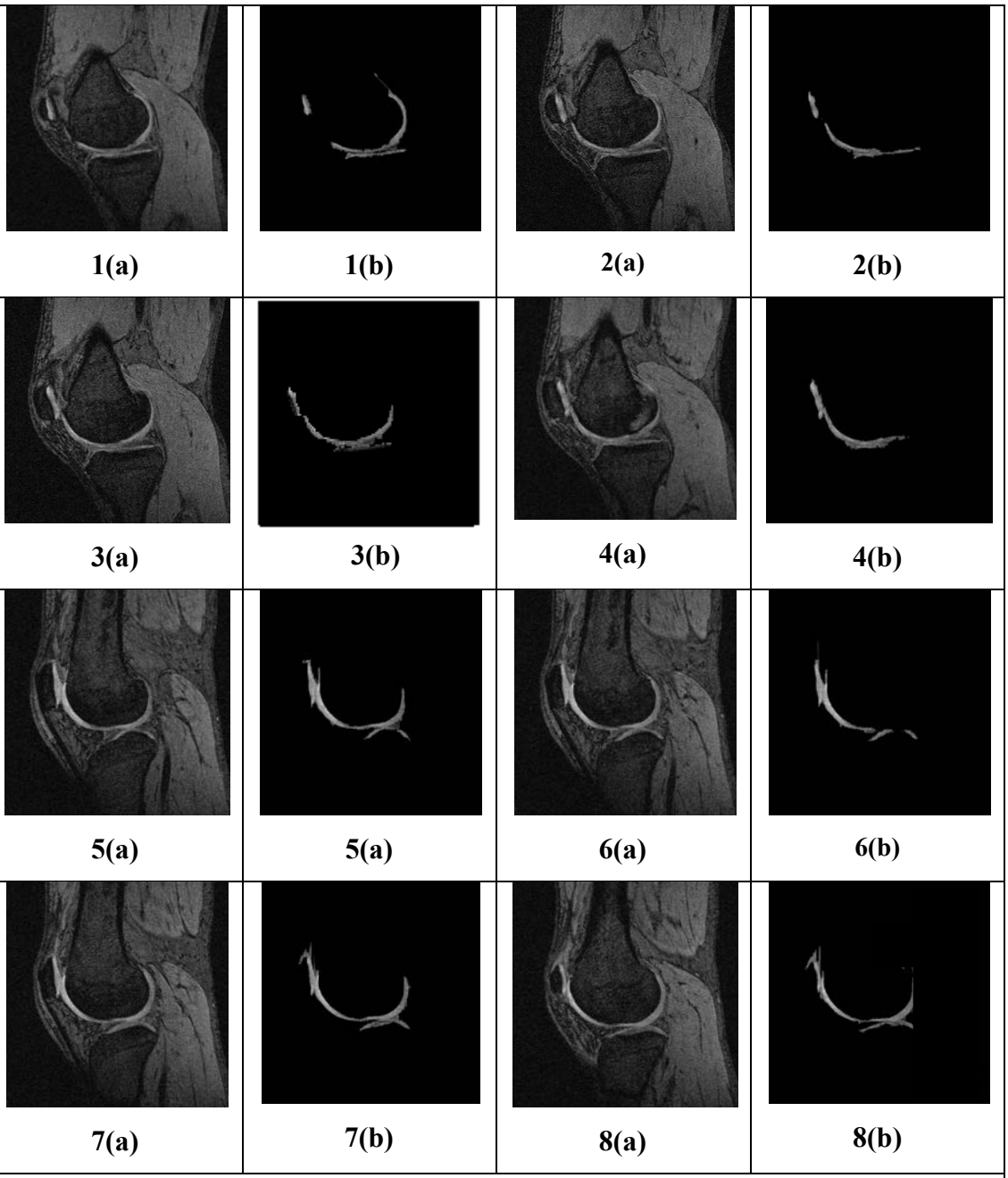


Figure 3.21 (a) Images on Image Set 1 and (b) Cartilage images extracted from Image Set 1 by using BSSM.

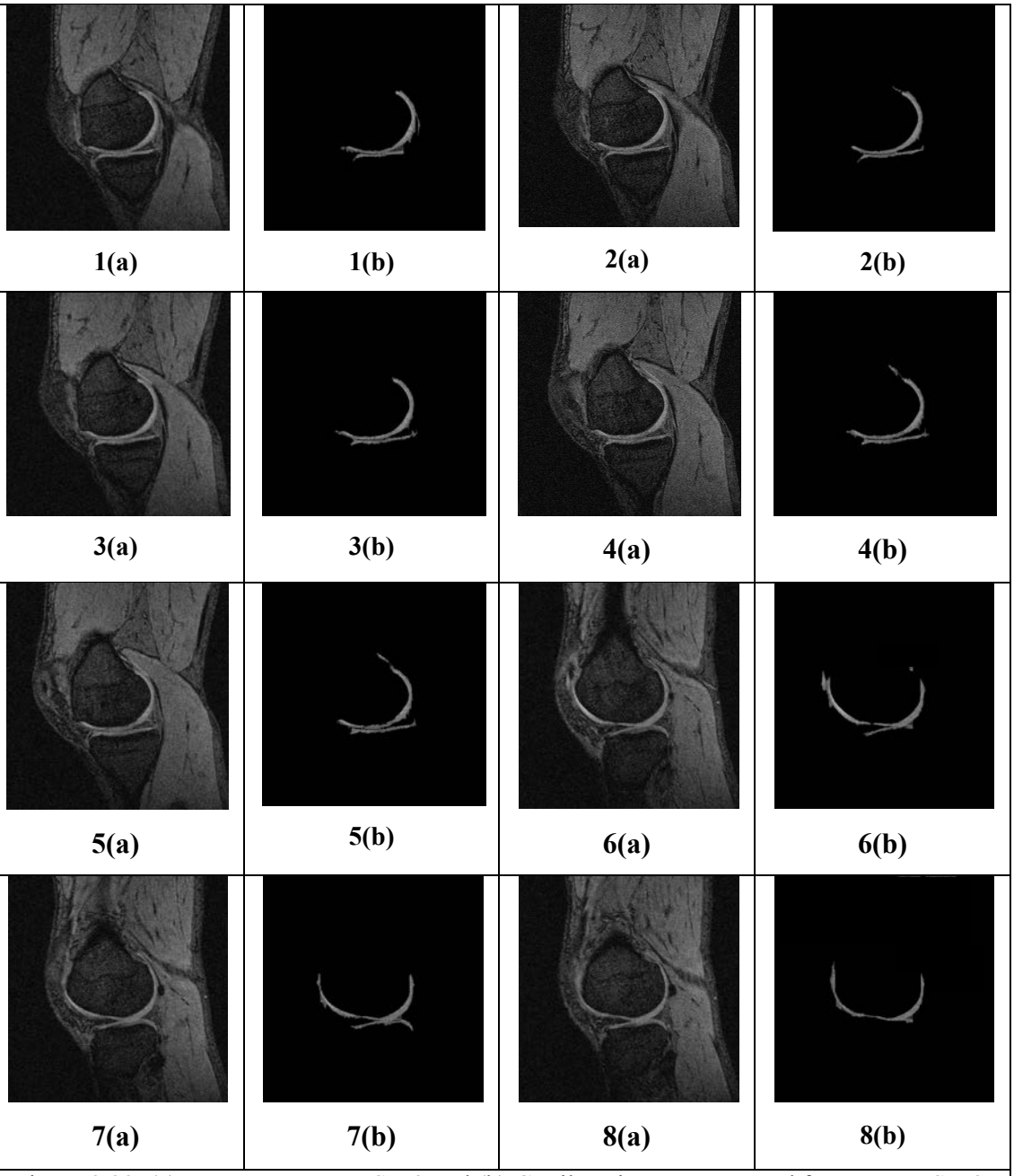


Figure 3.22 (a) Images on Image Set 2 and (b) Cartilage images extracted from Image Set 2 by using BSSM

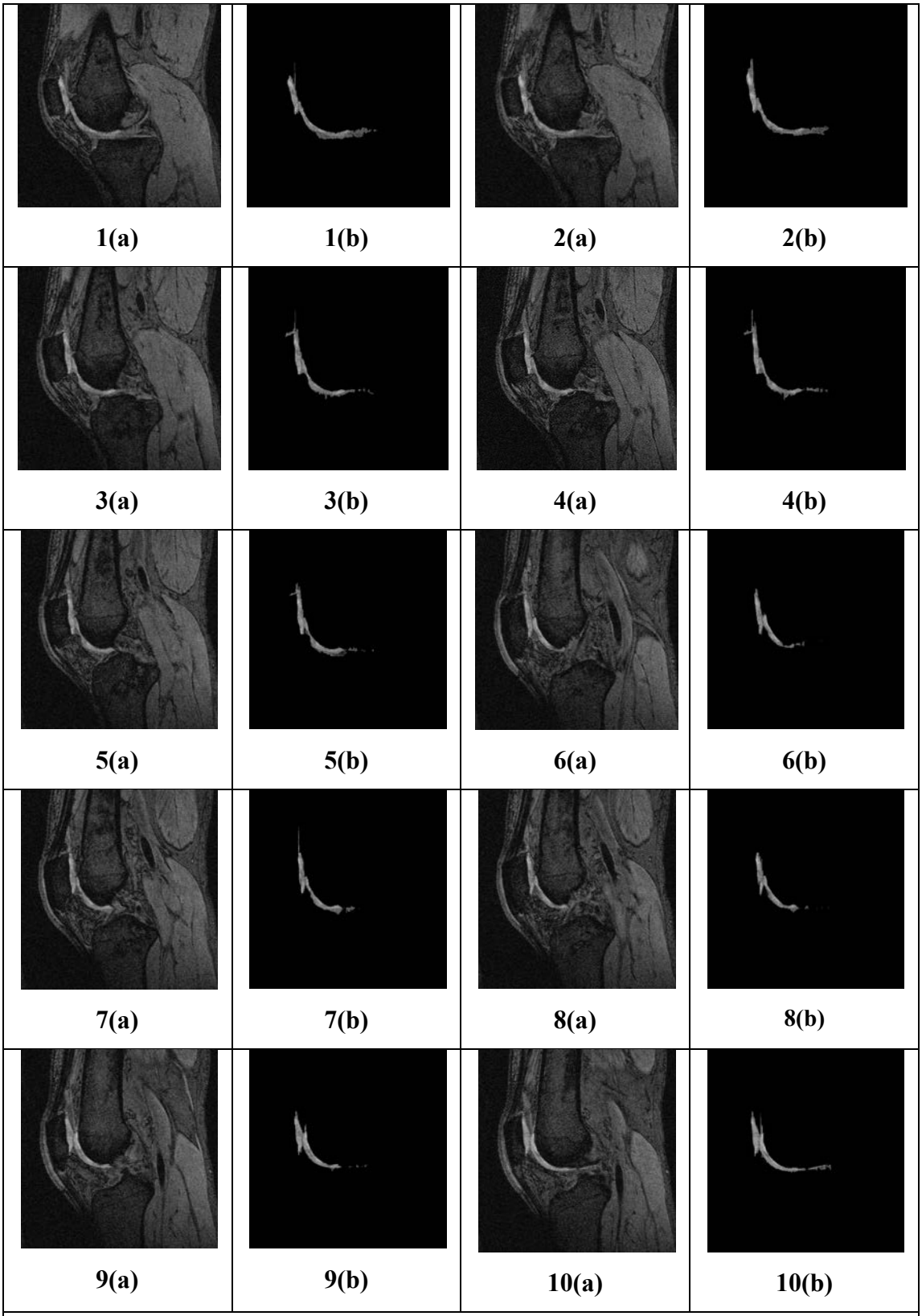


Figure 3.23 (a) Images on Image Set 3 and (b) Cartilage images extracted from Image Set 3 by using BSSM

3.6 Conclusion

BSSM is an automatic method that is used to extract the cartilage from MR images. It is based on the two properties of intensity value of pixels: discontinuity and similarity. It can also be based on the statistical analysis such as curve fitting algorithms and average weight calculation. This approach works well when the contrast between the cartilage and background regions is high. That is, the cartilage pixels have high intensity values while its background pixels have low intensity values. When the contrast is low, BSSM often goes in struggle in detecting the cartilage boundaries.

Fig 3.24 illustrates the disadvantage of BSSM when it fails to detect the cartilage boundaries when there is a low contrast between cartilage and background regions.

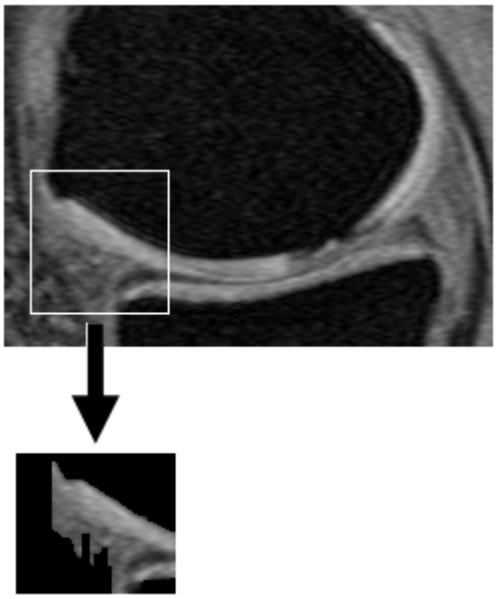


Figure 3.24 Example of low contrast between cartilage and background regions.

On the other hand, the success of BSSM depends on how we define rules or principles of combination between edge detection, thresholding and statistical analysis. Therefore, we need to find another automatic method, which is more reliable and accurate. We are going to present the neural network classifier method (NNCM), which is based on artificial neural network in next chapter.

Chapter 4

NEURUAL NETWORK CLASSIFIER

4.1 Overview

Even though high-speed computers with central processing units capable of performing millions of operations per second have become widely available, the human brain can still perform a variety of tasks much more efficiently than computers. Task such as recognizing cartilage parts from MRI scans can be performed effortlessly by the human brain, but can only be performed in controlled situations by conventional computers. The reason that the human brain can perform so efficiently is that it uses parallel computation effectively. Thousands or even millions of nerve cells called neurons are organized to work simultaneously on the same problem [28].

Therefore, in this chapter, we present a method named neural network classifier method (NNCM), which use artificial neural network as cartilage classification on a MR image [1].

4.2 Artificial Neural Networks

Artificial neural networks are an attempt to emulate the processing capabilities of biological neural systems. The basic idea is to realize systems capable of performing complex processing tasks by interconnecting a set of very simple processing elements that might even work in parallel. They solve cumbersome and intractable problems that are difficult for conventional computers or human beings such as pattern recognition or clustering data by learning directly from data. An artificial neural network usually consists of some simple processing units, namely, neurons, via mutual interconnection. It learns to solve problems by logically adjusting the strength of the interconnections according to input data. Moreover, it can be easily adapted to new environments by learning. In addition, it can deal with information that is noisy, inconsistent, vague, or probabilistic.

The main features of artificial neural networks are their massive parallel processing architectures and the capabilities of learning from the present inputs. They can be

utilized to perform a specific task only by means of adequately adjusting the connection weights, that is, by training them with the presented data. For each type of artificial neural network, there exists a corresponding, learning algorithm by which we can train the network in an iterative updating manner.

There exist many types of neural networks that solve a wide range of problems in the area of image processing. There are also many types of neural networks and they are determined by the type of connectivity between the processing elements, characteristics, and training or learning rules. These rules specify an initial set of weights and indicate how weights should be modified during the learning process to improve network performance. [29].

There are two types of classification known in image processing: region-based and pixel-based classification. An object is classified in region-based classification based on features that are usually computed to describe the entire object. Those are mostly geometric features as various sizes and shape measurements. Fig 4.1(a) illustrates this fact. Other features are computed at pixel level. This means that in pixel-level classification a feature value is computed for each pixel and so each pixel is classified individually as show in Fig 4.1(b).

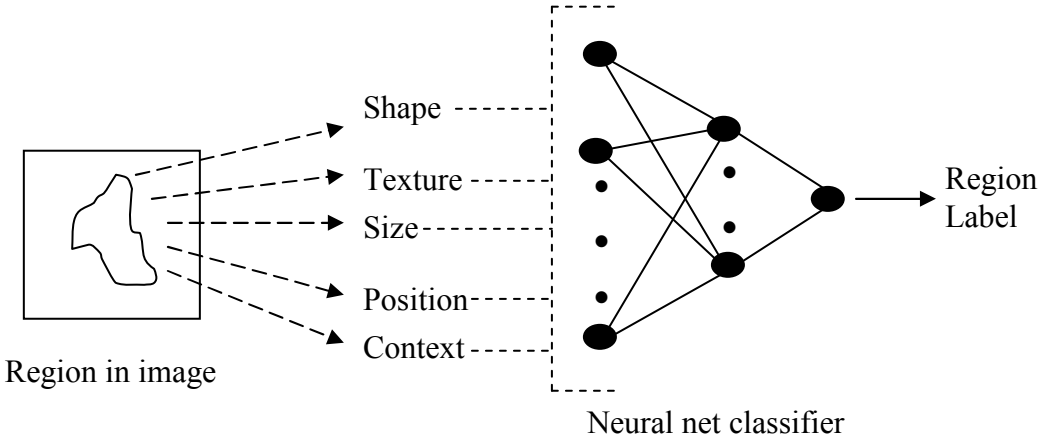


Figure 4.1 (a) Classification of a region based upon a feature set.

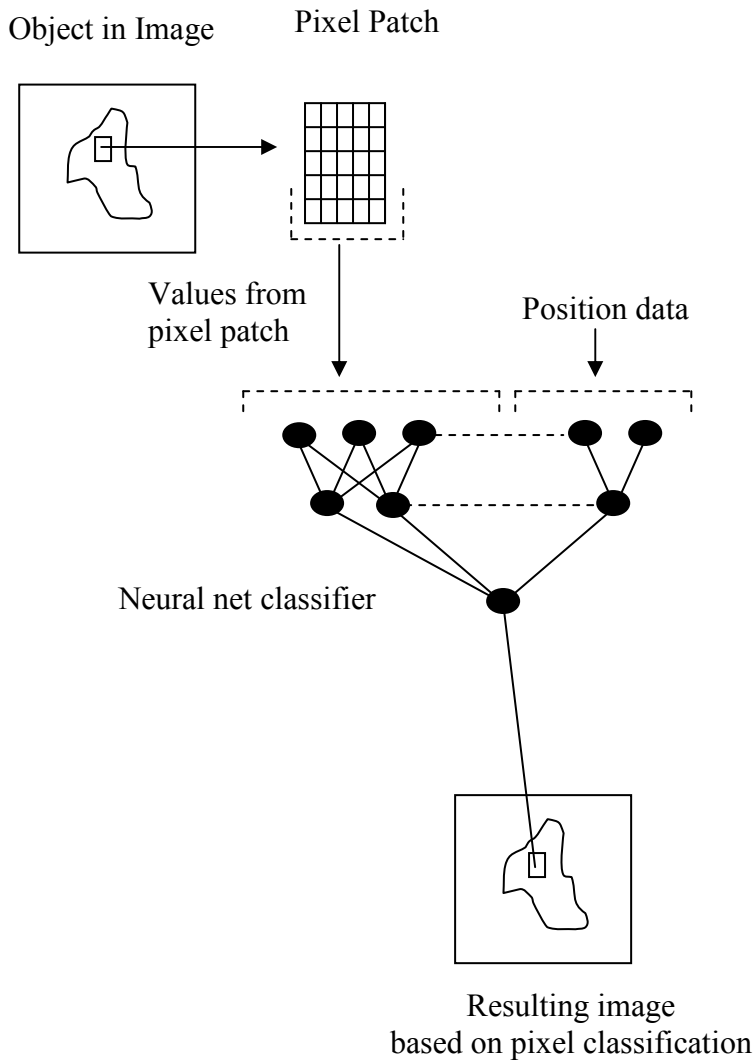


Figure 4.1 (b) Classification at pixel level.

In our chapter, we will apply pixel-base classification for cartilage recognition.

4.3 Multilayer perceptrons (MLP)[30]

Multilayer perceptrons (MLP) are one of the most important types of neural networks because many applications [31] are successful implementations of MLPs. MLPs are specially suited for object recognition problems. They are fast and reliable networks for the problems they can solve.

Fig 4.2 illustrates how MLP is used as neural network classifier (NNC) for object recognition base on pixel level classification. For each pixel on MR image, a pixel patch is generated. It is used as the input for the neural network classifier (NNC) as well as its location. Throughout the network, pixel is classified as object pixel or background pixel.

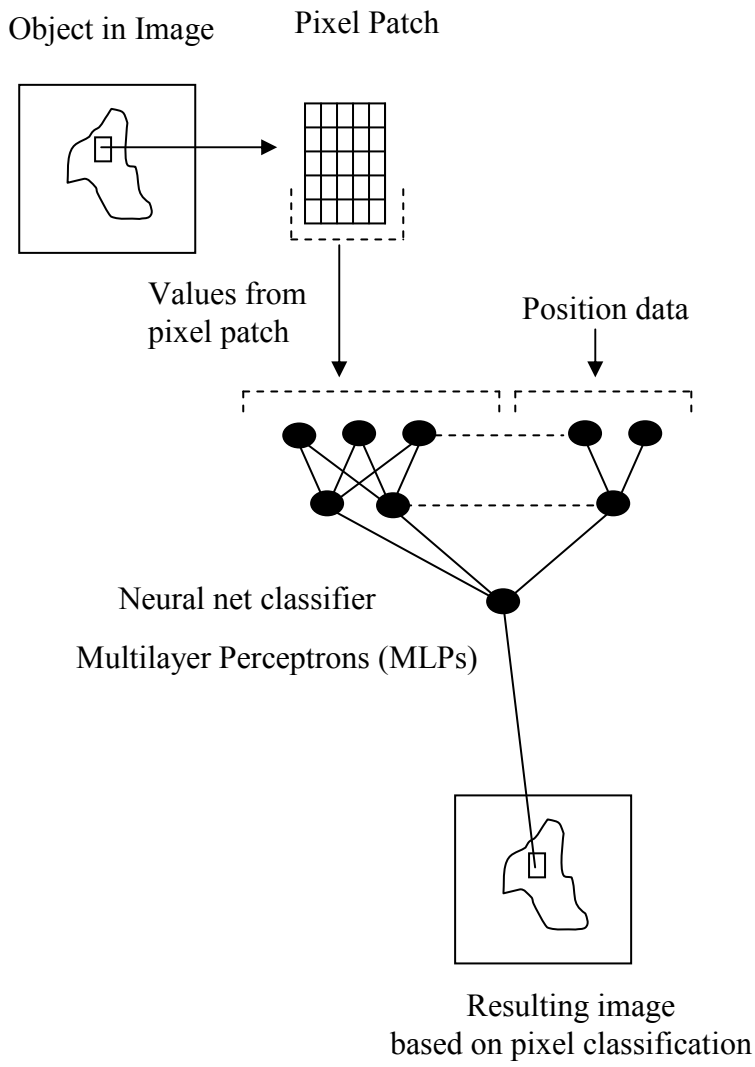


Figure 4.2 Using multilayer perceptrons as NNC for cartilage classification at pixel level.

Typically, the network consists of a set of processing units that constitute the input layer, one or more hidden layers, and an output layer. The input signal propagates through the network in a forward direction, on a layer-by-layer basic. Fig 4.3 (a) illustrates the configuration of the MLP.

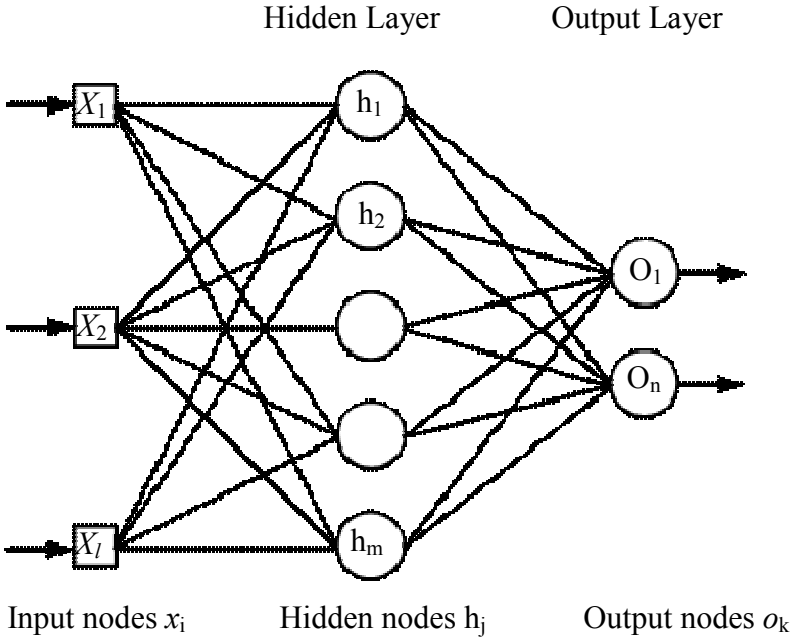


Figure 4.3 (a) Two-layer perceptron.

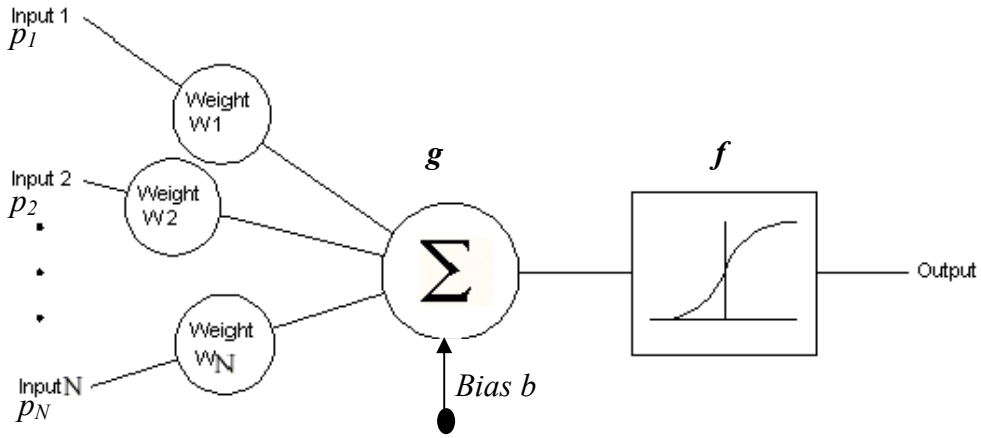


Fig 4.3 (b) Propagation rule and activation function for the MLP network.

A node in a hidden layer is connected to every node in the next layer and previous it. In Fig 4.3 (a) weight w_{ij} connects input node x_i to hidden node h_j and weight v_{jk} connects h_j to output node o_k. Classification begins by presenting a pattern to the input

nodes x_i, $1 \le i \le l$. From there data flow in one direction through the perceptron until the output nodes o_k, $1 \le k \le n$, are reached. Output nodes will have a value of either 0 or 1. Thus, the perceptron is capable of partitioning its patterns space into 2^n classes.

The MLP algorithm is described as follow:

1. Present the pattern $\boldsymbol{p} = [p_1, p_2, \ldots, p_l] \in \Re^l$ to the perceptron, that is, set $x_i = p_i$ for $1 \le i \le l$.
2. Compute the values of the hidden layer nodes as it is illustrated in Fig 4.3(b)

$$h_j = \frac{1}{1 + \exp\left[-\left(w_{oj} + \sum_{i=1}^{l} (w_{ij} x_i + b) \right) \right]} \qquad 1 \le j \le m$$

   The activation function is the sigmoid function $f(x) = \frac{1}{1 + \exp(-x)}$ and it is also the most common form of activation function in MLP.
3. Calculate the values of the output nodes according to

$$o_k = \frac{1}{1 + \exp\left(v_{ok} + \sum_{j=1}^{m} (v_{jk} h_j + b) \right)} \qquad 1 \le k \le n$$

4. The class $\boldsymbol{c} = [c_1, c_2, \ldots, c_n]$ that the perceptron assigns at $\boldsymbol{p}$ must be a binary vector. So o_k must be the threshold of a certain class at some level τ and depends on the application.
5. Repeat steps 1, 2, 3 and 4 for each pattern that is to be classified.

In order to obtain a NNC, there are two main processes: building and training a network.

- Building a network is to specify a neural network structure and some relative parameters in a network such as input size, number of layers, activation function for the network etc…
- Training a network is to obtain the all weights and biases of a network according to training data.

4.4 Neural Network Classifier Definition:

In order to classify a pixel into two classes: cartilage and background class, two-layer perceptrons is the most suitable choice. Fig 4.4 presents the structure of the two-layer perceptrons as neural network classifier (NNC):

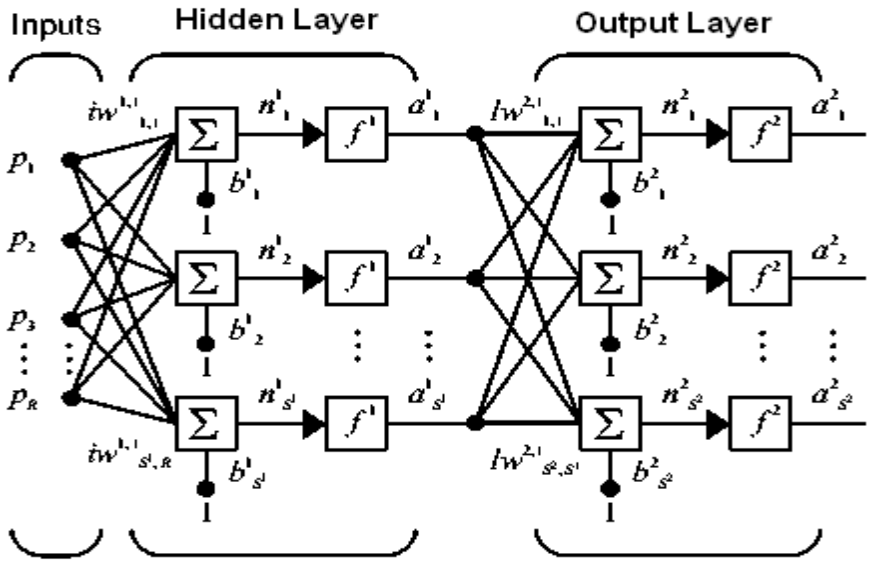


Figure 4.4 Two-layer peceptrons

Where R is the number of elements in input vector, S^1 is number of neurons (or nodes) in hidden layer and S^2 is number of neurons in output layer. The weights ***iw(S^1,R)*** and biases $\boldsymbol{b^1}(S^1)$ are weights and biases between input and hidden layer. The weights ***lw(S^2,S^1)*** and biases $\boldsymbol{b^2}(S^2)$ are weights and biases between hidden and output layers.. f^1 is activation function of hidden layer and f^2 is activation function of output layer.

$\boldsymbol{a^1}$ is output of hidden layer, it also is input of output layer. $\boldsymbol{a^1}$ is defined as:

$$a^1 = f^1(iw_{s^1,R} + b^1)$$

$\boldsymbol{a^2}$ is output of output layer, it also is output of a network. $\boldsymbol{a^2}$ is defined as:

$$a^2 = f^2(lw_{S^2,S^1}.a^1 + b^2)$$

4.4.1 Input Vectors.

As mentioned in Section 4.3, input of the network is the vector ***p*** computed from pixel patch of a pixel on the MR image. Pixel patch is a matrix formed by a pixel and its neighbourhood. In another ways, it is an image window surrounding a pixel. Fig 4.5 illustrates the relationship between the pixel patch and the input vector.

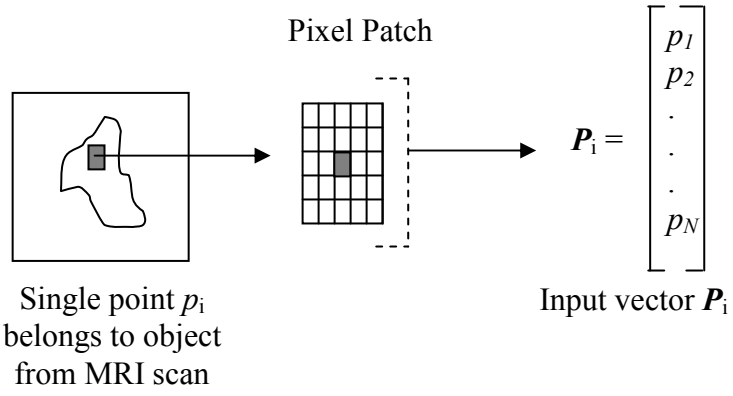


Figure 4.5 Generation of the input vector from a point on MR image.

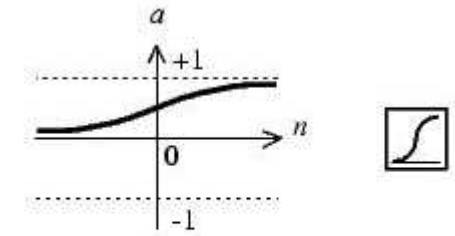
Since input vector is formed from pixel patch, there is relationship between sizes of them. A pixel patch which is a matrix has size of [PxQ], input vector's size is then [1,(PxQ)]. For instance, [3x3] pixel patch resulting in [1x9] input vector.

Input vectors represent the features of object (cartilage) or background on a MR image. Base on that, the network is trained to dichotomise those classed. Hence, the input vectors show more information of objects or background features will improve the network performance. For that reason, an appropriated size of input vector provides the best results.

By simulation experiments, we found the most suitable size of input vector is [1x81] (pixel patch size is [9x9]).

4.4.2 Activation Function

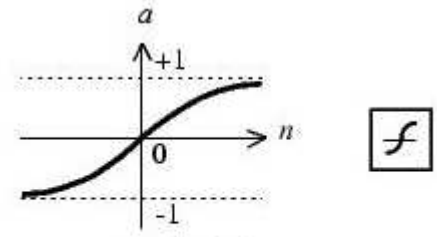
We have three activation functions that can be used in MLPs. First, Log-Sigmoid function is used most often.



$a = logsig(n)$

Log-Sigmoid Transfer Function

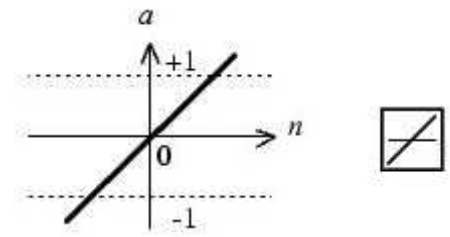
Alternatively, multilayer networks can use the tan-sigmoid transfer function tansig.



$a = tansig(n)$

Tan-Sigmoid Transfer Function

Occasionally, the linear transfer function purelin is used in backpropagation networks.



$a = purelin(n)$

Linear Transfer Function

Depend on the application of the MLPs, we can decide the most suitable function. To solve problems in cartilage classification, we want to constrain the output value of the network in the range [0, 1] where output value 0 representing background pixel and 1 representing cartilage pixel. Because the function log-sigmoid generates outputs between 0 and 1 as the neuron's net input goes from negative to positive infinity, the log-sigmoid function is best suitable selection for activation function.

4.4.3 Number of Neuron in Layers

Due to classification of two classes: cartilage class and background class, only one neuron on output layer is required.

The number of neurons in the hidden layer was heuristically chosen to be 40 (the average of the number of elements in input vector). There are no known rules for specifying the number of neurons in the hidden layers of a network, so this number generally is based either on prior experience or simply chosen arbitrarily and then refined by testing.

4.5 Network Training

Training a network is to specify all weights and biases of a network such as the weights ***iw*** and biases $\boldsymbol{b}^1$ between input and hidden layer, and the weight ***lw*** and biases $\boldsymbol{b}^2$ between hidden and output layers.

4.5.1 Back-propagation Algorithm

The network is trained by a popular algorithm know as the error back-propagation algorithm. This process consists of two passes through the different layers of the network: a forward and a backward pass. During the forward pass, a training pattern is presented to the perceptron and classified.

The backward pass recursively, level by level, determines error terms used to adjust to the perceptron weights. The error terms at the first level of the recursions are a function of $\boldsymbol{c}^t$ and output of the perceptron (o_1, o_2… o_n). After all the errors have been computed, weights are adjusted using the error terms that correspond to their level. The algorithm description of the back-propagation is given here:

1. Initialization: initialize the weights of the perceptron randomly with numbers between -0.1 and 0.1; that is,

w_{ij} = random([-0.1,0.1]) $\quad 0 \le i \le l,\ 1 \le j \le m$

v_{jk} = random([-0.1,0.1]) $\quad 0 \le j \le m,\ 1 \le k \le n$

2. Presentation of training examples:

Present $\boldsymbol{p}^t = [p^t_1, p^t_2, \ldots, p^t_l]$ from the training pair $(\boldsymbol{p}^t, \boldsymbol{c}^t)$ to the perceptron and apply steps 1, 2, and 3 from the perceptron classification algorithm described earlier in Section 4.3.

Collecting the training data will be described in section 4.5.2

3. Forward computation: Compute the error δ_{ok}, $1 \le k \le n$ in the output layer using

$$\delta_{ok} = o_k(1-o_k)(c^t_k - o_k)$$

Where $\boldsymbol{c}^t = [c^t_1, c^t_2, \ldots, c^t_n]$ represents the correct class of $\boldsymbol{p}^t$. The vector $(o_1, o_2 \ldots o_n)$ represents the output of the perceptron.

4. Forward computation: Compute the error δ_{hj}, $1 \le j \le m$, in the hidden layer using:

$$\delta_{hj} = h_j(1-h_j)\sum_{k=1}^{n}\delta_{ok}v_{jk}$$

5. Backward computation: Let v_{jk} denote the value of weight v_{jk} after the t^{th} training pattern has been presented to the perceptron. Adjust the weights between the output layer and the hidden layer using

$$v_{jk}(t) = v_{jk}(t-1) + \eta\delta_{ok}h_j$$

Parameter $0 \le \eta \le 1$ represents the learning rate.

6. Backward computation: Adjust the weights between the hidden layer and the input layer according to:

$$w_{ij}(t) = w_{ij}(t-1) + \eta\delta_{hj}p^t_i$$

7. Iteration: Repeat steps 2 through 6 for each element of the training set. One cycle through the training set is called an iteration.

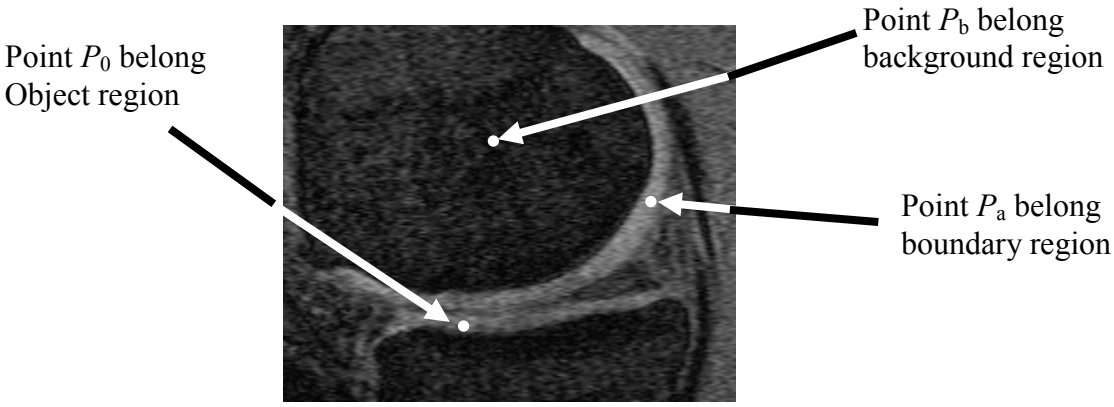
There are several algorithms that are called training algorithms to adjust the weights and biases of a network. We will discuss about this matter in section 4.5.3.

4.5.2 Collecting Training Data

Back-propagation network requires two types of training data: input vectors (see Section 4.4) that represent two classes: cartilage class and background class, and the corresponding target vectors.

Input vectors are collected from a typical MR image. Fig 4.7 shows an image in which training data are collected. There are three types of pixel patch corresponding to pixel's location: the cartilage pixel (pixel that lie on cartilage region), the background pixel (the pixel lie on the background region), and the boundary pixel (the pixel lie on

the adjacent location between cartilage and background regions). Therefore, we have three types of input vector.



(a)

P_{11}				P_{15}
		P_i		
P_{51}				P_{55}

(b)

99	103	71	86	101
115	104	93	88	87
106	100	104	105	97
90	102	93	100	99
88	98	99	102	111

(c)

95	86	97	92	97
92	100	110	100	50
88	105	99	47	52
101	112	60	33	32
99	51	45	17	16

(d)(d)

40	27	15	12	20
41	35	19	6	42
35	22	17	21	30
33	47	30	11	32
26	22	35	19	12

(e)

Figure 4.6 Example of pixel and pixel patch from MRI scan.

(a) Original MRI scan : Point P_0 – a pixel belongs to cartilage region, point P_b – a pixel belongs to background region, and P_b – a pixel belongs to boundary region.

(b) Structure of pixel patch that generated from point P_i and its neighbourhoods.

(c) Pixel patch of point P_0.

(d) Pixel patch of point P_b.

(e) Pixel patch of point P_a.

Fig 4.6 illustrates three types of pixel and its features. Corresponding to pixel level (gray level), the pixel patch of a cartilage pixel often contains high values (Fig 4.6 (c))

while the pixel patch of a background pixel often contains low values (Fig 4.6 (e)). The pixel patch of boundary pixel contains both low and high values and it falls into two regions (see Fig 4.6 (d)). It is due to the characteristic of MRI scan. The cartilage pixels are visualized with high contrast to its surrounding pixels (background).

Figure 4.7 Original image in which training data collected

Target vectors

The target vectors values are set according to different types of input vectors. For input vector of an object (cartilage) class, the value of target vector is 1; for input vector of a background class, the value of target vector is 0. The size of single target vector is 1-by-1. In additional, corresponding to set of input vectors, the size of target vectors is 1x[number of input vectors].

Fig 4.8 illustrates the relations between the input vectors and the targets vector. Figure 4.8 (a) shows the size relationship while Figure 4.8 (b) shows the value relationships. *m* input vectors produces target vectors of size 1x*m*. In Figure 4.8 (b), input vector P_1 and P_3 are object vectors and input vector P_2 is background vector. As the result, the value of target vectors are [1 0 1].

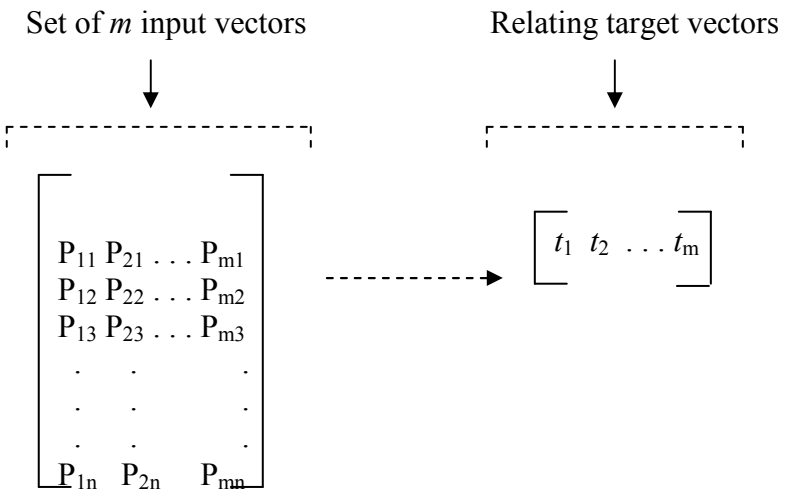


Figure 4.8 (a) Example of size relation between input vectors and target vectors.

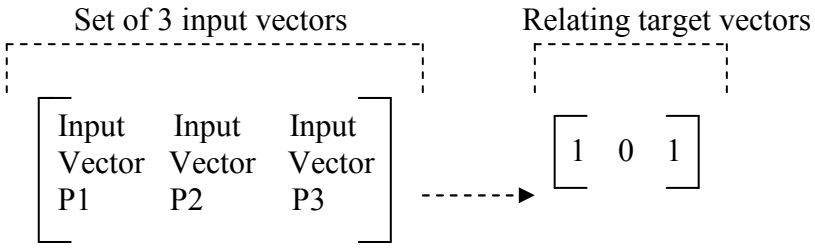


Figure 4.8 (b) Example of value relation input vectors and target vectors.

4.5.3 Training Algorithms

For back-propagation network, there are several training algorithms to adjusted all the weights and biases of the network to minimize the mean square error (MSE) between the network outputs and the target outputs. Due to faster training, we applied three types of training algorithms using optimization techniques: Conjugate Gradient, Quasi-Newton, and Levenberg- Marquardt.

Conjugate Gradient Algorithms

The basic back-propagation algorithm adjusts the weights in the steepest descent direction (negative of the gradient), the direction in which the performance function is decreasing most rapidly. It turns out that, although the function decreases most rapidly along the negative of the gradient, this does not necessarily produce the fastest convergence. In the conjugate gradient algorithms, a search is performed along conjugate directions, which produces generally faster convergence than steepest descent directions. There are four variations of conjugate gradient algorithms:

Fletcher-Reeves Update [32]

All the conjugate gradient algorithms start out by searching in the steepest descent direction (negative of the gradient) on the first iteration.

$$p_0 = -g_0$$

A line search is then performed to determine the optimal distance to move along the current search direction:

$$x_{k+1} = x_k + \alpha_k p_k$$

Then the next search direction is determined so that it is conjugate to previous search directions. The general procedure for determining the new search direction is to combine the new steepest descent direction with the previous search direction:

$$p_k = -g_k + \beta_k p_{k-1}$$

The various versions of the conjugate gradient algorithm are distinguished by the manner in which the constant β_k is computed. For the Fletcher-Reeves update the procedure is:

$$\beta_k = \frac{g_k^T g_k}{g_k^T - 1 g_k - 1}$$

This is the ratio of the squared norm of the current gradient to the squared norm of the previous gradient.

Polak-Ribiere Update [33]

Another version of the conjugate gradient algorithm was proposed by Polak and Ribiére. As with the Fletcher-Reeves algorithm, the search direction at each iteration is determined by:

$$p_k = -g_k + \beta_k p_k - 1$$

For the Polak-Ribiére update, the constant β_k is computed by:

$$\beta_k = \frac{\Delta g_k^T - 1 g_k}{g_k^T - 1 g_k - 1}$$

This is the inner product of the previous change in the gradient with the current gradient divided by the squared norm of the previous gradient.

Powell-Beale Restart [34, 35]

For all conjugate gradient algorithms, the search direction is periodically reset to the negative gradient. The standard reset occurs when the number of iterations is equal

to the number of network parameters (weights and biases), but there are other reset methods that can improve the efficiency of training. One such reset method was proposed by Powell, based on an earlier version proposed by Beale [36]. This technique restarts if there is very little orthogonality left between the current gradient and the previous gradient. This is tested with the following inequality:

$$g_k^T - 1 g_k \geq 0.2 \left\| g_k \right\|^2$$

If this condition is satisfied, the search direction is reset to the negative of the gradient.

Scaled Conjugate Gradient [37, 38]

Each of the conjugate gradient algorithms discussed so far requires a line search at each iteration. This line search is computationally expensive, because it requires that the network response to all training inputs be computed several times for each search. The scaled conjugate gradient algorithm (SCG), developed by Moller[39], was designed to avoid the time-consuming line search. This algorithm combines the model-trust region approach (used in the Levenberg-Marquardt algorithm,), with the conjugate gradient approach.

Quasi-Newton Algorithms

There are two types of Quasi-New algorithms: Broyden, Fletcher, Goldfarb, and Shanno (BFGS) Alogrithm and One Step Secant Algorithm

Broyden,Fletcher,Goldfarb, and Shanno (BFGS) Algorithm [40, 41]

Newton's method is an alternative to the conjugate gradient methods for fast optimization. The basic step of Newton's method is

$$x_{k+1} = x_k - A_k^{-1} g_k$$

where A_k^{-1} is the Hessian matrix (second derivatives) of the performance index at the current values of the weights and biases. Newton's method often converges faster than conjugate gradient methods. Unfortunately, it is complex and expensive to compute the Hessian matrix for feedforward neural networks. There is a class of algorithms that is based on Newton's method, but which doesn't require calculation of second derivatives. These are called quasi-Newton (or secant) methods. They update an approximate

Hessian matrix at each iteration of the algorithm. The update is computed as a function of the gradient. The quasi-Newton method that has been most successful in published studies is the Broyden, Fletcher, Goldfarb, and Shanno (BFGS) update [42].

One Step Secant Algorithm [43]

Because the BFGS algorithm requires more storage and computation in each iteration than the conjugate gradient algorithms, there is need for a secant approximation with smaller storage and computation requirements. The one step secant (OSS) method is an attempt to bridge the gap between the conjugate gradient algorithms and the quasi-Newton (secant) algorithms. This algorithm does not store the complete Hessian matrix; it assumes that at each iteration, the previous Hessian was the identity matrix. This has the additional advantage that the new search direction can be calculated without computing a matrix inverse.

Levenberg-Marquardt Alogrithm [44]

Like the quasi-Newton methods, the Levenberg-Marquardt algorithm was designed to approach second-order training speed without having to compute the Hessian matrix. When the performance function has the form of a sum of squares (as is typical in training feedforward networks), then the Hessian matrix can be approximated as

$$H = J^T J$$

and the gradient can be computed as:

$$g = J^T e$$

where **J** is the Jacobian matrix that contains first derivatives of the network errors with respect to the weights and biases, and ***e*** is a vector of network errors. The Jacobian matrix can be computed through a standard backpropagation technique [45] that is much less complex than computing the Hessian matrix.

The Levenberg-Marquardt algorithm uses this approximation to the Hessian matrix in the following Newton-like update:

$$x_{k+1} = x_k - [J^T J + \mu I]^{-1} J^T e$$

When the scalar μ is zero, this is just Newton's method, using the approximate Hessian matrix. When μ is large, this becomes gradient descent with a small step size. Newton's method is faster and more accurate near an error minimum, so the aim is to shift toward Newton's method as quickly as possible. Thus, μ is decreased after each

successful step (reduction in performance function) and is increased only when a tentative step would increase the performance function. In this way, the performance function is always reduced at each iteration of the algorithm.

Therefore, we have total 7 different training algorithms. Depend on speed and performance of each algorithm, we will determine which one is the most suitable.

Speed and performance of each training algorithm

The speed and performance of each training algorithm depend on many factors, including the number of input vectors in the training set, the number of weights and biases in the network, the error goal and the network is being used for cartilage recognition. Training algorithm, which provides the best performance, will be selected. Hence, we have taken several experiments about those different training algorithms to investigate their speed and performance.

We made experiments on Matlab environment by using Matlab software. Here are some computer's features that we used:

- Computer: Intel(R), Pentium(R) 4CPU 2.66GHz; 512 MB of RAM
- Operation system: Microsoft Windows XP Professional, Version 2002, Service Pack 2

First Experiment

We made a first experiment to get a general idea about speed and performance of each training algorithm.

- Training data:
  + 200 input vectors that denote cartilage class, include 100 input vectors of pixels on cartilage region and 100 input vectors of pixels on cartilage boundaries.
  + 200 input vectors that denote background class.

  Training data are randomly selected from a sample image shown in Fig 4.9
- Neural Network: Two layer perceptrons with log-sigmoid activation function in each layer. Hidden layer has 40 neurons whereas only one neuron on output layer.

Figure 4.9 Original image in which training data collected.

The following table is the results of training the network using seven different training algorithms that are mentioned in previous section in terms of time and mean square error (MSE). We only take one trial for each training algorithm.

- The time shown in the table is the total time for the computer to compute all the weights and biases of a neural network by using a specific training algorithm to minimize the mean square error (MSE) between network output and target output. The speed of training a network is faster when the time is shorter.

- The mean square error (MSE) shown in the table is the mean square error between network outputs and target outputs. Performance of a network is determined by MSE value. Network performance is higher when MSE value is smaller.

Algorithms	**Time (s)**	**MSE**
Polak-Ribiere Conjugate Gradient	1.62815	0.13596
Fletcher-Powell Conjugate Gradient	1.43912	0.13024
Powell/Beale Restarts	2.12743	0.15211
Scaled Conjugate Gradient	4.58572	0.12199
BFGS Quasi-Newton	240	0.14801
One Step Secant	3.36133	0.15015
Levenberg-Marquardt	210	0.16421

Table 4.1 Speed and performance of seven different training algorithms

Overall, Levenberg-Marquardt and BFGS Quasi-Newton methods do not perform well on object recognition problems. The time for computing all network's weights and biases by using those training algorithms is significant longer. The Levenberg-Marquardt is designed for least squares problems that are approximately linear. Because the output neurons in object recognition problems are generally saturated, it should not be operated in the linear region.

As a result, we can use five training algorithms such as Polak-Ribiere, Fletcher-Powell, Scaled Conjugate Gradient, Powell/Beale Restarts and One Step Secant methods.

Second Experiment

In second experiment, we processed intensely in order to choose the most suitable training methods.

- Training data and neural network that are used in second experiment are the same as in the first experiment.

- For each training algorithm, we made 30 different trials where different random initial weights and biases are used in each trial.

The following table summarized the results of training a network with five different algorithms. During 30 different trials, each entry in the table represents:

- Min Time: is the shortest time for computing all network's weights and biases
- Max Time: is the longest time for computing all network's weights and biases
- Mean Time: is the average time for computing all network's weights and biases.
- Min MSE: is the lowest mean square error.
- Max MSE: is the highest mean square error.
- Mean MSE: is the average mean square error.
- Number of successful trials: that we are successful in computing all network's weights and biases to minimize mean square error under condition that gradient value is less than 0.00001.

	Polak-Ribiere	Fletcher-Powell	Powell/Beale Restarts	Scaled Conjugate Gradient	One Step Secant
Min Time (s)	1.2161	1.33151	1.11611	2.82169	1.09089
Max Time (s)	3.2315	2.59043	2.65109	4.58572	3.71076
Mean Time (s)	1.67	1.7101	1.5836	3.654	1.7723
Min MSE	0.11771	0.12158	0.12309	0.11809	0.13673
Max MSE	0.15119	0.14373	0.15211	0.14068	0.17612
Mean MSE	0.133507	0.130638	0.132037	0.13056	0.15309
Number of successful trials	23	13	21	30	29

Table 4.2 Speed and performance of five different training algorithms

From the table, Scaled Conjugate Gradient is the most suitable training method. It not only produces lowest MSE but also provides consistency and reliability (Scaled Conjugate Gradient method has highest successful rate (30/30)). Therefore, we will select Scaled Conjugate Gradient method for training a network.

4.6 Application of Neural Network Classifier

When we success in creating a NNC, we apply it as cartilage recognition to extract a cartilage from an original image. Because NNC can specify a class of a pixel on an image, one obvious simple way is to apply NNC to all pixels. However, it is not effective and reliable according to noise. Similar to BSSM, there are also two processes on our NNCM: left and right process. The left process is to extract the cartilage on partitioned sub-images on the left direction according to initial sub-image. On the other hand, right process is to extract the cartilage on partitioned sub-images on the right direction according to initial sub-image.

However, because NNC can classify a pixel as cartilage or background, it cannot classify cartilage pixel into cartilage component's class. We apply NNCM to obtain a cartilage instead of individual cartilage component from an input image. A general flowchart of NNCM is shown as Fig 4.10 (a).

Consider an input image is a matrix ***I*** that is defined from pre-processing step. Sub-images are columns of matrix ***I.*** Sub-images that contain the cartilage segments are defined as cartilage sub-images. Our NNCM algorithms are described as:

1. From input image [K rows and L columns] $\boldsymbol{I}$, find initial cartilage sub-image representing $i^{\text{-th}}$ column of matrix $\boldsymbol{I}$, where i = 1, 2, …, L. Initial cartilage sub-image determination is described in section 4.6.1.
2. Left Process: Corresponding to initial sub-image, find a next cartilage sub-image representing $(i + \Delta)^{\text{-th}}$ column, Δ = 1,2,3… Cartilage sub-image determination is described in section 4.6.1.
3. Right Process: Corresponding to initial sub-image, find a previous cartilage sub-image representing $(i - \Delta)^{\text{-th}}$ column, Δ = 1,2,3…
4. For each pixel on cartilage sub-images that found in step 1, 2, and 3 , compute output of a network classifier:

$$a^1 = \log sig(iw_{s^1,R} + b^1)$$

$$a^2 = \log sig(lw_{S^2,S^1}.a^1 + b^2)$$

Where a^2 is output of a network; $\boldsymbol{iw}$ and $\boldsymbol{b^1}$ are weights and biases between input and hidden layer; $\boldsymbol{lw}$ and $\boldsymbol{b^2}$ are weights and biases between hidden layer and output layer.

R is number of elements of a input vector; S^1 is number of neurons of hidden layer; and S^2 is number of neurons of output layer.

Output network is then filtered with a threshold $\boldsymbol{K}$

- Output value is "1" if $a^2 > K$: this pixel is classified as cartilage pixel.
- Output value is "0" if $a^2 < K$: this pixel is classified as background pixel.

5. Continue until all cartilage pixels are classified on cartilage sub-images.

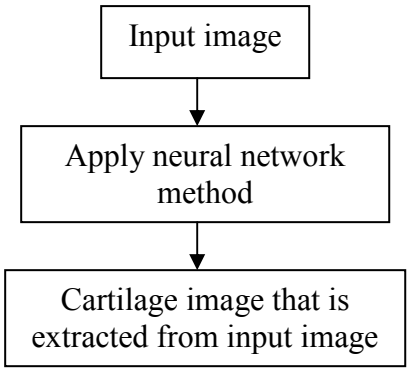


Figure 4.10(a) General flowchart of NNCM

Fig 4.10(b) illustrates operation of NNCM.

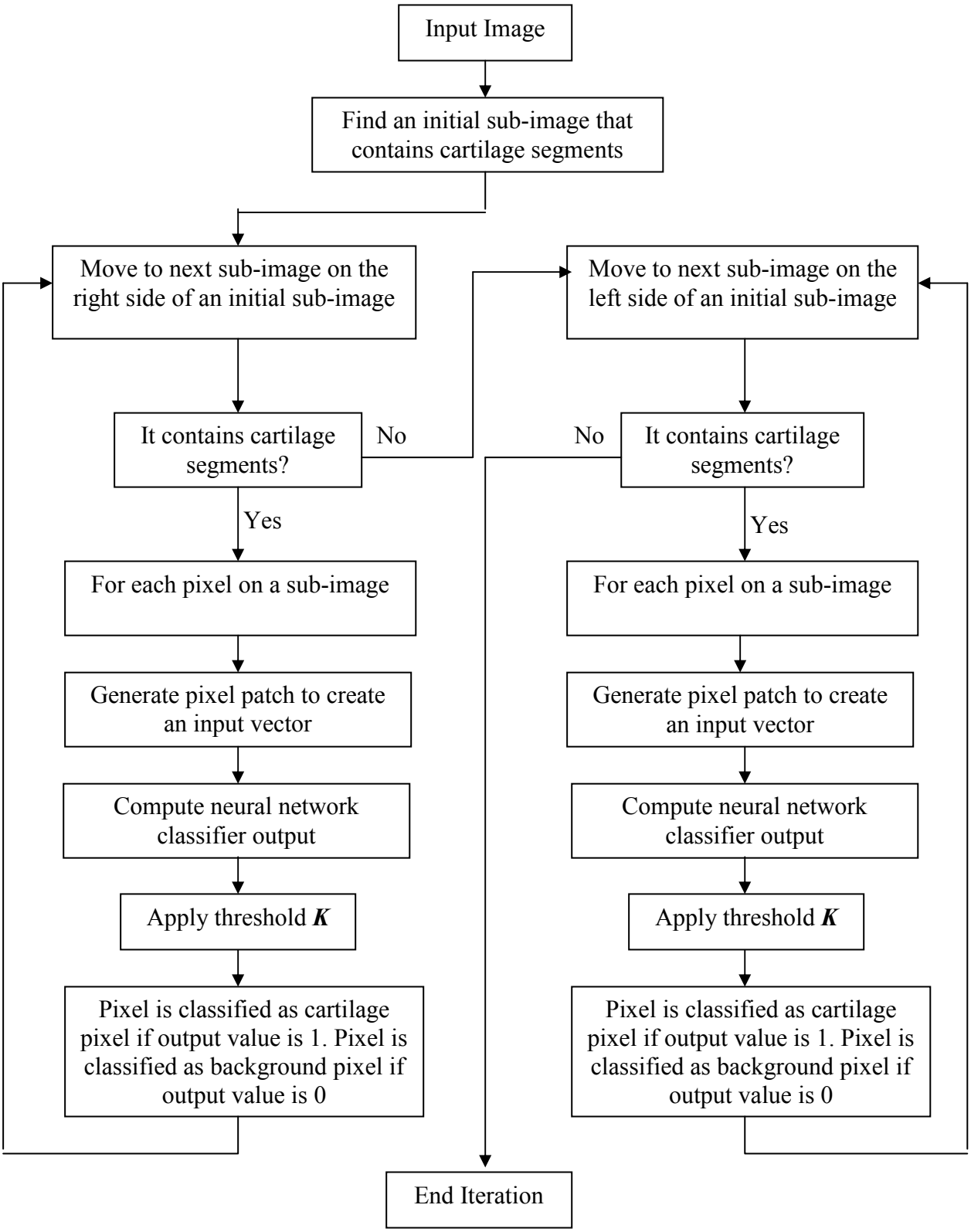


Figure 4.10 Operation of NNCM.

4.6.1 Cartilage sub-image and initial cartilage sub-image determination

Unlikely cartilage sub-image determination of BSSM, which is based on the group of pixels that have high intensity than a reference threshold value V (see Chapter 3), we apply NNC on each pixel on a sub-image to make determination.

A sub-image $f(x,y)$ is defined as
- Cartilage sub-image if $g(x,y) \in$ Cartilage class
- Otherwise, Background sub-image

Where $g(x,y)$ is a group of pixels on a sub-image that are classified as cartilage class by a NNC.

An initial cartilage sub-image is a cartilage sub-image, which locates in the middle of a cartilage region as well as in the middle of an input image.

4.7 Experiment Results

This section presents the visual results obtained by using NNCM to extract a cartilage from images sets. The quatitative evaluation of this method is then described in Chapter 6 (Area and Volume Calculation). We use the same image sets that are used in case of BSSM in order to test and make comparison between BSSM and NNCM.

- Image set 1: Images that contain Femur, Tibia, and Patella.
- Image set 2: Images that contain Femur, and Tibia.
- Image set 3: Images that contain Femur and Patella.

Image Set 1

Fig 4.11 shows cartilage images extracted from Image Set 1.

Image Set 2

Fig 4.12 shows cartilage images extracted from Image Set 2.

Image Set 3

Fig 4.13 shows cartilage images extracted from Image Set 3.

Results obtained by using NNCM appeared more accurate compared to the results obtained by using BSSM. In both cases, the cartilage images were compared with the original cartilage image. It was observed that NNCM worked well with several images that BSSM could not work well. It can be observed in the following images as follows:

- Image 2, image 6 on Image set 1 in Fig 4.11
- Image 6, image 8 on Image set 2 in Fig 4.12

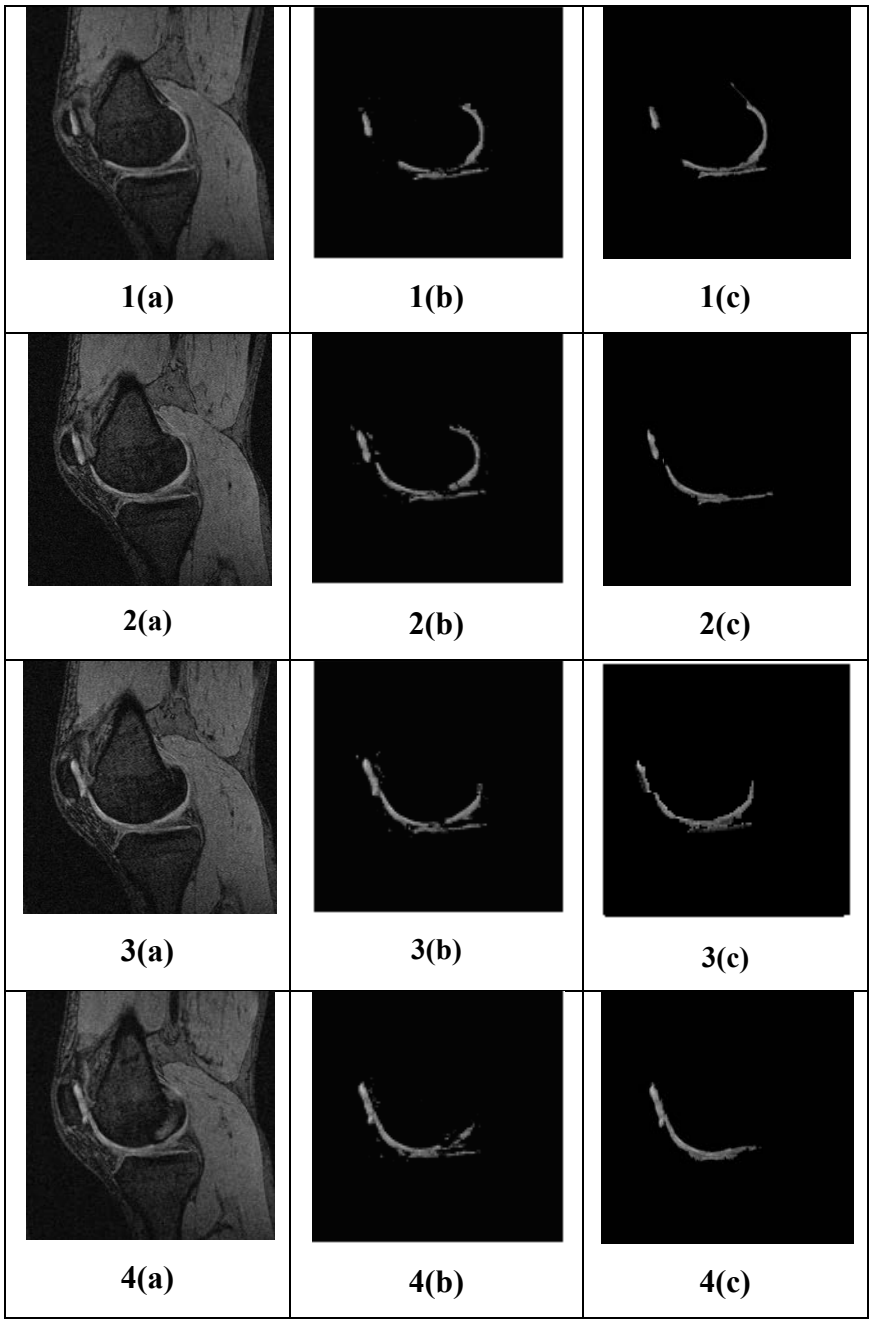


Figure 4.11 (a) Images on Image Set 1, Cartilage images extracted from Image Set 1 by using NNCM (b) and BSSM (c).

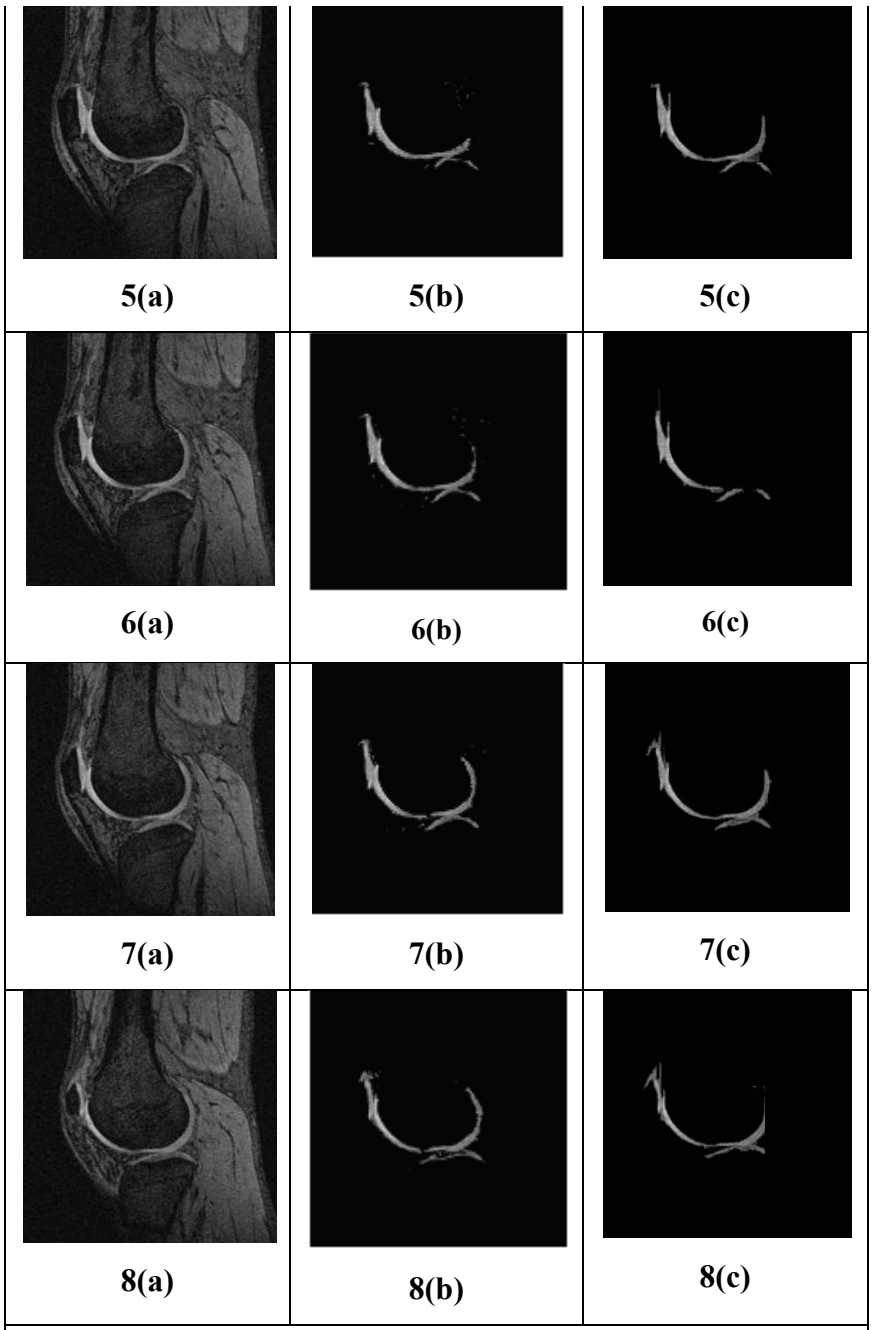


Figure 4.11 (a) Images on Image Set 1, Cartilage images extracted from Image Set 1 by using NNCM (b) and BSSM (c).

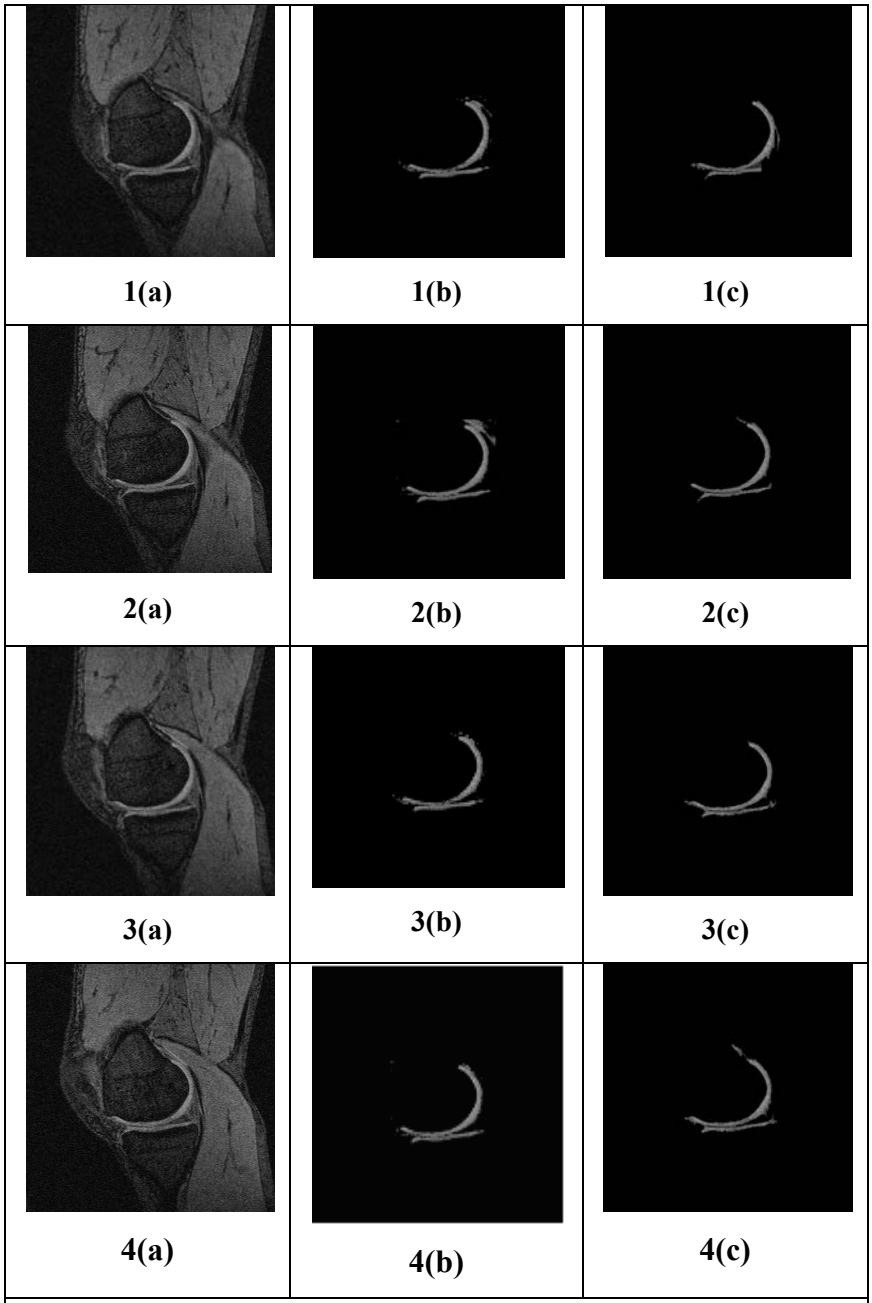


Figure 4.12 (a) Images on Image Set 2, Cartilage images extracted from Image Set 1 by using NNCM (b) and BSSM(c).

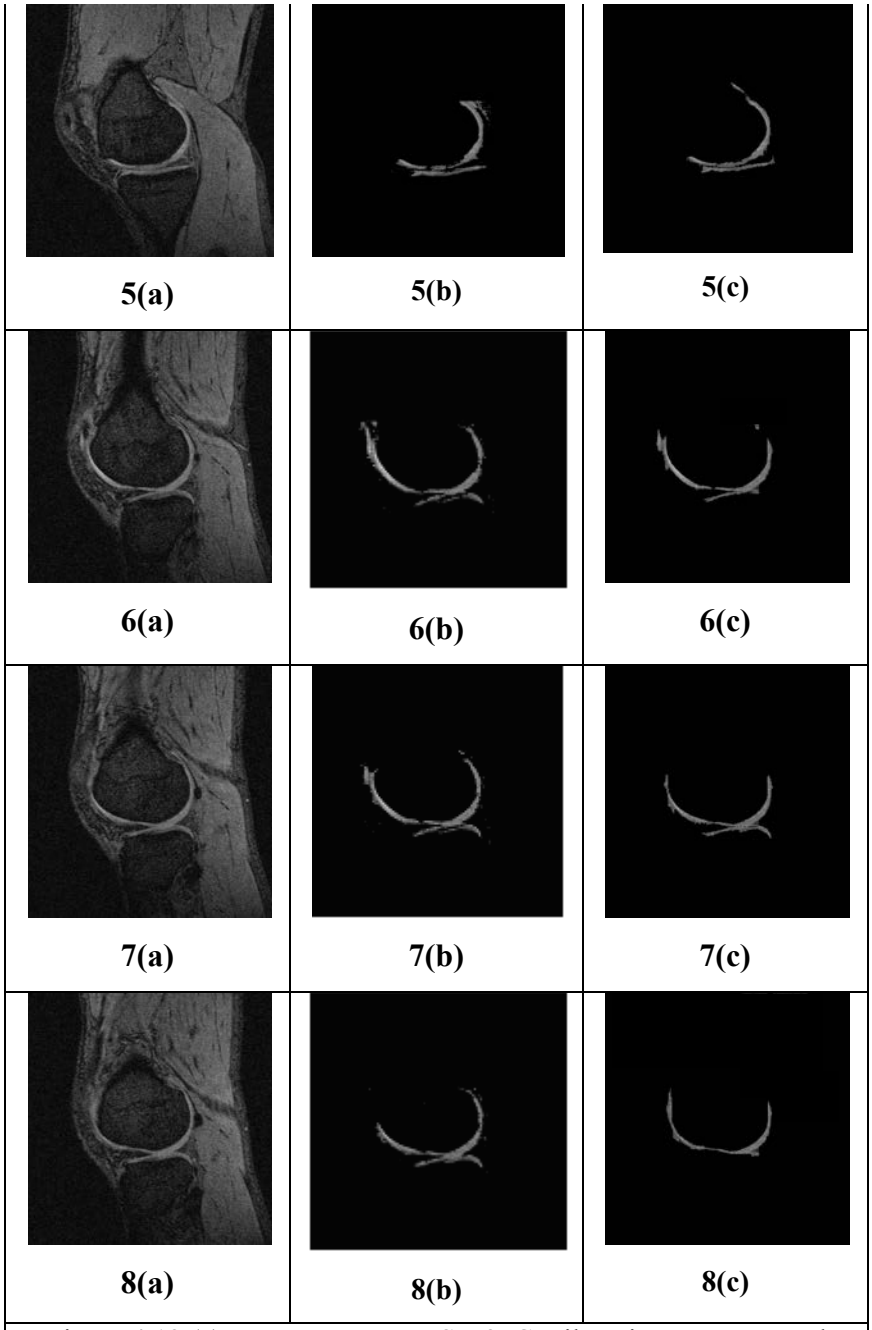


Figure 4.12 (a) Images on Image Set 2, Cartilage images extracted from Image Set 1 by using NNCM (b) and BSSM (c).

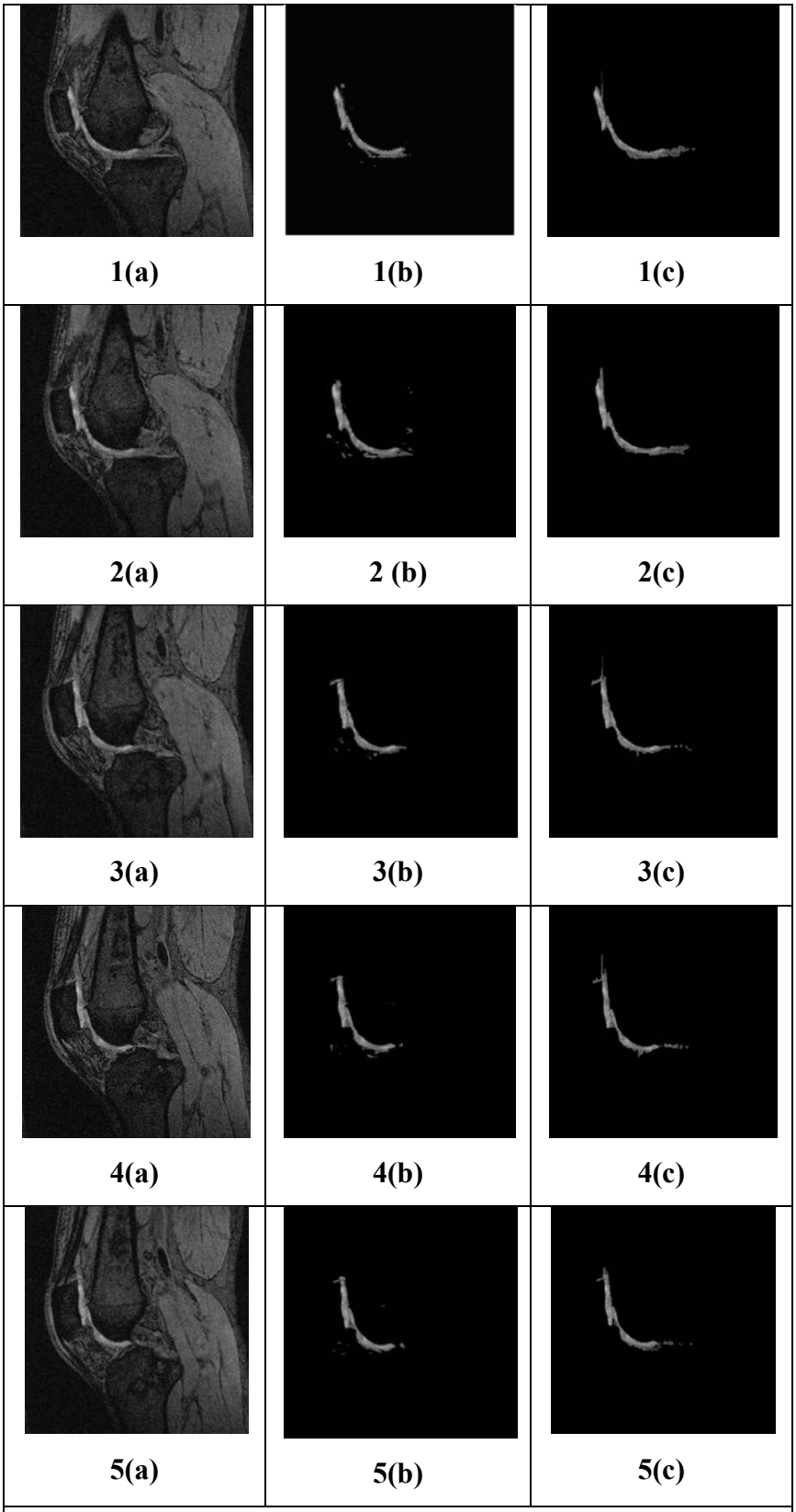


Figure 4.13(a) Images on Image Set 3, Cartilage images extracted from Image Set 1 by using NNCM (b) and BSSM (c).

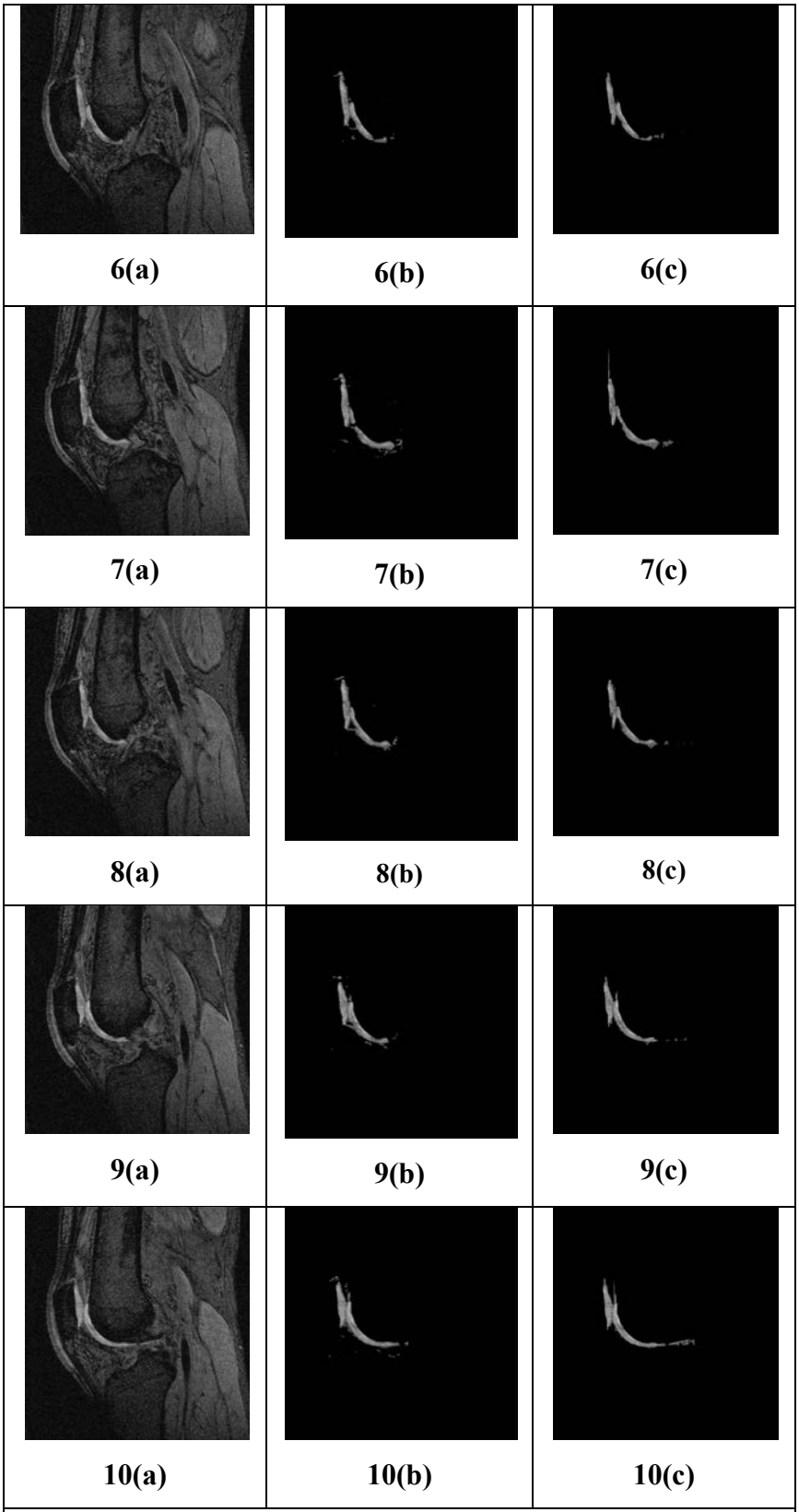


Figure 4.13(a) Images on Image Set 3, Cartilage images extracted from Image Set 1 by using NNCM (b) and BSSM (c).

4.8 Conclusion

Multilayer Perceptrons (MLPs) are one of the most important types of neural network that can be used for object classification in image processing. The success of classification depends on training the sample data with learning rules and training algorithms. Therefore, NNCM can work well when there is low contrast between object and background regions that cause difficulty for BSSM. Fig 4.14 is an example of this.

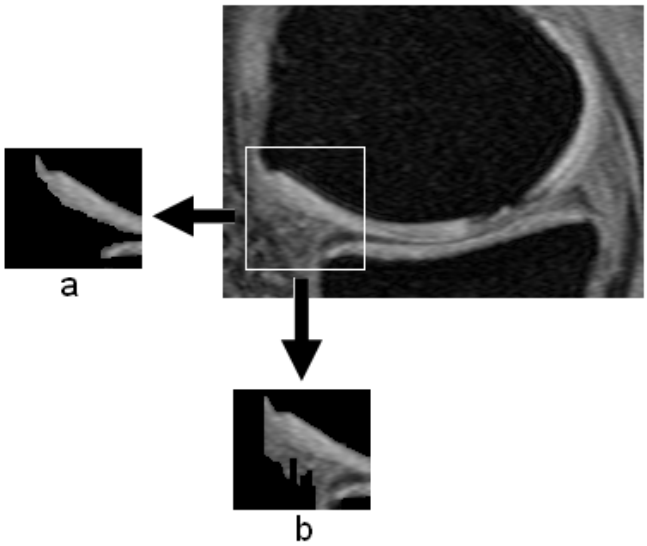


Figure 4.14 Comparison of NNCM and BSSM according to low contrast between cartilage and background regions

(a) Result of extracting cartilage by using NNCM.

(b) Result of extracting cartilage by using BSSM

However, NNCM may also have drawbacks. When cartilage pixels have features similar to background pixels, NNCM is likely to consider them as background pixels. Therefore, it is recommended to decrease the size of a cartilage or its number. Fig 4.15 illustrates this.

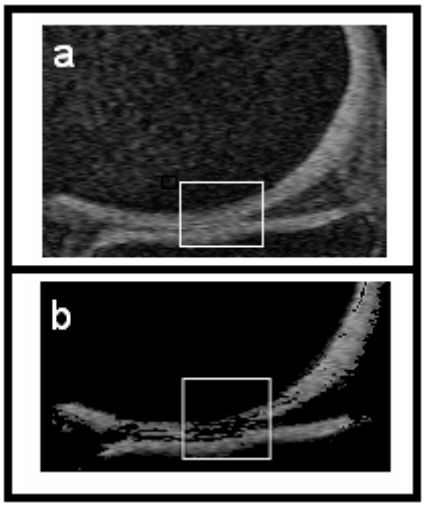


Figure 4.15 Example of NNCM when object pixels are similar to background pixels. (a) Original MR Image and (b) Result obtained by using NNCM.

Therefore, a final method that can takes advantages of both BSSM and NNCM is developed. It is based on active contour models, BSSM and NNCM. This method named active contour models method (ACMM) will be introduced in next chapter.

Chapter 5

ACTIVE CONTOUR MODELS

5.1 Overview

Active contours was first introduced by Kass et al.1988 [46]. An active contour is a set of points to enclose a target feature to be extracted. It is a bit like using a balloon to 'find' a shape: the balloon is placed outside (or inside) the shape, enclosing it. Then by taking air out (or in) of the balloon, making it smaller (or bigger), the shape is found when the balloon stops shrinking (or expanding), when it fits the target shape. By this manner, active contours arrange a set of points so as to describe a target feature by enclosing it.

Give an approximation of the boundary of an object in image; an active contour model can be used to find the 'actual' boundary. Active contour models should be able to find the boundary in MR images of cartilage when an initial guess is provided by a user or by some other method, possibly an automated one.

An active contour is an ordered collection of *n* points in the image plane:

$$V = \{v_1, v_2, ..., v_n\}$$
$$v_i = \{x_i, y_i\}, i = \{1,2,3,...n\}$$

where *n* is number of pixels that are supposed to initial contour of the object.

The points in the contour iteratively approach the boundary of an object through the solution of an energy minimization problem. For each point in the neighbourhood of υ_i, an energy term is computed:

$$E_i = \alpha E_{\text{int}}(\upsilon_i) + \beta E_{ext}(\upsilon_i)$$

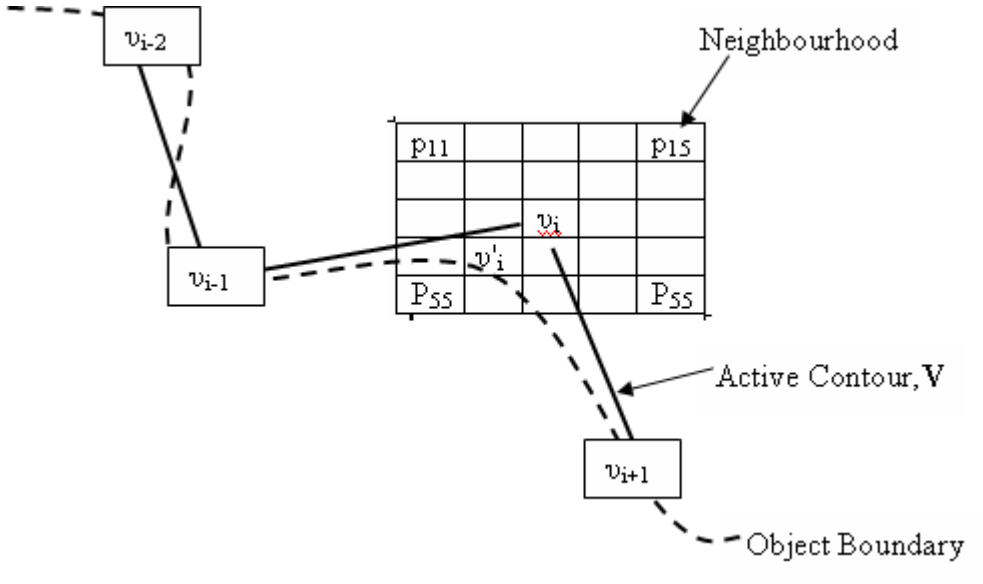


Figure 5.1. An example of the movement of a point υ_i, in an active contour. The point υ'_i, is the location of minimum energy.

where $E_{int}(\upsilon_i)$ is an energy function dependent on the shape of the contour and $E_{ext}(\upsilon_i)$ is an energy function dependent on the image properties near point υ_i. α and β are constants providing the relative weighting of the energy terms.

E_i, E_{int}, E_{ext} are matrices. The value at the center of each matrix corresponds to the contour energy at point υ_i. Other values in the matrices correspond (spatially) to the energy at each point in the neighbourhood of υ_i.

Each point, υ_i, is moved to the point υ'_i, corresponding to the location of the minimum value in E_i. This process is illustrated in Fig. If the energy functions are chosen correctly, the contour V, should approach, and stop at, the object boundary.

In this chapter, we present a method named active contour models method (ACMM) that is mainly based on active contour models to extract the articular cartilage from original MR knee image [1]. This method has also a combination of two previous introduced methods: BSSM and NNCM. BSSM is used for defining an initial active contour and NNCM is used for computing the external energy.

5.2 Energy Formulation

5.2.1 Internal Energy

The internal energy function is defined to enforce a shape on the deformable contour and to maintain a constant distance between the points in the contour. Additional terms can be added to influence the motion of the contour.

The internal energy function used herein is defined as follows:

$$\alpha E_{\text{int}}(\upsilon_i) = cE_{con}(\upsilon_i) + bE_{bal}(\upsilon_i)$$

where $E_{\text{con}}(\upsilon_i)$ is the continuity energy that enforces the shape of the contour and $E_{\text{bal}}(\upsilon_i)$ is a balloon force that causes the contour to grow (balloon) or shrink. c and b provide the relative weighting of the energy terms.

Continuity Energy

In the absence of other influences, the continuity energy term causes an open deformable contour into a straight line and a closed deformable contour into a circle. The formulation of the continuity energy has been adopted from [47]. The energy term for each element, $e_{\text{jk}}(\upsilon_i)$, in the matrix, $E_{\text{con}}(\upsilon_i)$ is defined as follows:

$$e_{jk}(\upsilon_i) = \frac{1}{l(V)}\left\| p_{jk}(\upsilon_i) - \gamma(\upsilon_{i-1} + \upsilon_{i+1}) \right\|^2$$

where $p_{\text{jk}}(\upsilon_i)$ is the point in the image that corresponds spatially to energy matrix element $e_{\text{jk}}(\upsilon_i)$.

$\gamma = 0.5$ for an open contour. In this case, the minimum energy point is the point exactly half way between υ_{i-1} and υ_{i+1}.

For the case of a closed contour, V is given a modulus of n. Therefore, $\upsilon_{n+i} = \upsilon_i$. γ is then defined as follows:

$$\gamma = \frac{1}{2\cos\left(\frac{2\pi}{n}\right)}$$

Here, the point of minimum energy of $E_{\text{con}}(\upsilon_i)$ is pushed outward so that V become a circle. This behaviour is illustrated in Fig 5.2.

The normalization factor, l(V), is the average distance between points in V:

$$l(V) = \frac{1}{n}\sum_{i=1}^{n}\left\|\upsilon_{i+1} - \upsilon_i\right\|^2$$

Magnitudes have been left squared to reduce the computation load. The normalization is required to make $E_{con}(\upsilon_i)$ independent of the size, location, and orientation of V.

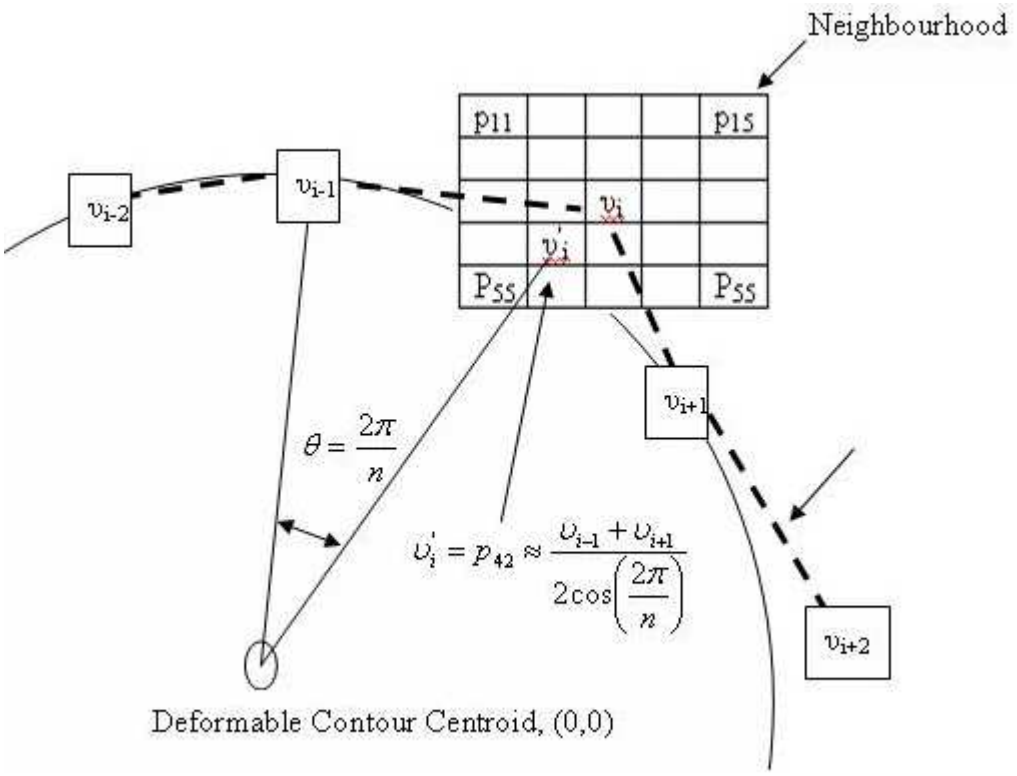


Figure 5.2 An example of the movement of a point in an active contour due to continuity energy. The point υ'_i, is the location of minimum energy because it lies on the circle connecting υ_{i-1} and υ_{i+1}.

Balloon Force

A balloon force can be used on a closed deformable contour to force the contour to expand (or shrink) in the absence of external influences. A contour initialized within a uniform image object will expand under the influence of a balloon force until it nears the object boundary (at which point the external energy function affects its motion). Fig 5.3 illustrates this behaviour.

Chalana et al. 1995 suggests an adaptive balloon force that varies inversely proportionally to the image gradient magnitude [[48]]. The adaptive balloon force is strong in homogeneous regions and weak near object boundaries, edges, and lines.

The energy term for element, $e_{jk}(\upsilon_i)$, in the matrix, $E_{bal}(\upsilon_i)$ is expressed as a dot product:

$$e_{jk}(\upsilon_i) = n_i \bullet \left(\upsilon_i - p_{jk}(\upsilon_i)\right)$$

Where n_i is the outward unit normal of V at point υ_i and $p_{jk}(\upsilon_i)$ is the point in the neighbourhood of υ_i corresponding to entry $e_{jk}(\upsilon_i)$ in the energy matrix. Therefore, the balloon energy is smallest at points farest from υ_i in the direction of n_i.

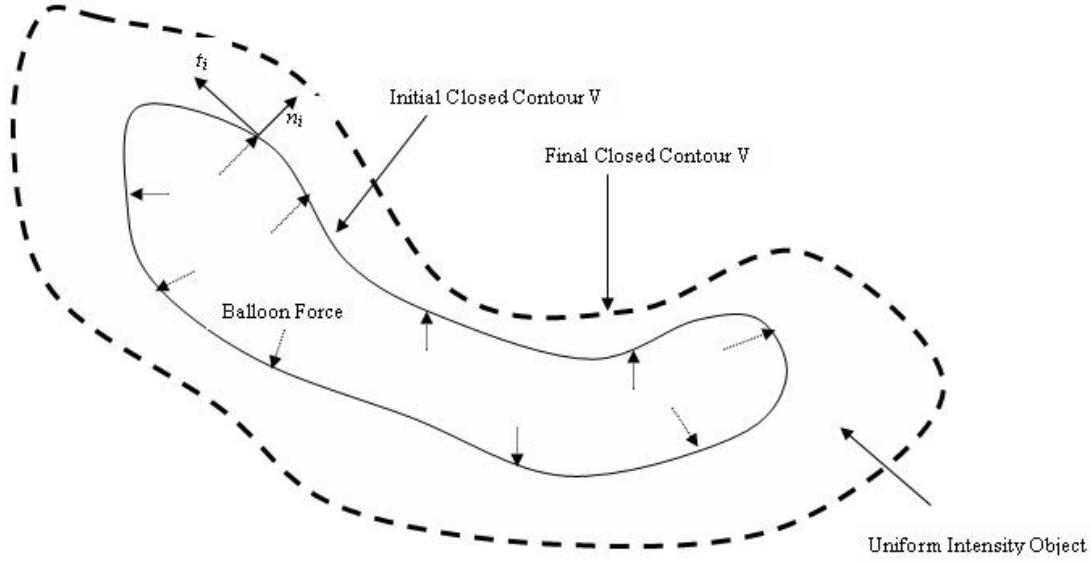


Figure 5.3 An example of the movement of a deformable contour due to balloon energy. Because the object has uniform intensity, a balloon force is required to push the contour toward the object boundary.

n_i can be found by rotating the tangent vector, t_i, by 90^o. t_i is easily computed:

$$t_i = \frac{\upsilon_i - \upsilon_{i-1}}{\|\upsilon_i - \upsilon_{i-1}\|} + \frac{\upsilon_{i+1} - \upsilon_i}{\|\upsilon_{i+1} - \upsilon_i\|}$$

So n_i is a unit vector normal to t_i.

5.2.2 External Energy

The external energy function attracts the deformable contour to interesting features in an image. Traditional external energy is looked at image gradient and intensity. For example, we can use BSSM, which is based on intensity as external energy. However, it is clearly see that this method has difficulty when there are low contrast between cartilage pixels and background pixels. This problem is previously illustrated in Chapter 3. Hence, because NNC (chapter 4) is good at handling this problem, we then apply a NNC for computing the external energy. Moreover, in order to attract the deformable contour, the NNC is more advanced than BSSM. Comparison between NNC and BSSM is demonstrated in section 4.7 Chapter 4. In additional, although using

NN is computationally expensive (in both training stage and in testing stage), in this thesis's situation, the speed is not critical. In return, we can get greater benefit by using NN since NN is a lead toward other more advanced method.

For each point in the neighbourhood of υ_i, an external energy is computed:

$$E_{ext}(\upsilon_i) = E_{NN}(\upsilon_i)$$

where $E_{NN}(\upsilon_i)$ is NNC output value. The NNC can be adopted from Chapter 4, which is specified as:

- Neural network classifier (NNC): two-layer perceptrons with log-sigmoid activation function in each layer, 40 neurons on hidden layer and 1 neuron on output layer.
- Output of a network is computed as:

$$a^1 = \log sig(iw_{s^1,R} + b^1)$$

$$a^2 = \log sig(lw_{S^2,S^1}.a^1 + b^2)$$

Where a^2 is output of a network; $\boldsymbol{iw}$ and $\boldsymbol{b^1}$ are weights and biases between input and hidden layer; $\boldsymbol{lw}$ and $\boldsymbol{b^2}$ are weights and biases between hidden layer and output layer.

R is number of elements of a input vector; S^1 is number of neurons of hidden layer; and S^2 is number of neurons of output layer.

Diagram of NNC algorithm is described as Fig 5.4:

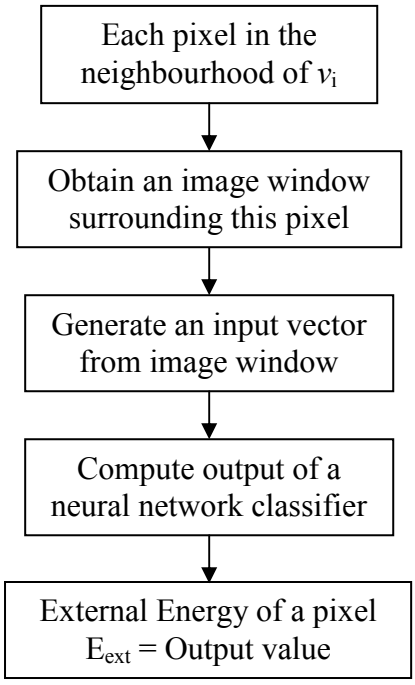


Figure 5.4 Diagram of External Energy Computation by using NNC

Each point, υ_i, through the neural network, provides a value from [0 to 1] in which 0 indicates that point belongs to background (other tissues) and 1 indicates that point belongs to object (cartilage). Point υ_i, is moved to the point υ'_i, corresponding to the location of the minimum value E_{NN}.

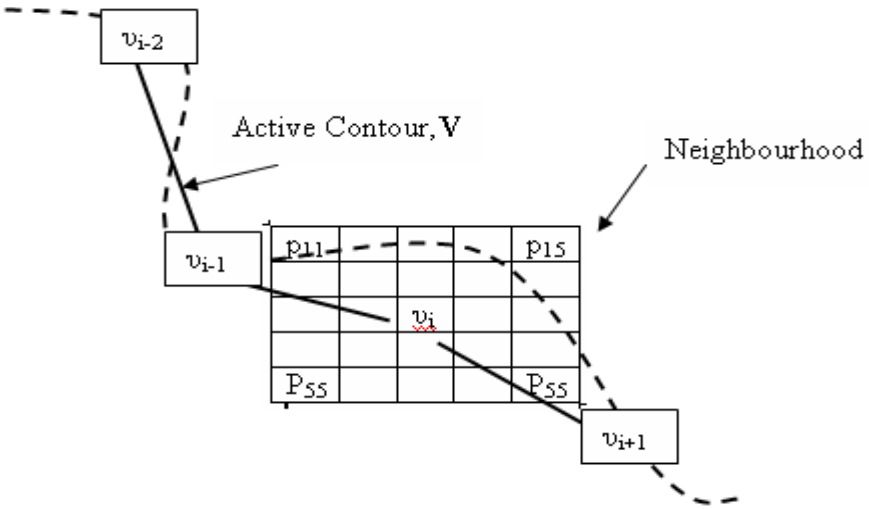


Figure 5.5 An example of the movement of a deformable contour due to neural network energy. The point, $\upsilon'_i = p_{15}$, is the location of minimum energy because it lies on background.

5.3 Regularization

The energy functions introduced in the previous sections should be scaled so that the neighbourhood matrices contain comparable values. This process is referred to as regularization. Here, each of the energy functions is adjusted to the range [0, 1].

The balloon energy is further modified to adapt to the image gradient magnitude. Regularization is not required for external energy function because neural network energy function already provide value in the range [0, 1].

Continuity Energy

At each point in the deformable contour, the elements in neighbourhood matrix for the continuity energy are simply scaled to the range [0, 1]:

$$e'_{jk}(\upsilon_i) = \frac{e_{jk}(\upsilon_i) - e_{\min}(\upsilon_{jk})}{e_{\max}(\upsilon_i) - e_{jk}(\upsilon_i)}$$

Where $e_{min}(\upsilon_i)$ and $e_{max}(\upsilon_i)$ are the minimum and maximum valued elements, respectively, in $E_{con}(\upsilon_i)$.

Balloon Energy

The balloon energy is scaled to the range [0, 1], then adapted to the image gradient intensity:

$$e'_{jk}(\upsilon_i) = \frac{e_{jk}(\upsilon_i) - e_{\min}(\upsilon_i)}{e_{\max}(\upsilon_i) - e_{jk}(\upsilon_i)} \cdot \left(1 - \frac{|\nabla I(\upsilon_i)|}{|\nabla I|_{\max}}\right)$$

Where $|\nabla I|_{\max}$ is the maximum gradient magnitude in the entire image

5.4 Application of Active Contour Models Algorithms

Similar to BSSM, ACMM algorithms tend to extract individual cartilage components (femur, tibia, and patella) from an original image. Therefore, our algorithms also include three processes:

- First process: we apply ACMM on original image with initial contour of femur to obtain femur image (Refer to Fig 5.6).
- Second process: we apply ACMM on an image that femur was extracted with initial contour of tibia to obtain tibia image (Refer to Fig 5.6).
- Final process: we apply ACMM on an image that femur and tibia were extracted with initial contour of patella to obtain patella image (Refer to Fig 5.6).

Fig 5.6 illustrates flowchart of ACMM algorithms:

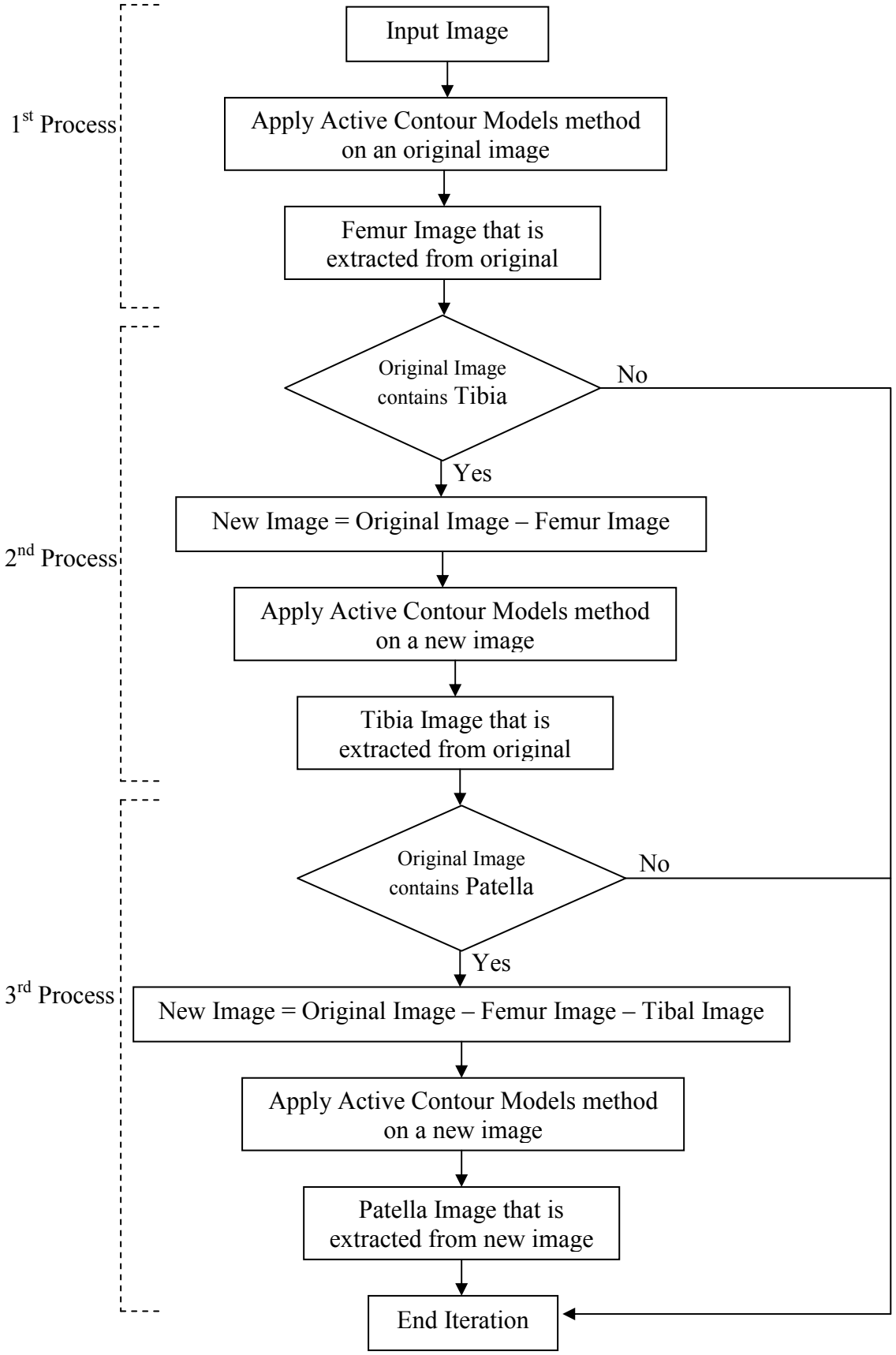


Figure 5.6 Flowchart of using ACMM to extract a cartilage from an original image.

Fig 5.7 illustrates ACMM to extract an individual cartilage component from input image.

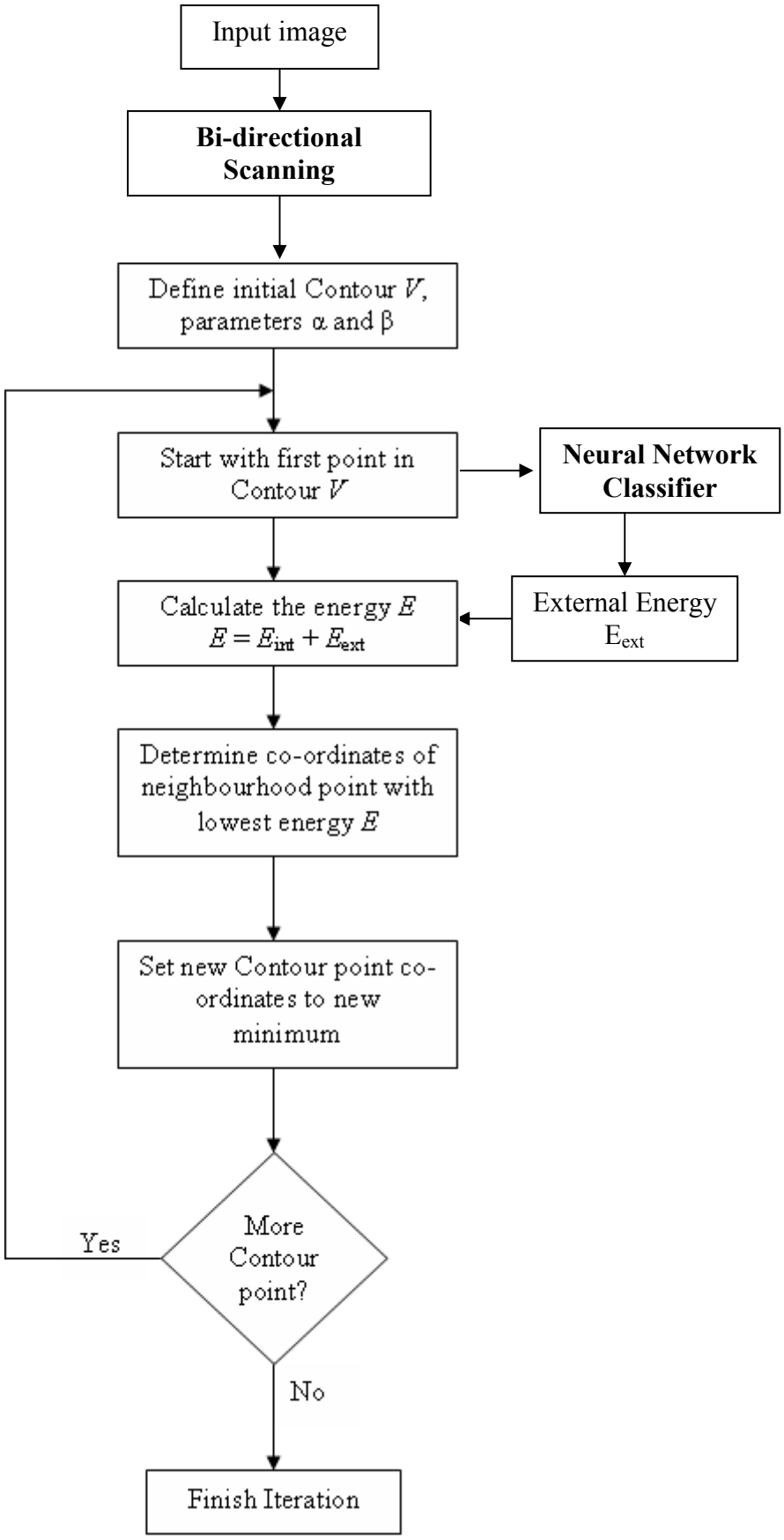


Figure 5.7 Operation of ACMM

In this implementation, an initial contour *V* can be provided by a user, or by some method, possibly an automated one. In our case, we apply BSSM (chapter 3) as an automated method to define an initial contour *V*.

The contour *V* is stored as a matrix of vectors. Each vector has five elements: *x* and *y* co-ordinates of the contour point, the values of α, β, and the flag for that contour point. α and β are constants providing the relative weighting of the energy terms. The flag provides value 0, which indicates the minimum energy at that contour point, and value 1, which indicates the minimum energy, is not at that contour point. The iteration will finish when the flags of all contour point are set to 0 which mean the contour V is enclosed the cartilage boundaries

5.4.1 Initial Contour Definition

As mentioned in Chapter 3, we can use BSSM to find cartilage boundaries on an image. Hence, we can apply BSSM to define an initial contour as an automatic method. Initial contour definition algorithms are described as:

1. Apply BSSM on input image.
2. For each cartilage sub-image, we obtain cartilage boundaries according to *(x,y)* location.
3. Those cartilage boundaries will be considered as initial contour points.
4. Continue until all cartilage boundaries that are detected by BSSM are considered as initial contour points.

5.5 Experiment Results

Discussion on initial contour

The first important part in contour algorithm is the quality of the initial contour. The contour algorithm improves when the image contains little noise and the contrast between the cartilage parts and other tissues is significant. When the condition of image is bad: noise and the contrast between the cartilage parts and other tissues is low, the contour algorithm is likely to attach itself to noise and other tissues boundaries.

Fig 5.8 shows that initial contour used to obtain actual boundary of patella segments is not good because it also contain femur segments. Hence, the calculated patella contour also contains femur segments. The initial patella contour is difference in Fig 5.9. In this case, the obtained patella contour is reasonable.

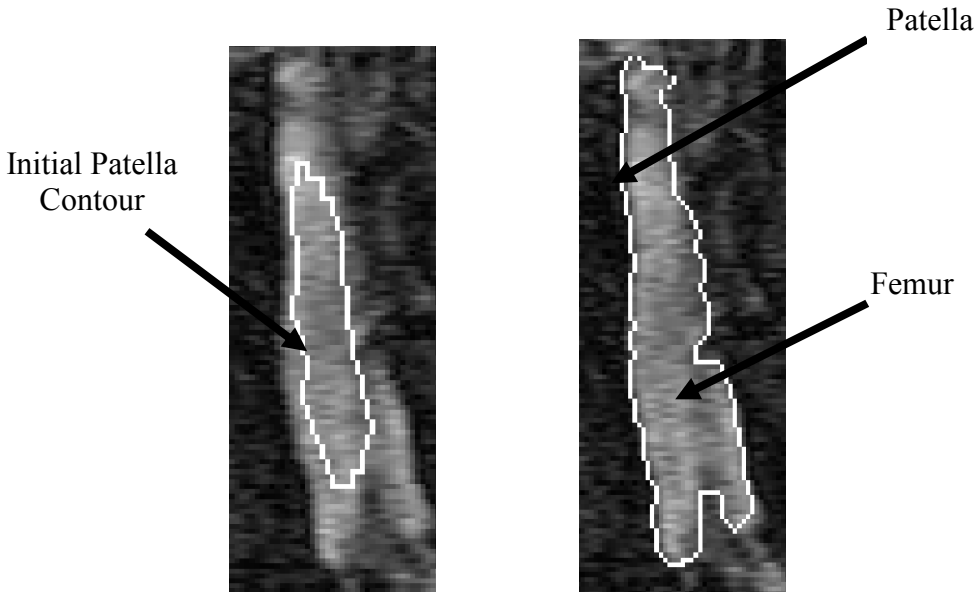


Figure 5.8 An example of behaviour of contour algorithm when initial contour is not good. (a) The initial patella contour and (b) The final computed patella contour

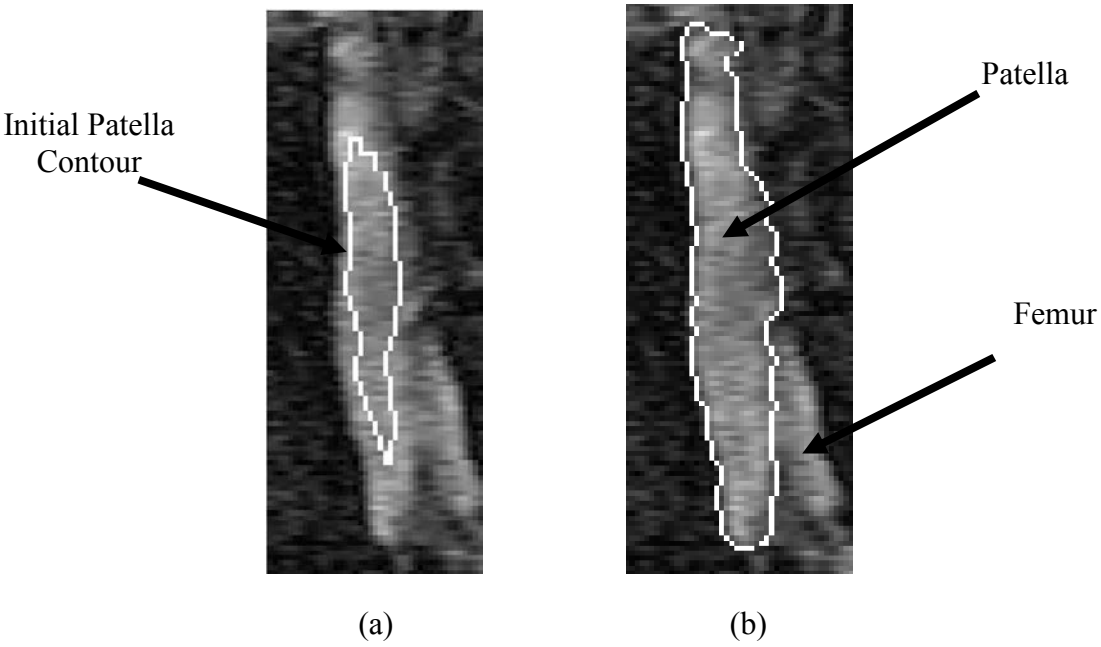
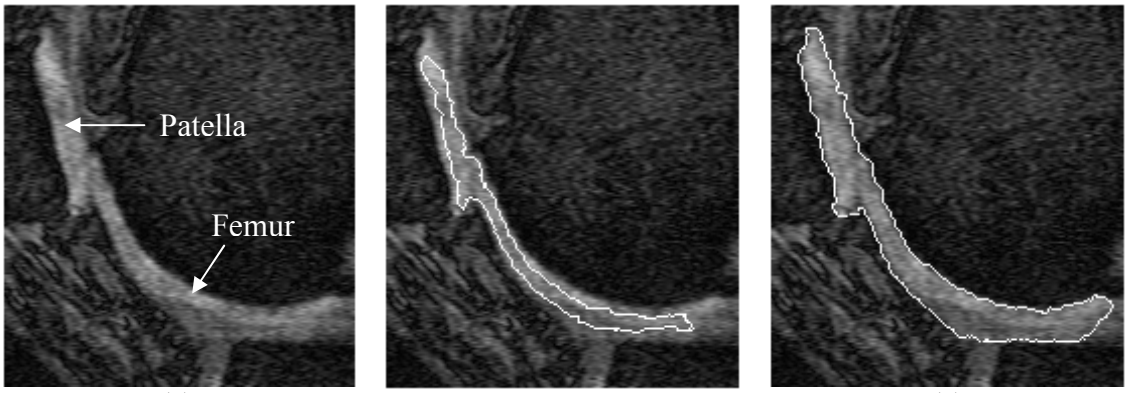


Figure 5.9 An example of behaviour of contour algorithm when initial contour is good. (a) The initial patella contour and (b) The final computed patella contour.

Because the initial contour V is defined from applying BSSM, initial contour V depends on output of this approach. For case BSSM cannot detect femur boundaries when femur and patella are connected, femur boundaries are also include patella. Therefore, initial femur contour V is then defined including patella. The final femur contour is then computed also contain patella. Fig 5.10 illustrates this case.



(a) (b) (c)

Figure 5.10 (a) Original image (b) Initial femur contour V defined from applying BSSM (c) Final femur contour is computed by using ACMM.

Experiment Results

This section presents the visual results obtained by using ACMM to extract cartilage from image sets. The quatitative evaluation of this method will be demonstrated in Chapter 6 (Area and Volume Calculation). We apply ACMM to the same image sets that were used in case of BSSM and NNCM in order to test and make comparison among three methods.

- Image set 1: Images that contain Femur, Tibia, and Patella.
- Image set 2: Images that contain Femur, and Tibia.
- Image set 3: Images that contain Femur and Patella.

Image Set 1

Fig 5.11 shows cartilage images extracted from Image Set 1.

Image Set 2

Fig 5.12 shows cartilage images extracted from Image Set 2.

Image Set 3

Fig 5.13 shows cartilage images extracted from Image Set 3.

The results obtained by using ACMM are more accurate than BSSM and NNCM. ACMM provides cartilage images that are the closest to actual cartilage. ACMM can

deal with problem existing in BSSM when there are low contrast between cartilage and background. For example:

- Image 2 , 6 on image set 1 in Fig 5.11
- Image 6, 8 on image set 2 in Fig 5.12

ACMM also can works well with problem existing in NNCM that NNCM reduces number of cartilage pixels when some cartilage pixels are classified as background class because they have same features of background pixels. For example:

- Image 1, 5, and 8 on image set 1 in Fig 5.11
- Image 5, 8 on image set 2 in Fig 5.1

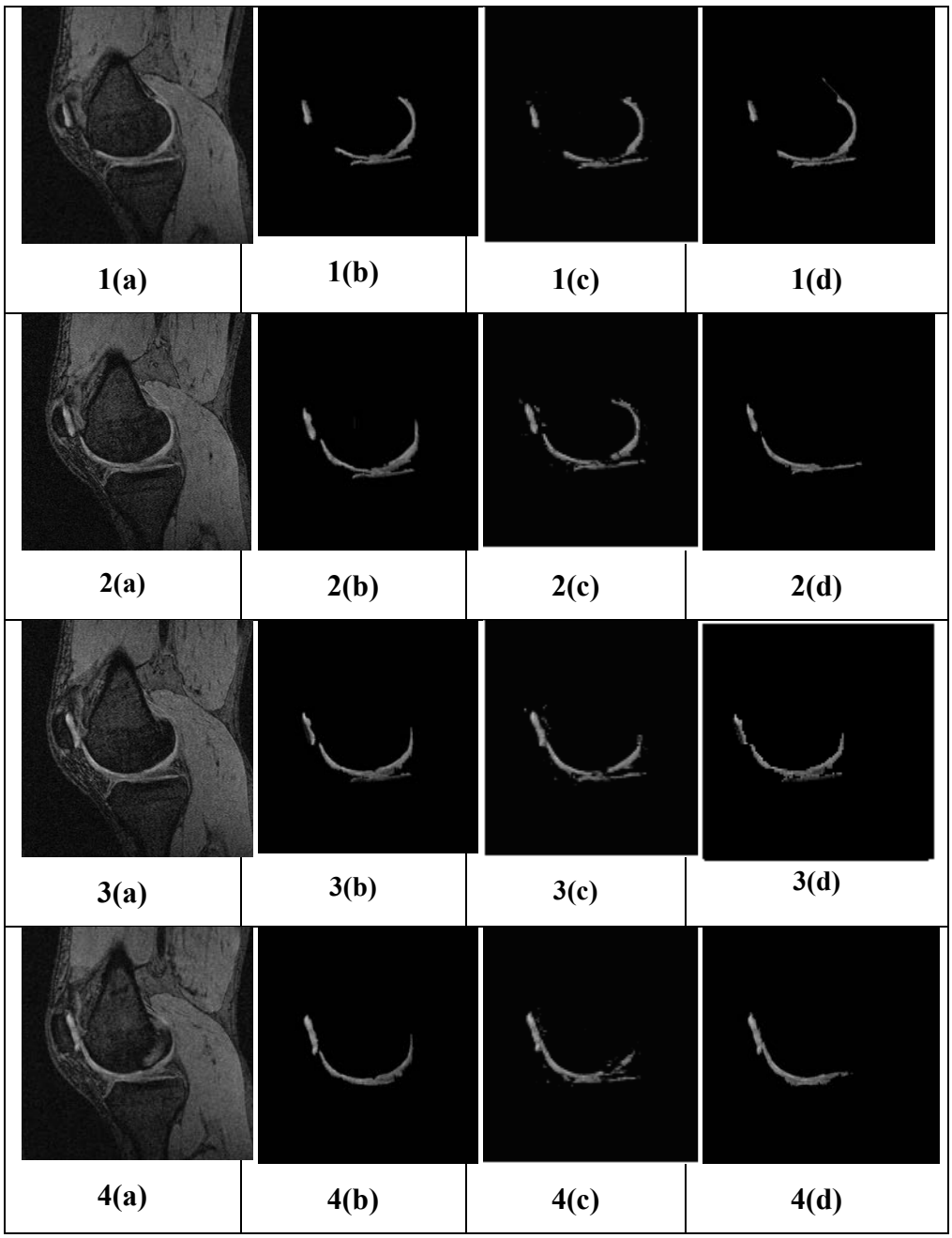


Figure 5.11 (a) Images on Image Set 1, Cartilage images extracted from Image Set 1 by using ACMM (b), NNCM (c) and BSSM (d).

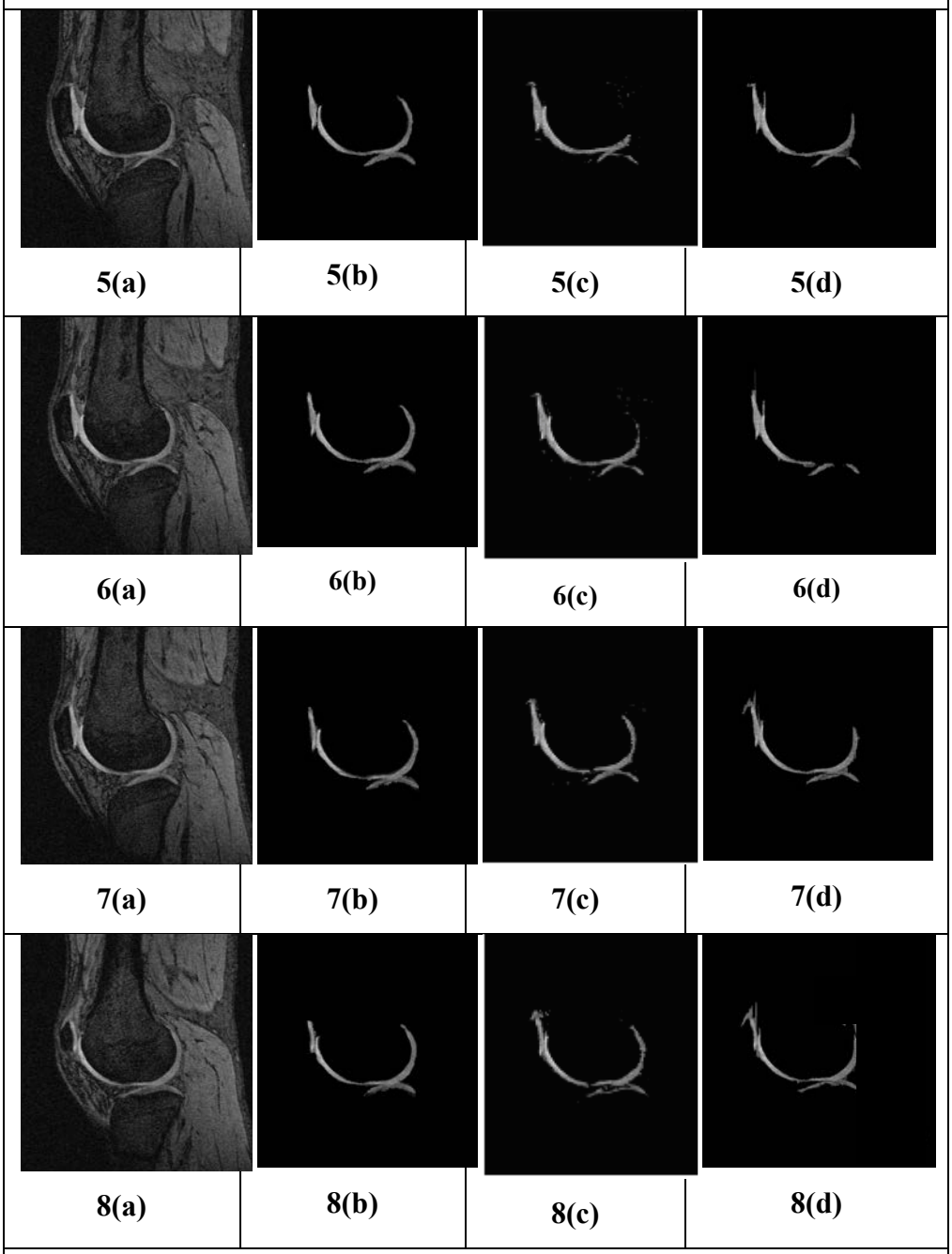


Figure 5.11 (a) Images on Image Set 1, Cartilage images extracted from Image Set 1 by using ACMM (b), NNCM (c) and BSSM (d).

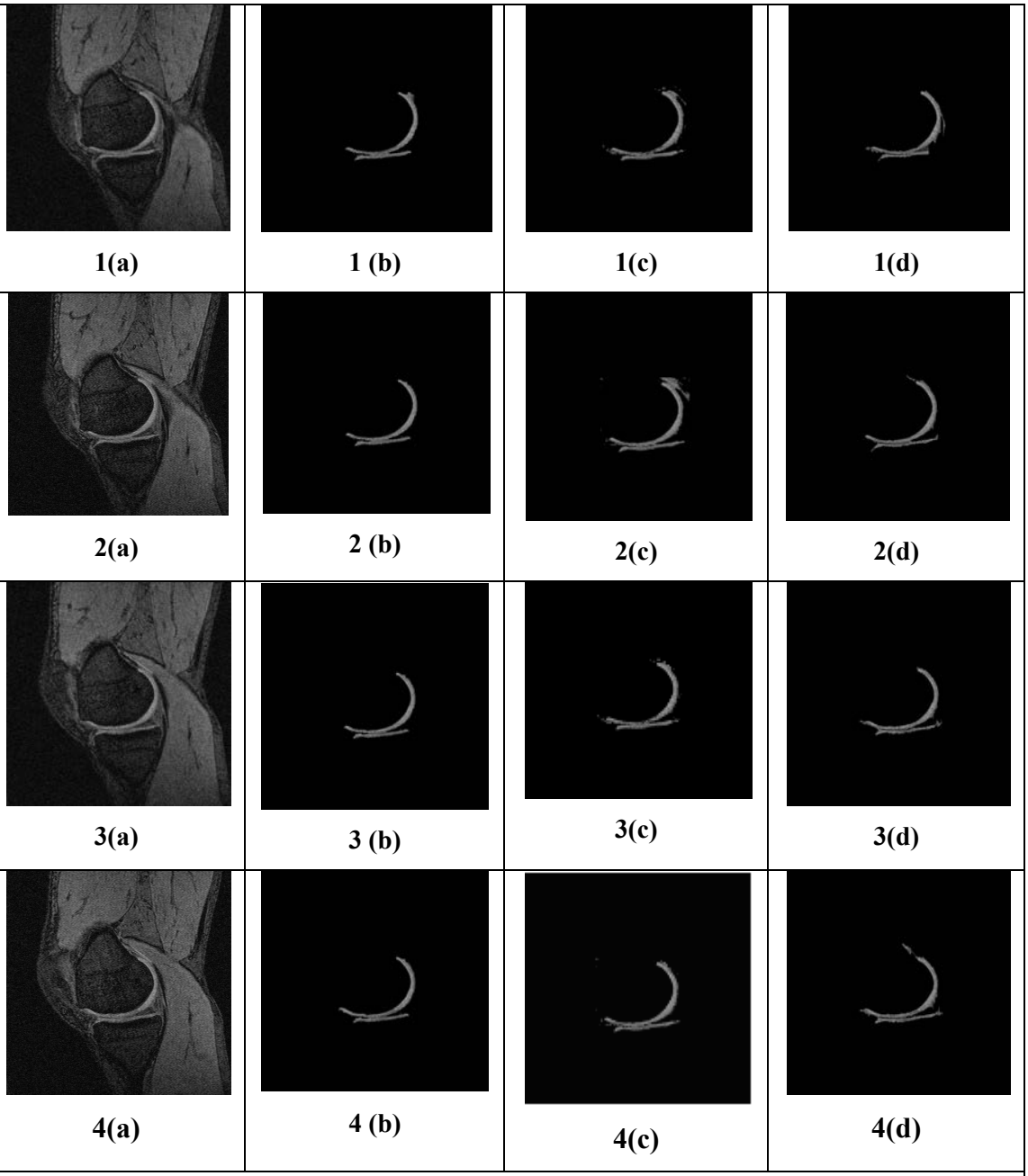


Figure 5.12 (a) Images on Image Set 2, Cartilage images extracted from Image Set 1 by using ACMM (b), NNCM (c) and BSSM (d).

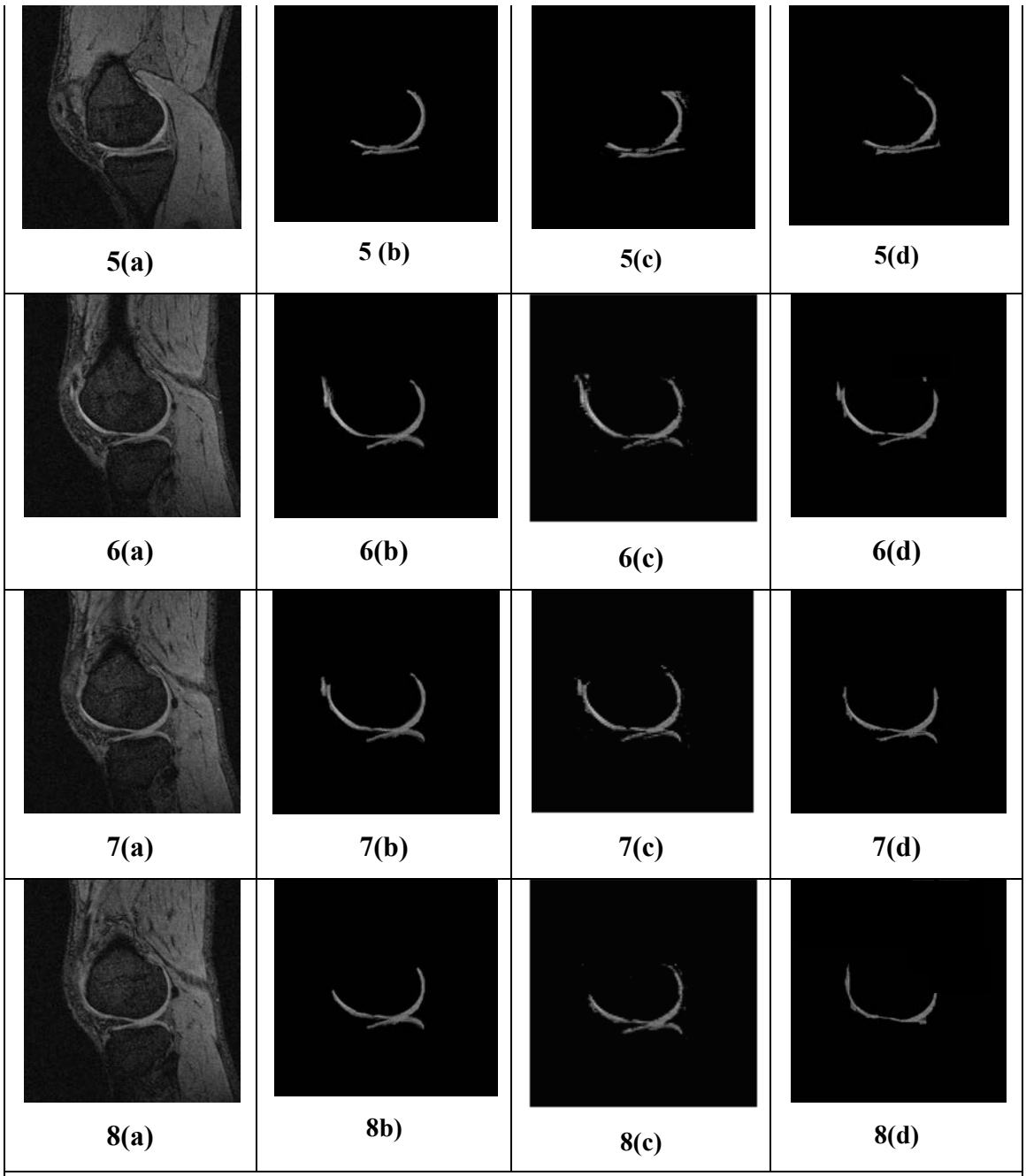


Figure 5.12 (a) Images on Image Set 2, Cartilage images extracted from Image Set 1 by using ACMM (b), NNCM (c) and BSSM (d).

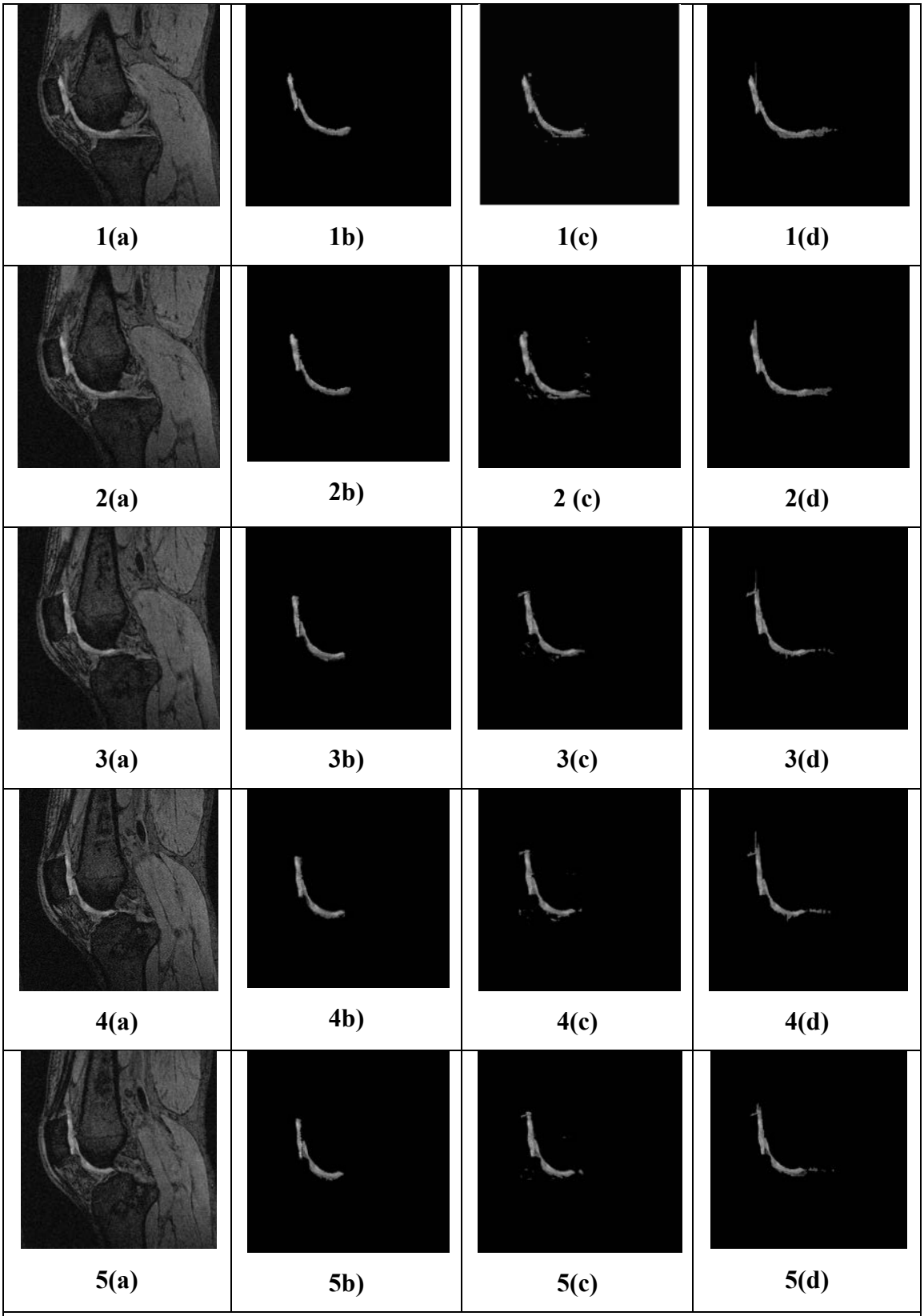


Figure 5.13(a) Images on Image Set 3, Cartilage images extracted from Image Set 1 by using ACMM (b), NNCM (c), and BSSM (d)

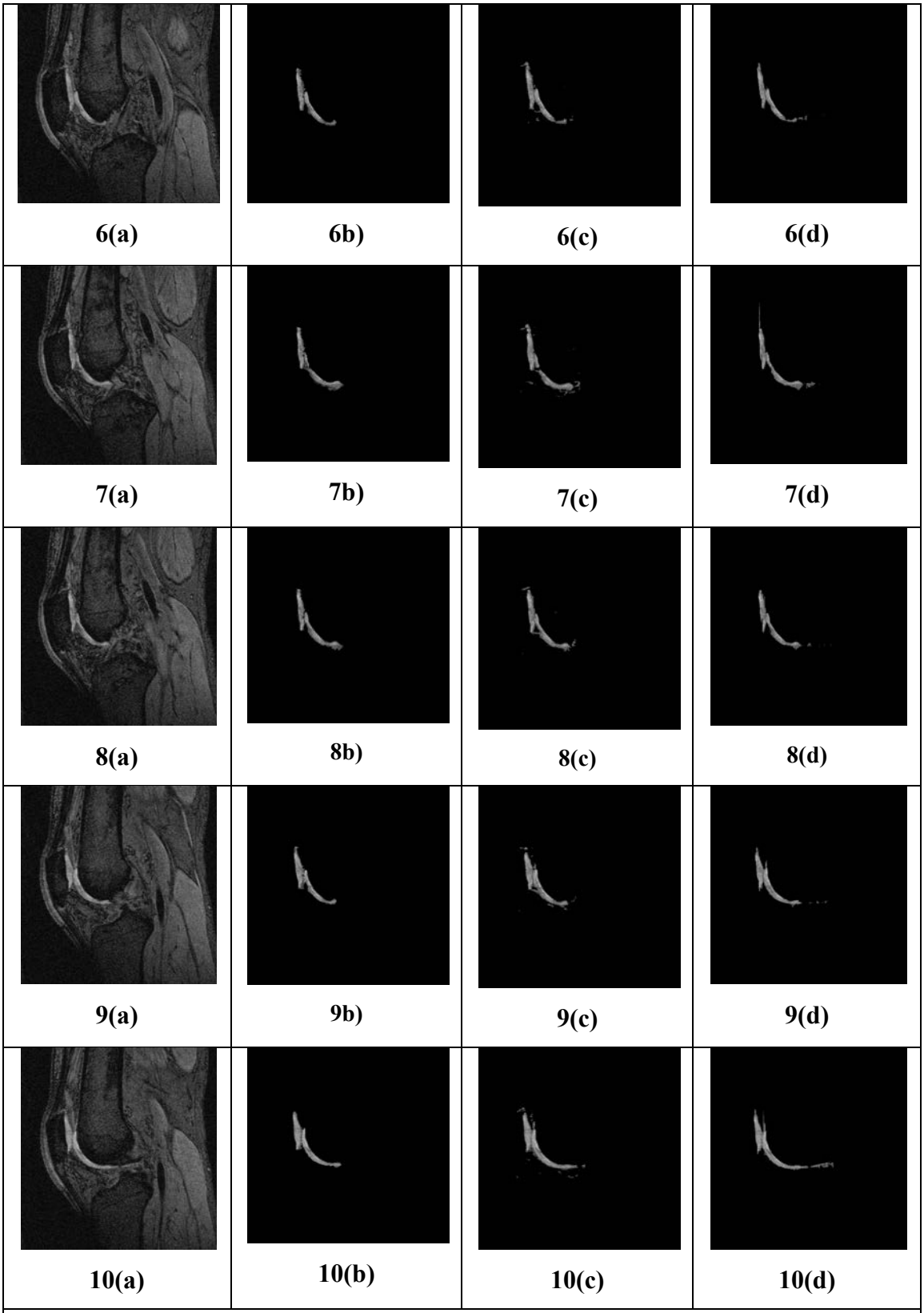


Figure 5.13(a) Images on Image Set 3, Cartilage images extracted from Image Set 1 by using ACMM (b), NNCM (c), and BSSM (d)

5.6 Conclusion

The active contour models algorithms can be used to extract an object from an original image. Our method is mainly based on active contour models algorithms. From an initial contour, we explore it to find the cartilage boundaries by calculating the internal and external energy. We apply BSSM as an automatic method to define initial contour. We also apply NNCM to compute the external energy. By doing this, we can take advantages of those two methods and avoid their disadvantages. Therefore, ACMM not only work well when there is low contrast between cartilage and background regions but also handle the problem existing in NNCM that similarly features of cartilage and background pixels. Therefore, ACMM is the most suitable approach for obtaining a cartilage from an original MR image.

Chapter 6

CARTILAGE AREA AND VOLUME CALCULATION

6.1 Overview

After an articular cartilage is extracted from original MR image by using one of three mentioned methods (bi-directional scanning segmentations, neural network, or active contour models), we can compute the cartilage's area and volume.

This chapter present the way we compute the cartilage's area and volume from cartilage image.

6.2 Cartilage Area Calculation

The area of the cartilage is proportional to the number of pixels of the cartilage. Then, we make a convert to area unit (such as *mm*).

$$\text{Area} = \delta A_{cartilage}$$

Where $A_{cartilage}$ is the number of cartilage pixels. Parameter δ is the constant ratio to convert from number of pixel to *mm*.

Consider the cartilage image after extraction, that is composed of cartilage pixels (values are greater than 0) and background pixels (values are 0). Cartilage area calculation algorithms are described as:

1. Set initial cartilage area $A_{Cartiage} = 0$;
2. For each pixel in cartilage image $f(x,y)$:

   If $g(x,y)$ is greater than 0, $A_{cartilage} = A_{cartilage} + 1$;

   If $g(x,y)$ is 0, remain $A_{cartilage}$ and move to next pixel.

   Where $g(x,y)$ is the value of pixel at location (x,y).
3. Repeat step 2 until all pixels of cartilage image $f(x,y)$ are processed.
4. The area of the cartilage is the final $A_{cartilage}$.

6.3 Cartilage Volume Calculation

From characteristics of MR knee images, the cartilage volume is calculated from the set MR knee images. Cartilage volume calculation algorithms are described as:

1. For each cartilage image, apply step 1 to step 4 of cartilage area calculation algorithms to compute an area of a cartilage on a cartilage image A_i. where *i* is the sequence of the cartilage image in MR image set.
2. Calculate the volume of cartilage's part from two adjacent cartilage images.

   $$V_i = d.\delta.A_{i+1}.A_i$$

   Where d is the distance between two cartilage images (d in *mm*). Parameter δ is the constant convert rati. The volume of a cartilage is the total volume of cartilage's parts.

   $$V_{total} = \sum_{i=1}^{N} d.\delta.A_{i+1}A_i$$

   Where N is the number of cartilage images.

6.4 Cartilage Area Calculation Validation

We apply three methods (BSSM, NNCM, and ACMM) to extract cartilage from input images. In each cartilage image, the cartilage pixels were used to compute cartilage area by using cartilage area calculation that is mentioned in section 6.2. The results are then compared with reference values, which is the number of pixels that are computed from reference cartilage images. Reference cartilage images are extracted manually from input image by using matlab function "imtool" on image processing toolbox.

The results obtained from cartilage images extracted manually correlated highly with the results obtained from cartilage images extracted by using BSSM, NNCM, and ACMM The correlation value in each case is nearly 1 (p = 1.0054, R^2 = 0.9995 in Fig 6.1; p = 1.0064, R^2 = 0.9991 in Fig 6.2; p = 0.9946, R^2 = 0.9991 in Fig 6.3). The validation results demonstrated that our three methods for extracting cartilage on an image were reliable for cartilage area calculation.

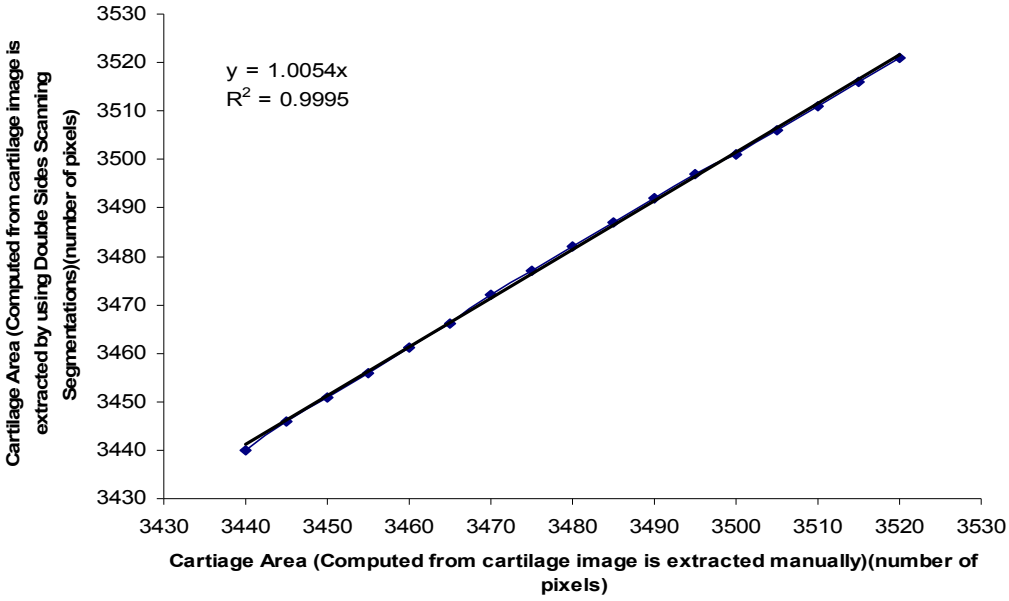


Figure 6.1 Cartilage area computed from cartilage image extracted by using BSSM correlated highly with the cartilage image extracted manually by using "imtool" function technique from Matlab

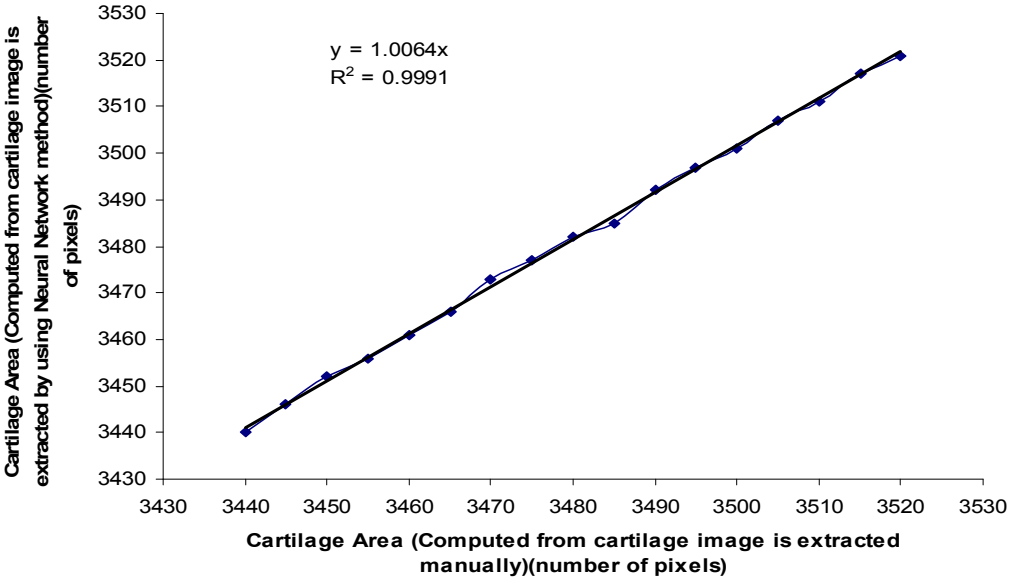


Figure 6.2 Cartilage area computed from cartilage image extracted by using NNCM correlated highly with the cartilage image extracted manually by using "imtool" function technique from Matlab

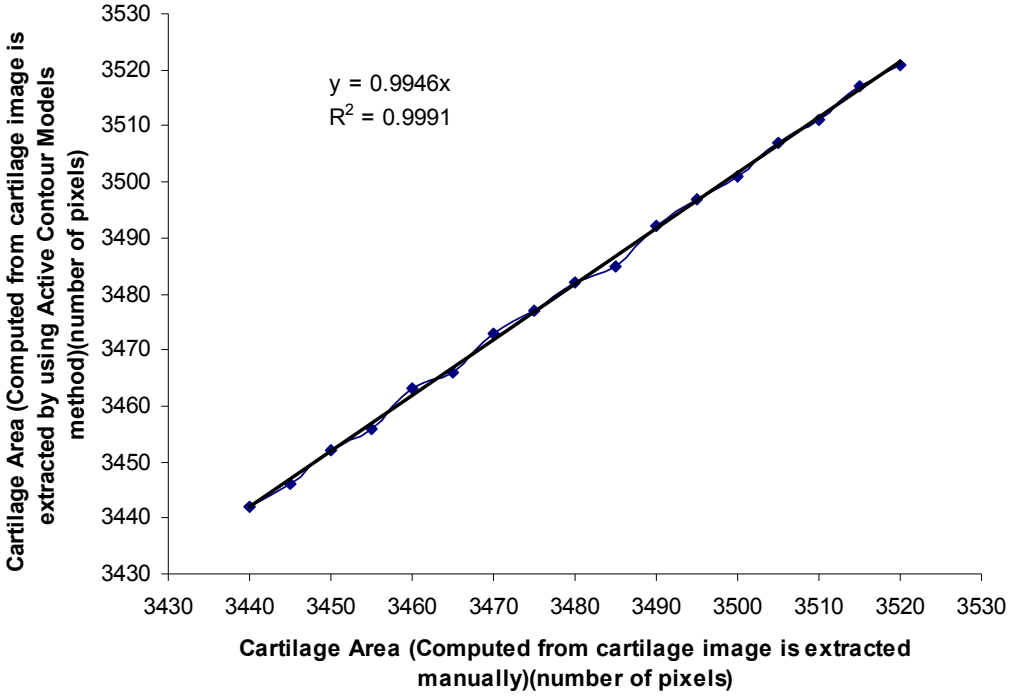


Figure 6.3 Cartilage area computed from cartilage image extracted by using ACMM correlated highly with the cartilage image extracted manually by using "imtool" function technique from Matlab

6.5 Experiment Results and Conclusion

The following table is the results of area calculation obtained from cartilage images that are extracted by using BSSM, NNCM, and ACMM. It also demonstrates the quantitative evaluation of each method while the visual results are described previously in Chapter 3, Chapter 4, and Chapter 5.

Image Sequence	BSSM	NNCM	ACMM	Manual Method
	Area (number of pixels)	Area (number of pixels)	Area (number of pixels)	Area (number of pixels)
1	3757	3600	3745	3740
2	3846	3421	3579	3565
3	4042	3552	3575	3559
4	4398	3547	3699	3670
5	4079	3612	3848	3790

6	3849	4523	4818	4820
7	5012	5516	5626	5590
8	5500	5011	5534	5550
9	4321	4023	4649	4590
10	3810	3199	3543	3520
11	3923	3276	3768	3710
12	4011	3123	3807	3800
13	3902	3566	3746	3745
14	4021	3742	3901	3880
15	3400	2918	3315	3347
16	1572	3122	3514	3460
17	3812	3014	3513	3477
18	3502	3155	3475	3380
19	3490	2977	3475	3400
20	3005	2516	2929	2959
21	2712	2413	2656	2565
22	2900	2676	2880	2751
23	2859	2812	2829	2822
24	3320	3011	3213	3252
25	3002	2892	3116	3031
26	4310	4011	4121	4060
27	4211	3815	4055	4980
28	4602	4556	4700	4651
29	6120	5413	5851	5753
30	5912	5045	5641	5571
31	6220	5359	5849	5722
32	5011	5123	5247	5157
33	5122	4923	5299	5287
34	4723	4899	5080	4982
35	4412	4789	4939	4803
36	3320	3615	3843	3759

Table 6.1 Result of area calculation of the cartilage by using three methods

The graph below illustrates the comparison of the cartilage areas that computed from a cartilage images extracted from image set by using BSSM, NNCM, and ACMM.

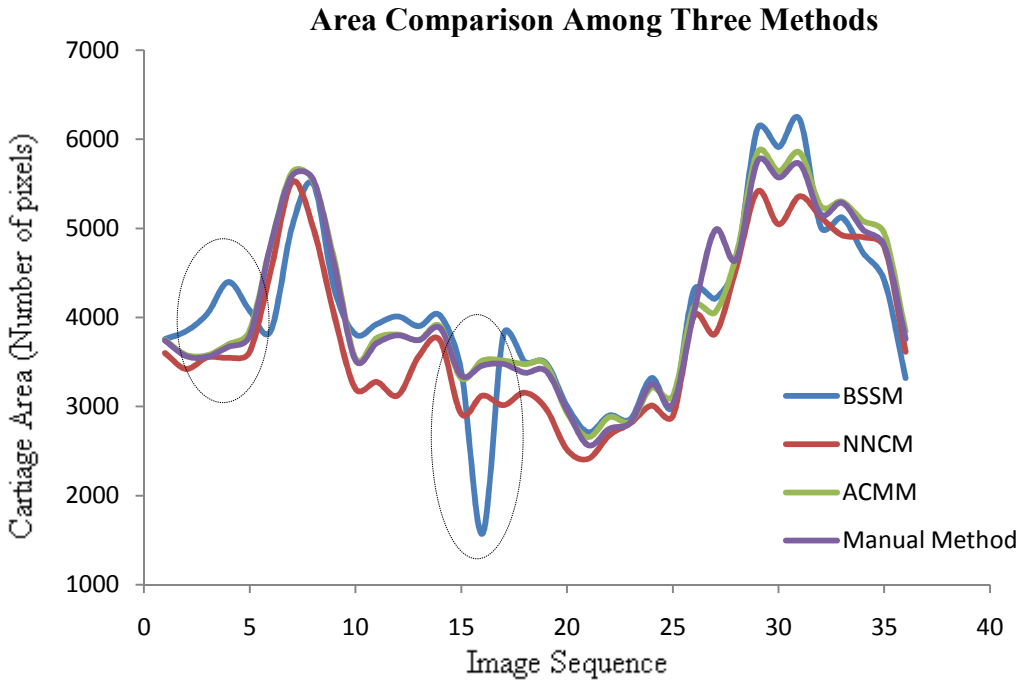


Figure 6.4 Area comparisons between using BSSM, NNCM, and ACMM

From the graphs, results obtained from NNCM and ACMM have similar shape and are close to the manual results. Results obtained from BSSM are also similar but there are differences in some sections (that are indicated in circle in Fig 6.4). It is due to the less sensitive ability to the noise of BSSM. The results that are from NNCM are generally smaller than from ACMM. It express that NNCM is likely to reduce number of cartilage pixels when some cartilage pixels are classified as background due to similar features of those pixels and background.

ACMM, which has advantages of both NNCM and BSSM, provides the most accurate results.

Chapter 7

CONCLUSION, DISCUSSION and FUTURE WORK

7.1 Conclusion

In vivo morphometry and functional analysis of human articular cartilage with quantitative magnetic resonance imaging (MRI), the size of the articular cartilage is an important element in detecting the Osteoarthritis (OA), which is a major public health problem in term of joint and knee.

Extracting an articular cartilage from an original MR knee image is an important process in cartilage size calculation. For doing this, we have studies and developed three automatic methods [1] such as BSSM, NNCM and ACMM.

BSSM is based on two basic properties of intensity value: discontinuity (edge detection algorithms) and similarity (thresholding method). It is also based on statistical analysis (curve fitting algorithms and average weight calculation)

NNCM is a method, which apply a neural network as cartilage classification. For each pixel on a MR image, through the neural network classifier, it is classified as cartilage pixel if output value is 1. Alternatively, it is classified as background pixel if output value is 0.

ACMM is a method, which uses an initial contour that approximates the boundary of a cartilage in an MR image to find the "actual" boundary. This is an innovative method because we apply BSSM to define an initial contour. We also apply NNCM to compute the external energy. Therefore, it takes advantages of both methods and avoids the problems existing in BSSM and NNCM.

7.2 Discussion

Our three methods have succeeded in automatically extracting the cartilage from input image. According to noise and complexity of image, each method has both advantages and disadvantages.

BSSM works well when there are significant high contrasts between cartilage and background regions. It often fails when the contrasts are low.

NNCM can handle the problem existing in BSSM. However, this method has problem when there are some cartilage pixels that have similar features of background. NNCM is likely to consider those pixels as background pixels. Therefore, it reduces the size of a cartilage on an image.

Since inheriting advantages of BSSM and NNCM, active contour models method is the most suitable method for extracting a cartilage. It can solve problems that occur in both previous methods. Nevertheless, using NN as external energy is computationally expensive (in both training stage and testing stage). In this thesis's situation, the speed is not critical and in return, NN bring greater benefit than other method.

7.3 Future Works

Since ACMM is an automatic potential method for extracting a cartilage from MR image, we have some suggestions to improve the performance of this approach.

1. Initial contour improvement: Because we apply BSSM for defining the initial contour, we can improve the quality of initial contour by improving the performance of BSSM. For doing this, we can try several ways such as:

- We can apply morphological watersheds for segmentation instead of using edge detection or thresholding algorithms.
- We can define more rules and principles in the combination of edge detection, thresholding, and statistical analysis to make this method more flexible and adaptive with noise.

2. External energy computation improvement: Because we apply NNCM to compute the external energy, we can improve the quality of external energy by improving the performance of NNCM. For doing this, we can try several ways such as:

- We can apply another type of neural network instead of multilayer perceptrons. For examples: radial-basic neural networks (RBNN) or transformation radial-basic networks (TRBNN)
- We can maintain multilayer perceptrons but improve its efficiency by improve the quality and quantity of training data.
- We can define more rules and principles in using network classifier as cartilage recognition.

REFERENCE

[1] Ngo, Q and Jiang, D and Ding, C, Application of Artificial Neural Networks in Automatic Cartilage Segmentation', *Proceedings of IWACI2010*, 25-27 August, 2010, Suzhou, China, pp. 81-85. ISBN 978-1-4244-6336-7, 2010..

[2] Peyron JG (1986); "Osteoarthritic"; The epidemiologic viewpoint; Clin Orthop 213: pp13-19.

[3] Felson DT (1988); "Epidemiology of hip and knee osteoarthritis"; Epidemiol Prev 10: pp1-28.

[4] Felson DT (1990); "Osteoarthristic"; Rheum Dis Clin North Am 16: pp499-512.

[5] Felson DT, Zhang Y, Hannan MT, Naimark A, Weissman BN, Aliabadi P, and Levy D (1995); "The incidence and natural history of knee osteoarthritis in the elderly"; The Framingham osteoarthritis study; Arthritis Rheum 38: pp1500-1505.

[6] Guccione AA, Felson DT, Anderson JJ, Anthony JM, Zhang Y, Wilson PW, Kelly H; Wolf PA; Kreger BE; Kannel WB (1994); "The effects of specific medical conditions on the functional limitations of elders in the Framingham study"; Am J Public Health 84: pp 351-358.

[7] Changhai D, Flavia C, and Graeme J; "How important is MRI for detecting early osteoarthristic"; Nature clinical practice Reumatology; Vol 4; 2008: pp 1-3.

[8] Eckstein F (1996); "Determination of knee joint cartilage thickness using three dimensional magnetic resonance chondro-crassometry (3D-MR-CCM)"; Magn Reson Med 36: pp 256-265.

[9] Piplani MA (1996); "Articular cartilage volume in the knee; semiautomatic determination from three-dimensional reformations of MR images"; Radiology 198: pp 855-859.

[10] Stammberger T (1999); "Interobserver reproducibility of quantitative cartilage measurements: comparison between B-spline snakes and manual segmentation";Magn Reson Imag 17: pp 1033-1042.

[11] Solloway S (1997); "The use of active shape models for making thickness measurements of articular cartilage from MR iamges"; Magn Reson Med 37: pp 943-952.

[12] Robson MD (1995); "A combined analysis and magnetic resonance imaging technique for computerised automatic measurement of cartilage thickness in the distal interphalangeal joint"; Magn Reson Imaging 13: pp 709-718.

[13] Kshirsagar AA (1998); "Measurement of localized cartilage volume and thickness of human knee joints by computer analysis of three-dimensional magnetic resonance images"; Invest Radiol 33: pp 289-299.

[14] Ghosh S (2000); "Segmentation of high resolution articular cartilage MR images"; Transactions of the 46th Meeting of the Orthop Res Soc vol 25: pp 246-250.

[15] Steines D (2000); "Segmentation of osteoarthritic femoral cartilage from MR images"; In: Lemke HU; Vannier NW; Inamura K; Farman AG; Doi K (eds) Proceedings of Computer Assisted Radiology and Surgery; 14th International Congress. Excerpta Medica Series 1214. Elsevier; Amsterdam; pp 303-308.

[16] Peterfy CG, and Genant HK (1996); "Emerging application of magnetic resonance imaging in the evaluation of articular cartilage"; Radiol Clin North Am 34: pp195-213.

[17] Stabler A, Glaser C, and Reiser M (2000); "Musculosketetal MR: Knee"; Eur Radiol 10: pp230-241.

[18] Petefy CG (2000); "Scratching the surface: articular cartilage disorders in the knee"; Magn Reson Imaging Clin North Am 8: pp409-430.

[19] Robson MD, Hodgson RJ, Herrod NJ, Tyler JA, and Hall LD (1995); "A combined analysis and magnetic resonance imaging technique for computerised automatic measurement of cartilage thickness in the distal interphalangeal joint"; Magn Reson Imaging 13: pp709-718.

[20] Munsterer O, Eckstein F, Hahn D, and Putz R (1996); "Computer aided 3D assessment of human knee cartilage in vitro and in vivo"; Clin Biomech 11: pp206-266.

[21] Recht MP, Kramer J, Marcelis S, Pathria MN, Trudell D, Haghigi P, Sartoris Dj, and Resnick D (1993); "Abnormalities of articular cartilage in the knee: analysis of available MR techniques"; Radiology 187: pp473-478.

[22] Rafael CG, and Richard EW; "Digital Image Processing"; Prentice Hall 2nd Edition 2002 chapter 2: pp57-66.

[23] Rafael CG, and Richard EW; "Digital Image Processing"; Prentice Hall 2nd Edition 2002 chapter 10: pp572-580.

[24] Rafael CG, and Richard EW; "Digital Image Processing"; Prentice Hall 2nd Edition 2002 chapter 10: pp581-594.

[25] Rafael CG, and Richard EW; "Digital Image Processing"; Prentice Hall 2nd Edition 2002 chapter 10: pp595-615.

[26] Alasdair MA; "Introduction to digital image processing"; Course Technology; a division of Thomson Learning; Inc; 2004: pp 222-225.

[27] From Wikipedia; "Least Mean Squares Algorithms". Available at: http://en.wikipedia.org/wiki/Least_mean_squares

[28] E.Gose, R.Johnsonbaugh, and S.Jost; Pattern Recognition and Image Analysis;1996.

[29] B.Jahne, and H.Haubecker; Computer vision and Applications: A guide for Students and Practitioners; 2000.

[30] Bernd J, and Horst H; "Computer vision and applications"; 2000: pp :577-583.

[31] Windrow B, Lehr MA; 30 Years of Adaptive Neural Networks: Perceptrons, Madaline, and Backpropagation; Proc. IEEE, vol 78, no 9, 1990

[32] Mathematics source library C&ASM. Available at: http://mymathlib.webtrellis.net/optimization/nonlinear/unconstrained/fletcher_reeves.html
- The MathWorks. Available at: http://www.mathworks.co.uk/access/helpdesk/help/toolbox/nnet/backpro7.html

[33] Fletcher R, and C.M. Reeves; "Function minimization by conjugate gradients;" Computer Journal; Vol. 7; 1964; pp. 149-154.
Hagan; M.T, H.B. Demuth; and M.H. Beale; "Neural Network Design"; Boston; MA: PWS Publishing; 1996.

[34] - Y.H.Dai; and Y.Yuan; Convergence Properties of Beale-Powell Restart Algorithm; Chinese Academy of Siences; Beijing 10080; China. Available at: ftp://lsec.cc.ac.cn/pub/home/yyx/papers/dy-bp-restart.pdf
- The MathWork. Available at: http://www.mathworks.co.uk/access/helpdesk/help/toolbox/nnet/backpro7.html

[35] Powell; M.J.D.; "Restart procedures for the conjugate gradient method;" Mathematical Programming; Vol. 12; 1977; pp. 241-254.
Beale; E.M.L.; "A derivation of conjugate gradients;" in F.A. Lootsma; Ed.; Numerical methods for nonlinear optimization; London: Academic Press; 1972.

[36] Powell, M.J.D., "Restart procedures for the conjugate gradient method," Mathematical Programming, Vol. 12, 1977, pp. 241-254.
Beale, E.M.L., "A derivation of conjugate gradients," in F.A. Lootsma, Ed., Numerical methods for nonlinear optimization, London: Academic Press, 1972.

[37] - M.F.Moller; A Scale Conjugate Gradient Algorithm for Fast Supervised Learning. Available at: ftp://ftp.dca.fee.unicamp.br/pub/docs/vonzuben/ia353_1s07/papers/moller_90.pdf
- The MathWork. Available at: http://www.mathworks.co.uk/access/helpdesk/help/toolbox/nnet/backpro7.html

[38] Moller; M.F.; "A scaled conjugate gradient algorithm for fast supervised learning;" Neural Networks; Vol. 6; 1993; pp. 525-533.

[39] Moller, M.F., "A scaled conjugate gradient algorithm for fast supervised learning," Neural Networks, Vol. 6, 1993, pp. 525-533.

[40] - Wikipedia. Available at: http://en.wikipedia.org/wiki/Quasi-Newton_method

- Wolffram Research Center. Available at:
http://reference.wolfram.com/mathematica/tutorial/UnconstrainedOptimizationQuasiNewtonMethods.html
- The MathWorks. Available at:
http://www.mathworks.co.uk/access/helpdesk/help/toolbox/nnet/backpro7.html

[41] Dennis; J.E.; and R.B. Schnabel; Numerical Methods for Unconstrained Optimization and Nonlinear Equations; Englewood Cliffs; NJ: Prentice-Hall; 1983.

[42] Dennis, J.E., and R.B. Schnabel, Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Englewood Cliffs, NJ: Prentice-Hall, 1983.

[43] Battiti; R.; "First and second order methods for learning: Between steepest descent and Newton's method;" Neural Computation; Vol. 4; No. 2; 1992; pp. 141-166.

[44] Hagan; M.T.; and M. Menhaj; "Training feed-forward networks with the Marquardt algorithm;" IEEE Transactions on Neural Networks; Vol. 5; No. 6; 1999; pp. 989-993; 1994.

[45] Hagan, M.T., and M. Menhaj, "Training feed-forward networks with the Marquardt algorithm," IEEE Transactions on Neural Networks, Vol. 5, No. 6, 1999, pp. 989-993, 1994.

[46] Kass M; Andrew W; and Demetri T; “Snakes: Active Contour Models”; International Journal of Computer Vision (1988); pp: 321-331.

[47] Kok Fung Lai. Deformable Contours: Modeling; Extraction; Detection; and Classification. PhD thesis; University of Wisconsin-Madison; Electrical Engineering Department; Madison; Wisconsin;1994

[48] V.Chalana;W.Costa; and Y.Kim. Integrating region growing and edge detection using regularization. In Procedding of the SPIE Conference on Medical Imaging. SPIE; 1995

APPENDIX

All algorithms in this thesis are implemented in Matlab environment by using Matlab software and Image Processing Toolbox. This appendix provides detailed Matlab codes of each algorithm.

1. Data Acquisition from MRI to Matlab environment

```
% The following codes is used to acquire an MRI scan to Matlab environment for
% image  processing

% select a MR image for processing
fid=0;
while fid < 1
   [fid,message] = fopen(ImageFile, 'r');
   if fid == -1
      errordlg('Please enter the correct file name','Invalid input');
      return
   end
end
if (strcmp(ImageFile,'')==0)
   errordlg('Please enter the correct file name','Invalid input');
   return
end
filename=char(ImageFile);

% Put MR image into Matlab environment
im_original=dicomread(char(ImageFile));
im_original=uint8(im_original);
```

1. Classical Image Segmentations Algorithms

```
function [Object1,Object2,Object3]=Main(im_original)

% This function is used to obtain a cartilage from the input image  and  measure the
% size of individual cartilage types.
% Determine the status of the input image by using "get_status" function.
% Obtain the first object (Femur) by using "get_object1" function.
% Obtain the second object(Tibia) by using "get_object2" function.
% Obtain the third object (Patella)by using "get_object3" function.
% Measure the size of the individual cartilage types by using  "getsize" function.

IM = im_original;
[K,L]=size(IM);
condition=0;

% Determine the status of an image.
status=get_status(IM);

% Obtain the first object (first type of cartilage) on an image.
[Object1,condition]=get_object1(IM);

% Obtain the second object (second type of cartilage) on  an image.
if status==1|| status==2
    IM2 = IM - Object1;
    Object2 = get_object2(IM2,condition);
else
    Object2 = uint8(zeros(K,L));
end

% Obtain the third object (third type of cartilage) on an image.
if status == 2 || status == 3
    IM3 = IM - Object1 - Object2;
    Object3 = get_object3(IM3);
```

```
    size(Object3)
else
    Object3=uint8(zeros(K,L));
end
```

1.2 "get_status" function

```
function status = get_status(im_original)

% This function is used to get the status of the image.
% Input: original image. Output: Status of the image.
% There are 4 status:
% Status = 0 : Nothing
% Status = 1 : Image contains two objects: Fermur and Tibia
% Status = 2 : Image contains three objects: Fermur, Tibia and  Patella
% Status = 3 : Image contains two objects: Fermur and Patella

mediadata = dicominfo(im_original);
ID = mediadata.PatientID;
number = mediadata.InstanceNumber;
if 0 <= number &&number <= 5
   status = 0;
elseif 6 <= number &&number <= 18
   status = 1;
elseif 19 <= number &&number <= 21
   status = 2;
elseif 22 <= number &&number <= 37
   status = 3;
elseif 38 <= number &&number <= 46
   status = 2;
elseif 47 <= number &&number <= 55
   status = 1;
else
   status = 0;
```

```
end
```

1.3 "get_object1" function

```
function [Object1,condition]=get_object1(im_original)

% This function is used to get the first object on an input image.
% Input : original image.
% Output : The first Object.
% To obtain the object boundaries on a sub-image: use "get_bound" function.
% To checking the validation of object boundaries: use "checking" function.
% To obtain the object on the left side of an image: use "half_left_process1"
% function.
% To obtain the object on the right side of an image: use "half_right_process1"
% function.

[K,L]=size(IM);
Object1=IM;
reference_intensity=80;

% Determine a starting object sub-image and upper/lower object boundaries
% on this sub- image.

for i=160:256
   column=IM(:,i);
   [a1 a2]=find(column==max(column(:)));
   position=a1(1,1);
   array1_1(i,1)=column(position,1);
   array1_2(i,1)=position;
end
[b1 b2]=find(array1_1==max(array1_1(:)));

% Determine the starting object sub-image.
started_column=b1(1,1);
```

```
position=array1_2(started_column,1);
column1=IM(:,started_column);

% Find the upper/lower boundaries on a starting object sub-image.
[temp_lower_bound,temp_upper_bound]=get_bound(column1,position,IM);
upper_bound=temp_upper_bound;
lower_bound=temp_lower_bound;

% Check the validation of boundaries
[n_upper_bound,n_lower_bound]=checking(upper_bound,lower_bound);
upper_bound=n_upper_bound;
lower_bound=n_lower_bound;

% Extract an object on starting sub-image based on upper/lower boundaries
Object1(1:upper_bound,started_column)=0;
Object1(lower_bound:K,started_column)=0;
started_upper_bound=upper_bound;
started_lower_bound=lower_bound;

% Obtain an object on the right side of an image
[Object1,condition]=half_right_process1(Object1,started_column,started_upper_bound
,started_lower_bound,IM);

% Obtain an object on the left side of an image.
Object1=half_left_process(started_column,started_upper_bound,started_lower_bound,
Object1,IM);
```

1.3.1 "get_bound" function

```
function [left,right] = get_bound(column,position,IM)

% This function is used to find the upper and lower boundaries of an object on
% a sub-image  using edge detection and thresholding.
% Input:
```

```
%    - column : sub_image
%    - position : starting point
%    - input image
% Output: upper boundary and lower boundary

column1=column;
[K L]=size(IM);
alpha=3;

% Calculate optimal threshold value
reference_value=graythreshold(column1);

% Apply edge detection to find edges combined with threshold value.
for x=3:(K-2)
   column1_1(x,1)=(column1(x-1,1)+column1(x,1)+column1(x+1,1))/alpha;
   if column1_1(x,1)>85
      column1_1(x,1)=85;
   end
end
for x=2:(K-3)
   column1_1Lap(x,1)=column1_1(x+1,1)+column1_1(x-1,1)-2*column1_1(x,1);
end

% Find the upper boundary on a sub-image.
for k=1:50
   if position-k<=0
      var1=column1_1(1,1);
   else
      var1=column1_1(position-k,1);
   end
   if var1<=reference_value
      temp_upper_bound=position-k;
      break;
   elseif k==50
```

```
      temp_upper_bound=position-50;
   end
end

% Find the lower boundary on a sub-image.
for k=1:50
   var1=column1_1(position+k,1);
   if var1<=reference_value
      temp_lower_bound=position+k;
      break;
   elseif k==50
      temp_lower_bound=position+50;
   end
end
```

1.3.2 "Checking" function.

```
% This function is used to check the validation of the upper and lower  boundaries of
% an object on a sub- image.

function [newupper_bound,newlower_bound]=checking(upper_bound,lower_bound)
if upper_bound<=0
   new_upper_bound=1;
else
   new_upper_bound=upper_bound;
end
if lower_bound<=0
   new_lower_bound=1;
else
   new_lower_bound=lower_bound;
end
```

1.3.3 "half_left_process1" function.

```
% The following codes is used to obtain the object in the left side of a sub-image.
% Input: starting object sub-image, starting upper/lower boundaries, original image.
% Output: object in the left side of a sub-image.

function
Object=half_left_process1(started_column,started_upper_bound,started_lower_bound,
Object,IM)
[K,L]=size(IM);
y=started_column;
pre_upper_bound=started_upper_bound;
pre_lower_bound=started_lower_bound;

% Obtain object boundaries on partitioned sub-images.
for q=1:round(L/2)-1
   column5=IM(:,y-q);
   column51=column5;
   column51(1:(pre_upper_bound),1)=0;
   column51((pre_lower_bound):K,1)=0;
   [a2 b2]=find(column51==max(column51(:)));
   position5=a2(1,1);
   array4(y-q,1)=position5;
   if column5(position5,1)>80

% Obtain upper/lower boundaries on a sub-image.
[temp_lower_bound5,temp_upper_bound5]=get_bound2(column5,position5,IM);
      upper_bound5=temp_upper_bound5;
      lower_bound5=temp_lower_bound5;

% Check the validity of boundaries.
[n_upper_bound5,n_lower_bound5]=checking(upper_bound5,lower_bound5);
      upper_bound5=n_upper_bound5;
      lower_bound5=n_lower_bound5;
```

```
        array41(y-q,1)=upper_bound5;
        array42(y-q,1)=lower_bound5;

% Extract an object from a sub-image
        Object(1:upper_bound5,y-q)=0;
        Object(lower_bound5:K,y-q)=0;
        pre_upper_bound=upper_bound5;
        pre_lower_bound=lower_bound5;
    else
        Object(:,1:y-q)=0;
        limit_swing_left=y-q;
        break;
    end
end
```

1.3.4 "half_right_process1" function.

```
% The following codes is used to obtain an object in the right side  of an image.
% Input: starting object sub-image, starting upper/lower  boundaries, original image.
% Output: Object in the right side of an image.

function
[Object1,condition]=half_right_process1(Object1,started_column,started_upper_boun
d,started_lower_bound,IM)
pre_upper_bound=started_upper_bound;
pre_lower_bound=started_lower_bound;
[K,L]=size(IM);

% Obtain object boundaries on partitioned sub-images
for y=(started_column+1):L
    column2=IM(:,y);
    column21=column2;
    column21(1:(pre_upper_bound),1)=0;
    column21((pre_lower_bound):K,1)=0;
```

```
   [a2 b2]=find(column21==max(column21(:)));
   position2=a2(1,1);
   array2_0(y,1)=position2;
   if column2(position2,1)>80

% Determine upper/lower boundaries on a sub-image.
[temp_lower_bound2,temp_upper_bound2]=get_bound(column2,position2,IM);
      upper_bound2=temp_upper_bound2;
      lower_bound2=temp_lower_bound2;
      array2_1(y,1)=upper_bound2;
      array2_2(y,1)=lower_bound2;
         if y>=started_column+2&&y<320
         value_1=upper_bound2-array2_1(y-1,1);
         value_2=lower_bound2-array2_2(y-1,1);
         if abs(value_1)>=5
            upper_bound2=array2_1(y-1,1);
         elseif abs(value_2)>=5
            lower_bound2=array2_2(y-1,1);
         end
      end

% Check the validity of boundaries.
[n_upper_bound2,n_lower_bound2]=checking(upper_bound2,lower_bound2);
      upper_bound2=n_upper_bound2;
      lower_bound2=n_lower_bound2;
      array2_1(y,1)=upper_bound2;
      array2_2(y,1)=lower_bound2;

% Extract an object from a sub-image
      Object1(1:upper_bound2,y)=0;
      Object1(lower_bound2:K,y)=0;
      pre_upper_bound=upper_bound2;
      pre_lower_bound=lower_bound2;
   else
```

```
      if y>=320
         Object1(:,y:L)=0;
         break;
      else
         Object1(:,y)=0;
         array2_1(y,1)=upper_bound2;
         array2_2(y,1)=lower_bound2;
      end
   end
end

% Check for probability of swing back
[m n]=size(array2_1);
array2_1_1=array2_1;
for i=1:m
   if array2_1_1(i,1)==0
      array2_1_1(i,1)=400;
   end
end
[a1 b1]=find(array2_1_1==min(array2_1_1(:)));
min_point=a1(1,1);
condition=0;
if abs(array2_1(started_column+1,1)-array2_1(min_point,1))>=80
    % Do Swing back process.
   Condition=1;
   Object_swing_back=swing_back(array2_1,min_point,IM);

    % Object is extracted from right side of an image.
   Object1=Object1+Object_swing_back;
end
```

1.4 “get_object2” function.

```
% The following codes is used to extract the second object (second  types of cartilage)
% from an original image.
% Input : original image
% Output : Object
% To obtain the object boundaries on a sub-image by using "get_bound"  function.
% To checking the object boundaries if it is accepted or not by using "checking"
% function.
% To obtain the object on the left side of sub-image by using "half_left_process2"
% function.
% To obtain the object on the right side of sub-image by using "half_right_process2"
% function.

function Object2 = get_object2(IM2,condition)
[K, L]=size(IM2);
Object2 = IM2;

% Obtain object boundaries on partitioned sub-images.
for i =200:256
   column=IM2(:,i);
   [a1 a2]=find(column==max(column(:)));
   position=a1(1,1);
   array1_1(i,1)=column(position,1);
   array1_2(i,1)=position;
end
[b1 b2]=find(array1_1==max(array1_1(:)));
started_column=b1(1,1);
position=array1_2(started_column,1);
column1=IM2(:,started_column);

% Determine upper/lower boundaries on a sub-image.
[temp_lower_bound,temp_upper_bound]=get_bound(column1,position,IM2);
upper_bound=temp_upper_bound;
lower_bound=temp_lower_bound;
```

```
% Check the validity of boundaries.
[n_upper_bound,n_lower_bound]=checking(upper_bound,lower_bound);
upper_bound=n_upper_bound;
lower_bound=n_lower_bound;

% Extract an object from a sub-image.
Object2(1:upper_bound,started_column)=0;
Object2(lower_bound:K,started_column)=0;
started_upper_bound=upper_bound;
started_lower_bound=lower_bound;

% Obtain an object on the right side of an image.
Object2=half_right_process2(Object2,started_column,started_upper_bound,started_low
er_bound,IM2,condition);

% Obtain an object on the left side of an image.
Object2=half_left_process2(started_column,started_upper_bound,started_lower_bound
,Object2,IM2);
```

1.4.1 “half_right_process2” function

```
% The following codes is used to obtain an object in the  right side of a original
% image.
% Input: starting object sub-image, starting upper/lower  object boundaries, original
% image.
% Output: Object on the right side of an original image.

function
Object2=half_right_process2(Object2,started_column,started_upper_bound,started_lo
wer_bound,IM2,condition)
[K,L]=size(IM2);
pre_upper_bound=started_upper_bound;
pre_lower_bound=started_lower_bound;
```

```
% Obtain object boundaries on partitioned sub-images.
for y=(started_column+1):L
   column2=IM2(:,y);
   column21=column2;
   column21(1:(pre_upper_bound),1)=0;
   column21((pre_lower_bound):K,1)=0;
   [a2 b2]=find(column21==max(column21(:)));
   position2=a2(1,1);
   array2_0(y,1)=position2;
   if column2(position2,1)>80

% Determine the upper/lower boundaries on a sub-image.
[temp_lower_bound2,temp_upper_bound2]=get_bound(column2,position2,IM2);
      upper_bound2=temp_upper_bound2;
      lower_bound2=temp_lower_bound2;
      array2_11(y,1)=upper_bound2;
      array2_21(y,1)=lower_bound2;
      if y>=started_column+2&&y<320
         value_1=upper_bound2-array2_11(y-1,1);
         value_2=lower_bound2-array2_21(y-1,1);
         if abs(value_1)>=5
            upper_bound2=array2_11(y-1,1);
         elseif abs(value_2)>=5
            lower_bound2=array2_21(y-1,1);
         end
      end

% Check the validity of boundaries.
[n_upper_bound2,n_lower_bound2]=checking(upper_bound2,lower_bound2);
      upper_bound2=n_upper_bound2;
      lower_bound2=n_lower_bound2;
      array2_11(y,1)=upper_bound2;
      array2_21(y,1)=lower_bound2;
```

```
% Extract an object from a sub-image.
        Object2(1:upper_bound2,y)=0;
        Object2(lower_bound2:K,y)=0;
        pre_upper_bound=upper_bound2;
        pre_lower_bound=lower_bound2;
    else
        if y>=320
            Object2(:,y:L)=0;
            break;
        else
            Object2(:,y)=0;
            array2_11(y,1)=upper_bound2;
            array2_21(y,1)=lower_bound2;
        end
    end
end

% Check for probability of swing back
if condition~=1
    [m n]=size(array2_11);
    array2_1_1=array2_11;
    for i=1:m
        if array2_1_1(i,1)==0
            array2_1_1(i,1)=400;
        end
    end
    [a1 b1]=find(array2_1_1==min(array2_1_1(:)));
    min_point=a1(1,1);
    if abs(array2_11(started_column+1,1)-array2_11(min_point,1))>=80
        % Do Swing back process.
Object_swing_back=swing_back(array2_11,min_point,IM2);

        % Obtain an object on the right side of an image.
        Object2=Object2+Object_swing_back;
```

```
   end
end
```

1.4.2 "half_left_process2" function

```
% The following codes is used to obtain an object on the  left side of an original
% image.
% Input: starting object sub-image, starting upper/lower  object boundaries, original
% image,
% Output: Object on the left side of an original image.

function
Object=half_left_process2(started_column,started_upper_bound,started_lower_bound,
Object,IM)
[K,L]=size(IM);
y=started_column;
pre_upper_bound=started_upper_bound;
pre_lower_bound=started_lower_bound;

% Obtain object boundaries on partitioned sub-images.
for q=1:round(L/2)-1
   column5=IM(:,y-q);
   column51=column5;
   column51(1:(pre_upper_bound),1)=0;
   column51((pre_lower_bound):K,1)=0;
   [a2 b2]=find(column51==max(column51(:)));
   position5=a2(1,1);
   array4(y-q,1)=position5;
   if column5(position5,1)>80

% Determine upper/lower boundaries on a sub-image.
[temp_lower_bound5,temp_upper_bound5]=get_bound2(column5,position5,IM);
      upper_bound5=temp_upper_bound5;
      lower_bound5=temp_lower_bound5;
```

```
        % Check the validity of boundaries.
    [n_upper_bound5,n_lower_bound5]=checking(upper_bound5,lower_bound5);
      upper_bound5=n_upper_bound5;
      lower_bound5=n_lower_bound5;
      array41(y-q,1)=upper_bound5;
      array42(y-q,1)=lower_bound5;

% Extract an object from a sub-image.
      Object(1:upper_bound5,y-q)=0;
      Object(lower_bound5:K,y-q)=0;
      pre_upper_bound=upper_bound5;
      pre_lower_bound=lower_bound5;
   else
      Object(:,1:y-q)=0;
      limit_swing_left=y-q;
      break;
   end
end
```

1.5 "get_object3" function.

```
% The following codes is used to obtain the third object (third types of cartilage) on
% an original image.
% Input: Image that the first and second object are  extracted.
% Output: The third object.
% To obtain an object on the right side of an image: use  "half_right_process3"
% function.
% To obtain an object on the left side of an image: use "half_left_process3" function.

function Object3 = get_object3(IM3)
Object3=IM3(150:300,70:150);
[p1,p2]=size(Object3);

% Determine a starting object sub-image and object boundaries on this  sub-image.
```

```
for i=round(p1/2)-20:round(p1/2)
   template=IM3(i,:);
   [a1 a2]=find(template==max(template(:)));
   position=a2(1,1);
   array1_1(i,1)=template(1,position);
   array1_2(i,1)=position;
end

% Determine a starting sub-image.
[b1 b2]=find(array1_1==max(array1_1(:)));
started_row=b1(1,1);
position=array1_2(started_row,1);
template1=IM3(started_row,:);
[a b]=find(template1==max(template1(:)));
position1=b(1,1);
if template1(1,position1)>80

% Determine upper/lower boundaries on a sub-image.
[temp_lower_bound1,temp_upper_bound1]=get_bound3(template1,position,IM3);
   upper_bound1=temp_upper_bound1;
   lower_bound1=temp_lower_bound1;

% Check the validity of boundaries.
[n_upper_bound1,n_lower_bound1]=checking(upper_bound1,lower_bound1);
   upper_bound1=n_upper_bound1;
   lower_bound1=n_lower_bound1;
   started_upper_bound=upper_bound1;
   started_lower_bound=lower_bound1;

% Extract an object from a sub-image.
   Object3(started_row,1:upper_bound1)=0;
   Object3(started_row,lower_bound1:p2)=0;
else
   Object3(round(p1/2),:)=0;
```

```
end

% Obtain an object on the right side of an image.
Object3=half_right_process3(started_row,started_upper_bound,started_lower_bound,O
bject3,IM3);

% Obtain an object on the left side of an image.
Object3=half_left_process3(started_row,started_upper_bound,started_lower_bound,Ob
ject3,IM3);
```

1.5.1 "half_right_process3" function.

```
% The following codes is used to obtain an object on the  right side of an image.
% Input: starting object sub-image, starting upper/lower % object boundaries, input
% image.
% Output: An object on the right side of an image.

function
Object3=half_right_process3(started_row,started_upper_bound,started_lower_bound,
Object3,IM3)

[p1,p2]=size(IM3);
pre_upper_bound3=started_upper_bound;
pre_lower_bound3=started_lower_bound;

% Obtain the object boundaries on partitioned sub-images.
for i=1:round(p1/2)
   template7=Object3(started_row+i,:);
   template7(1,1:pre_upper_bound3-2)=0;
   template7(1,pre_lower_bound3+2:p2)=0;
   [a7 b7]=find(template7==max(template7(:)));
   position7=b7(1,1);
   if template7(1,position7)>80
```

```
% Determine the upper/lower boundaries on a sub-image.
[temp_lower_bond7,temp_upper_bound7]=get_bound3_1(template7,position7,IM3);
      upper_bound7=temp_upper_bound7;
      lower_bound7=temp_lower_bound7;

% Check the validity of boundaries.
[n_upper_bound7,n_lower_bound7]=checking(upper_bound7,lower_bound7);
      upper_bound7=n_upper_bound7;
      lower_bound7=n_lower_bound7;
      pre_upper_bound2=upper_bound7;
      pre_lower_bound2=lower_bound7;

% Extract an object from a sub-image.
      Object3(started_row+i,1:upper_bound7)=0;
      Object3(started_row+i,lower_bound7:p2)=0;
      ori_position=position7;
   else
      Object3(started_row+i,:)=0;
      Object3(started_row+i:p1,:)=0;
      break;
   end
end
```

1.5.2 "half_left_process3" function.

```
% The following codes is used to obtain an object on the  left side of an image.
% Input: starting object sub-image, starting upper/lower  object boundaries, input
% image.
% Output: An object on the left side of an image.

function
Object3=half_left_process3(started_row,started_upper_bound,started_lower_bound,O
bject3,IM3)
[p1,p2]=size(IM3);
pre_upper_bound3=started_upper_bound;
```

```
pre_lower_bound3=started_lower_bound;

% Obtain object boundaries on partitioned sub-images.
for i=1:round(p1/2)
   if started_row-i<=0
      break;
   else
      template8=Object3(started_row-i,:);
   end
   template8(1,1:pre_upper_bound3-2)=0;
   template8(1,pre_lower_bound3+2:p2)=0;
   [a8 b8]=find(template8==max(template8(:)));
   position8=b8(1,1);
   if template8(1,position8)>80

% Determine upper/lower boundaries on a sub-image.
[temp_lower_bound8,temp_upper_bound8]=get_bound3(template8,position8,IM3);
      upper_bound8=temp_upper_bound8;
      lower_bound8=temp_lower_bound8;

% Check the validity of boundaries.
[n_upper_bound8,n_lower_bound8]=checking(upper_bound8,lower_bound8);
      upper_bound8=n_upper_bound8;
      lower_bound8=n_lower_bound8;
      pre_upper_bound3=upper_bound8;
      pre_lower_bound3=lower_bound8;

% Extract an object from a sub-image.
      Object3(started_row-i,1:upper_bound8)=0;
      Object3(started_row-i,lower_bound8:p2)=0;
   else
      Object3(started_row-i,:)=0;
      Object3(1:started_row-i,:)=0;
      break;
```

```
   end
end
```

1.6 "get_size" function.

```
% The following codes is used to calculate the number of  pixels of an object. It is
% also the area of an  object.
% Input: Object image that is extracted from original  image.
% Output: Number of pixels.

[M,N]=size(Object);
threshold=60;
Size=0;
for i=1:M
   for j=1:N
      if Object(i,j)>threshold
         Size=Size+1;              % Calculate the number of pixels
      end
   end
end
```

2. Artificial Neural Network Algorithms

2.1 Create a Neural Network Classifier:

```
% The following codes are used to create a network classifier.
% To obtain object input vectors set from a reference image and  relating target
% vector: use "getin"  function.
% To obtain background input vectors set from a reference image and  relating target
% vector: use  "getin2" function.
% To choose object input vectors and relating target vector for  network input: use
% "getset" function.
% To choose background input vectors and relating target vector for  network input:
% use "getset2"  function.
```

```
% Extract three types of cartilage (Object1, Object2, Object3) from a  reference
% image by using Bi-directional scanning Segmentations method.
[Object1, Object2, Object3] = main(im_original)
Object = Object1 + Object2 + Object3;

% Generate object input vectors set and object target vector.
[P,T1] = genin(Object,im_original);

% Generate background input vectors set and background target vector.
[Q,T2] = genin2(Object,im_original);

% Choose 200 object input vectors and target vectors for training  data.
[P1,t1] = getset(200,P);

% Choose 200 backgroud input vector and target vectors for training  data.
[Q1,t2] = getset2(200,Q);

% Create Network Target vector.
T = [t1 t2];

% Create Network Input vector.
Z = [P1 Q1];
Z = double(Z);

% Create Neural Network
net = newff(Z,T,40,{'logsig','logsig'},'trainscg');
net = init(net);
net.trainParam.goal = 10e-5;
net.trainParam.epochs = 20000;

% Training the network
net = train(net,Z,T);
NETWORK = net;
```

2.1.1 “getin” function.

```
% This function is used to generate object input vectors and relating target vector.
% Input: Object image, reference original image.
% Output: Object input vectors and relating target vectors

function [P,T] = genin(objectimage,im_original)
image = objectimage;
ori = im_original;
[K L] = size (image);
pad_ori_image=padarray(ori,[4 4],'replicate','pre');
pad_ori_image=padarray(pad_ori_image,[5 5],'replicate','post');
pad_obj_image=padarray(image,[4 4],'replicate','pre');
pad_obj_image=padarray(pad_obj_image,[5 5],'replicate','post');
N = 0;
for i = 5:K+4
   for j = 5: L+4
      if pad_obj_image(i,j)>0
         N = N + 1;
         windows = pad_ori_image(i-4:i+5,j-4:j+5);
         windows = windows';
         vector = reshape(windows,[1 100]);
         set(:,N) = vector;
         set2(:,N)= 1;
      end
   end
end
P = set;
T = set2;
```

2.1.2 “getin2” function.

```
% This function is used to generate background input vectors and  relating  target
% vector.
```

```
% Input: Object image, reference original image.
% Output: Background input vectors and relating target vectors

function [P,T] = genin2(objectimage,im_original)
image = objectimage;
ori = im_original;
[K L] = size (image);
pad_obj_image=padarray(image,[4 4],'replicate','pre');
pad_obj_image=padarray(pad_obj_image,[5 5],'replicate','post');
pad_ori_image=padarray(ori,[4 4],'replicate','pre');
pad_ori_image=padarray(pad_ori_image,[5 5],'replicate','post');
N = 0;
for i = 5:K+4
   for j = 5: L+4
      if N > 5000
         break;
      end
      if pad_obj_image(i,j)==0
         N = N + 1;
         windows = pad_ori_image(i-4:i+5,j-4:j+5);
         windows = windows';
         vector = reshape(windows,[1 100]);
         set(:,N) = vector;
         set2(:,N) = 0;
      end
   end
end
P = set;
T = set2;
```

2.1.3 “getset” function.

```
% This function is used to choose randomly 200 input vectors from  input vectors set
% and relating  target vectors.
```

```
% Input: object input vectors and relating target vectors set.
% Output: 200 input vectors and relating targetvectors.

function [out,target] = getset(number,inputset)
[M N] = size(inputset);
data = ceil(N.*rand(1,number));
for i = 1 : number
   if data(1,i)==0
      value = 1;
   else
      value = data(1,i);
   end
   out(:,i) = inputset(:,value);
   target(:,i) = 1;
end
```

2.1.4 "getset2" function.

```
% This function is used to choose randomly 200 input vectors from  background input
% vectors set and relating target vectors.
% Input: background input vectors and relating target vectors set.
% Output: 200 background input vectors and relating target vectors.

function [out,target] = getset(number,inputset)
[M N] = size(inputset);
data = ceil(N.*rand(1,number));
for i = 1 : number
   if data(1,i)==0
      value = 1;
   else
      value = data(1,i);
   end
   out(:,i) = inputset(:,value);
   target(:,i) = 1;
```

```
end
```

2.2 Network Classifier as Object Recognition Algorithms

```
% The following codes is used to extract an object from an original  image by using
% Network Classifier.
% Input: Original image.
% Output: Object image.
% To obtain an object on the right side of an original image: use
% "half_right_processNN" function.
% To obtain an object on the left side of an original image: use
% "half_left_processNN" function.
% To obtain an object on a sub-image: use "useNN" function.

function Object = runNN(original_image, Network)
IM = original_image.
[K,L]=size(IM);
reference_intensity=80;
Object = uint8(zeros(K,L));

% Determine a starting object sub-image and extract object from this  sub-image.
for i=160:256
   column=IM(:,i);
   [a1 a2]=find(column==max(column(:)));
   position=a1(1,1);
   array1_1(i,1)=column(position,1);
   array1_2(i,1)=position;
end
[b1 b2]=find(array1_1==max(array1_1(:)));
started_column=b1(1,1);
position=array1_2(started_column,1);

% Obtain an object on a sub-image.
Object = useNN(started_column,position,IM,IM_Network,Object);
```

```
[pre_up,pre_low,condition1] = bound_range(position,started_column,Object);
pre_up =  pre_up-100;
pre_low = pre_low+50;

% Obtain an object on the right side of an original image.
Object =
half_right_process_NN(Object,started_column,pre_up,pre_low,IM,IM_Network);

% Obtain an object on the left side of an original image.
Object=half_left_process_NN(started_column,pre_up,pre_low,Object,IM,IM_Network
);
```

2.2.1 "half_right_processNN" function.

```
% This function is used to obtain an object on the right side of an  image.
% Input: Starting object sub-image, original image, Network Classifier.
% Output: An object on the right side of an original image.

function Object =
half_right_processNN(Object,started_column,pre_up,pre_low,IM,IM_Network)
[K,L]=size(IM);

% Obtain an object on partitioned sub-images.
for y=(started_column+1):L
   column2=IM(:,y);
   column21=column2;
   column21(1:pre_up-10,1)=0;
   column21(pre_low+20:K,1)=0;
   [a2 b2]=find(column21==max(column21(:)));
   position2=a2(1,1);
   array2_0(y,1)=position2;
   if column2(position2,1)>80

% Obtain an object on a sub-image.
```

```
        Object=useNN(y,position2,IM,IM_Network,Object);
        [pre_up3,pre_low3,condition] = bound_range(position2,y,Object);
        CheckObject = Object;
        if condition == 0 && y > 320
            Object(:,y:L)=0;
            break;
        end
    else
        if y>=320
            Object(:,y:L)=0;
            break;
        else
            Object(:,y)=0;
        end
    end
end
```

2.2.2 “half_left_processNN” function.

```
% This function is used to obtain an object on the left side of an  image.
% Input: Starting object sub-image, original image, Network Classifier.
% Output: An object on the left side of an original image.

function
Object=half_left_processNN(started_column,pre_up,pre_low,Object,IM,IM_Network)
[K,L]=size(IM);
y=started_column;

% Obtain an object on partitioned sub-images.
for q=1:round(L/2)-1
    column5=IM(:,y-q);
    column51=column5;
    column51(1:pre_up-20,1)=0;
    column51(pre_low+20:K,1)=0;
```

```
   [a2 b2]=find(column51==max(column51(:)));
   position5=a2(1,1);
   array4(y-q,1)=position5;
   if column5(position5,1)>80

% Obtain an object on a sub-image.
      Object=useNN(y-q,position5,IM,IM_Network,Object);
    [pre_up,pre_low,condition]=bound_range(position5,y-q,Object);
      if condition == 0
         Object(:,y-q:L)=0;
         break;
      end
   else
      Object(:,y-q)=0;
      limit_swing_left=y-q;
      break;
   end
end
```

2.2.3 "useNN" function.

```
% This function is used to obtain an object on a sub-image using  Network Classifier.
% Input: Sub-image, Network Classifier.
% Output: An object on a sub-image.

function Object = useNN(column,position,IM,IM_Network,Object)
try
   va1 = 1;
   va2 = 0;
   threshold = 0.9;
   TestIM = IM;
   if 250<column&&column<280
      UP = 25;
      DOWN = 50;
```

```
elseif column<=250
    UP = 150;
    DOWN = 100;
elseif column>=280
    UP = 175;
    DOWN = 175;
end
for k=1:UP
    if position-k<=0
        win = IM(1:10,1:10);
    else
        win = IM((position-k)-4:(position-k)+5,column-4:column+5);
    end
    win = win';
    win = reshape(win,[100 1]);
    win = double(win);
    OUTPUT = sim(IM_Network,win);
    TstOutput=real(OUTPUT>threshold);
    if TstOutput == va1
        Object(position-k,column)=IM(position-k,column);
    elseif TstOutput == va2
        Object(position-k,column) = 0;
    end
end
for k=1:DOWN
    if position+k+5>512
        break;
    end
    win = IM((position+k)-4:(position+k)+5,column-4:column+5);
    win = win';
    win = reshape(win,[100 1]);
    win = double(win);
    OUTPUT = sim(IM_Network,win);
    TstOutput=real(OUTPUT>threshold);
```

```
        if TstOutput == va1
         Object(position+k,column)=IM(position+k,column);
        elseif TstOutput == va2
           Object(position+k,column) = 0;
        end
    end
end
```

3. Active Contour Models Algorithms

```
% This function is used to obtain an object on an original image by using Active
% Contour Models.
% Input : Original image.
% Output: An object on original image.

function Snake(im_original)
IM = im_original;
% Initializing Contour V
load intcontourV
CV = contourV;
CV(3,:) = 1;
[P Q] = size(CV);
SUM = sum(CV(3,:));
while SUM>20
    for i = 2:Q-1
        if CV(3,i)==1
            % Create windows for vector Vi
            Vi_row = CV(1,i);
            Vi_col = CV(2,i);
            P11 = [Vi_row-2 Vi_col-2]';
            P12 = [Vi_row-2 Vi_col-1]';
            P13 = [Vi_row-2 Vi_col]';
            P14 = [Vi_row-2 Vi_col+1]';
            P15 = [Vi_row-2 Vi_col+2]';
```

```
P21 = [Vi_row-1 Vi_col-2]';
P22 = [Vi_row-1 Vi_col-1]';
P23 = [Vi_row-1 Vi_col]';
P24 = [Vi_row-1 Vi_col+1]';
P25 = [Vi_row-1 Vi_col+2]';
P31 = [Vi_row Vi_col-2]';
P32 = [Vi_row Vi_col-1]';
P33 = [Vi_row Vi_col]';
P34 = [Vi_row Vi_col+1]';
P35 = [Vi_row Vi_col+2]';
P41 = [Vi_row+1 Vi_col-2]';
P42 = [Vi_row+1 Vi_col-1]';
P43 = [Vi_row+1 Vi_col]';
P44 = [Vi_row+1 Vi_col+1]';
P45 = [Vi_row+1 Vi_col+2]';
P51 = [Vi_row+2 Vi_col-2]';
P52 = [Vi_row+2 Vi_col-1]';
P53 = [Vi_row+2 Vi_col]';
P54 = [Vi_row+2 Vi_col+1]';
P55 = [Vi_row+2 Vi_col+2]';

% Calculate normalization factor l(V)
[m n] = size(CV);
for i1 = 1 : n-1
   lv(n,1) = ((abs(CV(1,i1+1) - CV(1,i1)))^2 + (abs(CV(2,i1+1)-CV(2,i1)))^2) ;
end
LV = sum(lv(:))/n;


% Calculate Continutity Energy
V = 0.5*(CV(1:2,i-1)+CV(1:2,i));
e11 = (1/LV)*((P11(1,1)-V(1,1))^2+(P11(2,1)-V(2,1))^2);
e12 = (1/LV)*((P12(1,1)-V(1,1))^2+(P12(2,1)-V(2,1))^2);
e13 = (1/LV)*((P13(1,1)-V(1,1))^2+(P13(2,1)-V(2,1))^2);
```

```
e14 = (1/LV)*((P14(1,1)-V(1,1))^2+(P14(2,1)-V(2,1))^2);
e15 = (1/LV)*((P15(1,1)-V(1,1))^2+(P15(2,1)-V(2,1))^2);
e21 = (1/LV)*((P21(1,1)-V(1,1))^2+(P21(2,1)-V(2,1))^2);
e22 = (1/LV)*((P22(1,1)-V(1,1))^2+(P22(2,1)-V(2,1))^2);
e23 = (1/LV)*((P23(1,1)-V(1,1))^2+(P23(2,1)-V(2,1))^2);
e24 = (1/LV)*((P24(1,1)-V(1,1))^2+(P24(2,1)-V(2,1))^2);
e25 = (1/LV)*((P25(1,1)-V(1,1))^2+(P25(2,1)-V(2,1))^2);
e31 = (1/LV)*((P31(1,1)-V(1,1))^2+(P31(2,1)-V(2,1))^2);
e32 = (1/LV)*((P32(1,1)-V(1,1))^2+(P32(2,1)-V(2,1))^2);
e33 = (1/LV)*((P33(1,1)-V(1,1))^2+(P33(2,1)-V(2,1))^2);
e34 = (1/LV)*((P34(1,1)-V(1,1))^2+(P34(2,1)-V(2,1))^2);
e35 = (1/LV)*((P35(1,1)-V(1,1))^2+(P35(2,1)-V(2,1))^2);
e41 = (1/LV)*((P41(1,1)-V(1,1))^2+(P41(2,1)-V(2,1))^2);
e42 = (1/LV)*((P42(1,1)-V(1,1))^2+(P42(2,1)-V(2,1))^2);
e43 = (1/LV)*((P43(1,1)-V(1,1))^2+(P43(2,1)-V(2,1))^2);
e44 = (1/LV)*((P44(1,1)-V(1,1))^2+(P44(2,1)-V(2,1))^2);
e45 = (1/LV)*((P45(1,1)-V(1,1))^2+(P45(2,1)-V(2,1))^2);
e51 = (1/LV)*((P51(1,1)-V(1,1))^2+(P51(2,1)-V(2,1))^2);
e52 = (1/LV)*((P52(1,1)-V(1,1))^2+(P52(2,1)-V(2,1))^2);
e53 = (1/LV)*((P53(1,1)-V(1,1))^2+(P53(2,1)-V(2,1))^2);
e54 = (1/LV)*((P54(1,1)-V(1,1))^2+(P54(2,1)-V(2,1))^2);
e55 = (1/LV)*((P55(1,1)-V(1,1))^2+(P55(2,1)-V(2,1))^2);
CE = [e11 e12 e13 e14 e15;e21 e22 e23 e24 e25;e31 e32 e33 e34 e35;e41 e42
e43 e44 e45;e51 e52 e53 e54 e55];

%Calculate Ballon Energy
t1 = ((1/sqrt((CV(1,i)-CV(1,i-1))^2+(CV(2,i)-CV(2,i-1))^2))*(CV(1:2,i)-
CV(1:2,i-1)));
t2= ((1/sqrt((CV(1,i+1)-CV(1,i))^2+(CV(2,i+1)-CV(2,i))^2))*(CV(1:2,i+1)-
CV(1:2,i)));
ti = t1 + t2;
ni = rot90(ti);
eb11 =dot(ni,(CV(1:2,i)-P11));
eb12 =dot(ni,(CV(1:2,i)-P12));
```

```
eb13 =dot(ni,(CV(1:2,i)-P13));
eb14 =dot(ni,(CV(1:2,i)-P14));
eb15 =dot(ni,(CV(1:2,i)-P15));
eb21 =dot(ni,(CV(1:2,i)-P21));
eb22 =dot(ni,(CV(1:2,i)-P22));
eb23 =dot(ni,(CV(1:2,i)-P23));
eb24 =dot(ni,(CV(1:2,i)-P24));
eb25 =dot(ni,(CV(1:2,i)-P25));
eb31 =dot(ni,(CV(1:2,i)-P31));
eb32 =dot(ni,(CV(1:2,i)-P32));
eb33 =dot(ni,(CV(1:2,i)-P33));
eb34 =dot(ni,(CV(1:2,i)-P34));
eb35 =dot(ni,(CV(1:2,i)-P35));
eb41 =dot(ni,(CV(1:2,i)-P41));
eb42 =dot(ni,(CV(1:2,i)-P42));
eb43 =dot(ni,(CV(1:2,i)-P43));
eb44 =dot(ni,(CV(1:2,i)-P44));
eb45 =dot(ni,(CV(1:2,i)-P45));
eb51 =dot(ni,(CV(1:2,i)-P51));
eb52 =dot(ni,(CV(1:2,i)-P52));
eb53 =dot(ni,(CV(1:2,i)-P53));
eb54 =dot(ni,(CV(1:2,i)-P54));
eb55 =dot(ni,(CV(1:2,i)-P55));

BE = [eb11 eb12 eb13 eb14 eb15;eb21 eb22 eb23 eb24 eb25;eb31 eb32 eb33
eb34 eb35;eb41 eb42 eb43 eb44 eb45;eb51 eb52 eb53 eb54 eb55];
% Calculate Internal Engery
IE = CE + BE;

% Calculate External Energy using NN
load neural_network_version2.mat
Vi_row = round(Vi_row);
Vi_col = round(Vi_col);
win11 = IM((Vi_row-2)-4:(Vi_row-2)+5,(Vi_col-2)-4:(Vi_col-2)+5);
```

```
win11 = win11';
win11 = reshape(win11,[100 1]);
win11 = double(win11);
OUTPUT11 = sim(IM_Network,win11);
win12 = IM((Vi_row-2)-4:(Vi_row-2)+5,(Vi_col-1)-4:(Vi_col-1)+5);
win12 = win12';
win12 = reshape(win12,[100 1]);
win12 = double(win12);
OUTPUT12 = sim(IM_Network,win12);
win13 = IM((Vi_row-2)-4:(Vi_row-2)+5,(Vi_col)-4:(Vi_col)+5);
win13 = win13';
win13 = reshape(win13,[100 1]);
win13 = double(win13);
OUTPUT13 = sim(IM_Network,win13);
win14 = IM((Vi_row-2)-4:(Vi_row-2)+5,(Vi_col+1)-4:(Vi_col+1)+5);
win14 = win14';
win14 = reshape(win14,[100 1]);
win14 = double(win14);
OUTPUT14 = sim(IM_Network,win14);
win15 = IM((Vi_row-2)-4:(Vi_row-2)+5,(Vi_col+2)-4:(Vi_col+2)+5);
win15 = win15';
win15 = reshape(win15,[100 1]);
win15 = double(win15);
OUTPUT15 = sim(IM_Network,win15);
win21 = IM((Vi_row-1)-4:(Vi_row-1)+5,(Vi_col-2)-4:(Vi_col-2)+5);
win21 = win21';
win21 = reshape(win21,[100 1]);
win21 = double(win21);
OUTPUT21 = sim(IM_Network,win21);
win22 = IM((Vi_row-1)-4:(Vi_row-1)+5,(Vi_col-1)-4:(Vi_col-1)+5);
win22 = win22';
win22 = reshape(win22,[100 1]);
win22 = double(win22);
OUTPUT22 = sim(IM_Network,win22);
```

```
win23 = IM((Vi_row-1)-4:(Vi_row-1)+5,(Vi_col)-4:(Vi_col)+5);
win23 = win23';
win23 = reshape(win23,[100 1]);
win23 = double(win23);
OUTPUT23 = sim(IM_Network,win23);
win24 = IM((Vi_row-1)-4:(Vi_row-1)+5,(Vi_col+1)-4:(Vi_col+1)+5);
win24 = win24';
win24 = reshape(win24,[100 1]);
win24 = double(win24);
OUTPUT24 = sim(IM_Network,win24);
win25 = IM((Vi_row-1)-4:(Vi_row-1)+5,(Vi_col+2)-4:(Vi_col+2)+5);
win25 = win25';
win25 = reshape(win25,[100 1]);
win25 = double(win25);
OUTPUT25 = sim(IM_Network,win25);
win31 = IM((Vi_row)-4:(Vi_row)+5,(Vi_col-2)-4:(Vi_col-2)+5);
win31 = win31';
win31 = reshape(win31,[100 1]);
win31 = double(win31);
OUTPUT31 = sim(IM_Network,win31);
win32 = IM((Vi_row)-4:(Vi_row)+5,(Vi_col-1)-4:(Vi_col-1)+5);
win32 = win32';
win32 = reshape(win32,[100 1]);
win32 = double(win32);
OUTPUT32 = sim(IM_Network,win32);
win33 = IM((Vi_row)-4:(Vi_row)+5,(Vi_col)-4:(Vi_col)+5);
win33 = win33';
win33 = reshape(win33,[100 1]);
win33 = double(win33);
OUTPUT33 = sim(IM_Network,win33);
win34 = IM((Vi_row)-4:(Vi_row)+5,(Vi_col+1)-4:(Vi_col+1)+5);
win34 = win34';
win34 = reshape(win34,[100 1]);
win34 = double(win34);
```

```
OUTPUT34 = sim(IM_Network,win34);
win35 = IM((Vi_row)-4:(Vi_row)+5,(Vi_col+2)-4:(Vi_col+2)+5);
win35 = win35';
win35 = reshape(win35,[100 1]);
win35 = double(win35);
OUTPUT35 = sim(IM_Network,win35);
win41 = IM((Vi_row+1)-4:(Vi_row+1)+5,(Vi_col-2)-4:(Vi_col-2)+5);
win41 = win41';
win41 = reshape(win41,[100 1]);
win41 = double(win41);
OUTPUT41 = sim(IM_Network,win41);
win42 = IM((Vi_row+1)-4:(Vi_row+1)+5,(Vi_col-1)-4:(Vi_col-1)+5);
win42 = win42';
win42 = reshape(win42,[100 1]);
win42 = double(win42);
OUTPUT42 = sim(IM_Network,win42);
win43 = IM((Vi_row+1)-4:(Vi_row+1)+5,(Vi_col)-4:(Vi_col)+5);
win43 = win43';
win43 = reshape(win43,[100 1]);
win43 = double(win43);
OUTPUT43 = sim(IM_Network,win43);
win44 = IM((Vi_row+1)-4:(Vi_row+1)+5,(Vi_col+1)-4:(Vi_col+1)+5);
win44 = win44';
win44 = reshape(win44,[100 1]);
win44 = double(win44);
OUTPUT44 = sim(IM_Network,win44);
win45 = IM((Vi_row+1)-4:(Vi_row+1)+5,(Vi_col+2)-4:(Vi_col+2)+5);
win45 = win45';
win45 = reshape(win45,[100 1]);
win45 = double(win45);
OUTPUT45 = sim(IM_Network,win45);
win51 = IM((Vi_row+2)-4:(Vi_row+2)+5,(Vi_col-2)-4:(Vi_col-2)+5);
win51 = win51';
win51 = reshape(win51,[100 1]);
```

```
win51 = double(win51);
OUTPUT51 = sim(IM_Network,win51);
win52 = IM((Vi_row+2)-4:(Vi_row+2)+5,(Vi_col-1)-4:(Vi_col-1)+5);
win52 = win52';
win52 = reshape(win52,[100 1]);
win52 = double(win52);
OUTPUT52 = sim(IM_Network,win52);
win53 = IM((Vi_row+2)-4:(Vi_row+2)+5,(Vi_col)-4:(Vi_col)+5);
win53 = win53';
win53 = reshape(win53,[100 1]);
win53 = double(win53);
OUTPUT53 = sim(IM_Network,win53);
win54 = IM((Vi_row+2)-4:(Vi_row+2)+5,(Vi_col+1)-4:(Vi_col+1)+5);
win54 = win54';
win54 = reshape(win54,[100 1]);
win54 = double(win54);
OUTPUT54 = sim(IM_Network,win54);
win55 = IM((Vi_row+2)-4:(Vi_row+2)+5,(Vi_col+2)-4:(Vi_col+2)+5);
win55 = win55';
win55 = reshape(win55,[100 1]);
win55 = double(win55);
OUTPUT55 = sim(IM_Network,win55);

EE = [OUTPUT11 OUTPUT12 OUTPUT13 OUTPUT14
OUTPUT15;OUTPUT21 OUTPUT22 OUTPUT23 OUTPUT24
OUTPUT25;OUTPUT31 OUTPUT32 OUTPUT33 OUTPUT34
OUTPUT35;OUTPUT41 OUTPUT42 OUTPUT43 OUTPUT44
OUTPUT45;OUTPUT51 OUTPUT52 OUTPUT53 OUTPUT54 OUTPUT55];
E = EE + IE;
[a b]=find(E==min(E(:)));

if a == 1
   flag = 1;
   new_Vi_row = Vi_row - 2;
```

```
        if b == 1
            new_Vi_col = Vi_col - 2;
        elseif b == 2
            new_Vi_col = Vi_col - 1;
        elseif b == 3
            new_Vi_col = Vi_col;
        elseif b == 4
            new_Vi_col = Vi_col + 1;
        elseif b == 5
            new_Vi_col = Vi_col + 2;
        end
    elseif a == 2
        flag = 1;
        new_Vi_row = Vi_row - 1;
        if b == 1
            new_Vi_col= Vi_col - 2;
        elseif b == 2
            new_Vi_col= Vi_col - 1;
        elseif b == 3
            new_Vi_col= Vi_col;
        elseif b == 4
            new_Vi_col= Vi_col + 1;
        elseif b == 5
            new_Vi_col= Vi_col + 2;
        end
    elseif a == 3
        flag = 1;
        new_Vi_row = Vi_row;
        if b == 1
            new_Vi_col= Vi_col - 2;
        elseif b == 2
            new_Vi_col= Vi_col - 1;
        elseif b == 3
            flag=0;
```

```
            new_Vi_col = Vi_col;
        elseif b == 4
            new_Vi_col= Vi_col + 1;
        elseif b == 5
            new_Vi_col= Vi_col + 2;
        end
    elseif a == 4
        flag = 1;
        new_Vi_row = Vi_row + 1;
        if b == 1
            new_Vi_col= Vi_col - 2;
        elseif b == 2
            new_Vi_col= Vi_col - 1;
        elseif b == 3
            new_Vi_col= Vi_col;
        elseif b == 4
            new_Vi_col= Vi_col + 1;
        elseif b == 5
            new_Vi_col= Vi_col + 2;
        end
    elseif a == 5
        flag = 1;
        new_Vi_row = Vi_row + 2;
        if b == 1
            new_Vi_col= Vi_col - 2;
        elseif b == 2
            new_Vi_col= Vi_col - 1;
        elseif b == 3
            new_Vi_col= Vi_col;
        elseif b == 4
            new_Vi_col= Vi_col + 1;
        elseif b == 5
            new_Vi_col= Vi_col + 2;
        end
```

```
        end
        CV(:,i) = [new_Vi_row new_Vi_col flag]';
    end
    SUM = sum(CV(3,:));
end
```