University of the Philippines Manila

College of Arts and Sciences

Department of Physical Sciences and Mathematics

Lung Nodule Detector and Classifier Tool

A SPECIAL PROBLEM

In Partial Fulfillment for the Degree in

Bachelor of Science in Computer Science

Submitted by:

Jhesed D. Tacadena

April 2013

ACCEPTANCE SHEET

The Special Problem entitled **"Lung Nodule Detector and Classifier Tool"** prepared and submitted by Jhesed D. Tacadena in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Geoffrey A. Solano, Ph.D.
Adviser

EXAMINERS:

		Approved	Disapproved
1.	Gregorio B. Baes, Ph.D. (candidate)		
2.	Avegail D. Carpio, M.S.		
3.	Richard Bryann L. Chua, Ph.D. (candidate)		
4.	Aldrich Colin K. Co, M.S. (candidate)		
5.	Perlita E. Gasmen M.S. (candidate)		
6.	Vincent Peter C. Magboo, M.D., M.S.		
7.	Ma. Sheila A. Magboo, M.S		
8.	Bernie B. Terrado, M.S. (candidate)		

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Avegail D. Carpio, M.S.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences and Mathematics

Marcelina B. Lirazan, Ph.D.
Chair
Department of Physical Sciences and Mathematics

Alex C. Gonzaga, Ph.D.
Dean
College of Arts and Sciences

Abstract

Lung cancer is the number one cause of death in the Philippines and in the world. Multi-detector CT scanners provide opportunity to examine thin-section CT images, which improves reader detection of focal findings and characterization of nodules. However, the sensitivity of manual detection of cancerous and non-cancerous lung nodules is reported to be 70-75% only. Lung Nodule Detector and Classifier Tool is decision support software that aims to aid the health professionals in detecting and classifying lung nodules. LNDCT uses algorithms which include diffusion, binarization, wavelet edge detection, and morphological operations to automatically segment the nodules. The tool uses LibSVM, an open source support vector machine software in classifying malignant from benign nodules. Three testing sets were used to test the accuracy of detection and classification using LNDCT. Set A contains lung images where most of the sizes of the cancerous nodules are greater than 200 pixels. Set B contains a combination of different sizes of cancerous nodules. Set C contains 400 training and 200 testing random data which came from pre-extracted features computed using LNDCT. The accuracy of the system is reported to be 93.87 %, 81.33 %, and 88.57% for sets A, B and C respectively. LNDCT is a tool which can help health professionals in detection, classification, and feature computations of lung nodules.

Keywords: Computer Aided Diagnosis, Lung Nodule Detection, Wavelets, Morpological Operations, Support Vector Machine

Table of Contents

I. Introduction

A. Background of the Study

Last 2008, there were an estimated 12.7 million cancer cases worldwide [1]. It is the leading cause of death accounting for 7.6 million deaths [2]. The number is expected to increase to 21 million cases [1] and 13.1 death cases [2] in 2030. Lung cancer is the number one cause of cancer deaths worldwide accounting for 1.37 million.

According to Dr. Dennis Tudtud, the president of Philippine Society of Medical Oncology, lung cancer is number one cause of cancer deaths worldwide and also in the Philippines [3]. Dr. Claire Soliman, head of breast section of the Department of Medical Oncology in St. Luke's Medical Center, stated that the survival rate from lung cancer is only 15 percent as compared to 89 percent in breast cancer [4]. But even though it is the deadliest, only 15% of lung cancer cases were diagnosed during its early stages [3].

The role of technology in providing fast and accurate diagnosis in the medical field is very important. This includes early detection of developing cancer in an individual. Medical image processing tests help health professionals analyze different areas in the human body that might be affected by cancer cells. These result to the knowledge of how far the disease have already spread and what particular treatment will be the most effective.

Multi-detector CT scanners provide opportunity to examine thin-section CT images. This improves reader detection of focal findings and characterization of these findings as nodules. The process is described as follows: The technologist will position the patient on the CT scan examination table. Then, the table will move quickly through the scanner to determine the correct starting position for the scans. The table will move slowly through the machine as actual CT scanning is performed. With CT scanning, numerous x-ray beams and a set of electronic x-ray detectors rotate around the patient. As an illustration, CT imaging is like looking into a loaf of bread and cutting it to thin slices. When the image slices are reassembled by computer software, the result is a very detailed multi-dimensional view of the body's interior. Many algorithms were developed to make CT scan process faster and cheaper in memory space. Leading CT scanner vendors also embedded some in their machines. However, these algorithms were focused on that purpose. These are not capable of automated segmentation and classification of lung nodules. The overall sensitivity of manual detection of pulmonary nodules has been reported to be 70%-75%. This sensitivity is lower for smaller pulmonary nodules related to volume averaging. Fatigue also affects the reader when the number of images to be examined increases by 5-fold when 1-mm-section images is used instead of 5-mm section images [5].

Wavelets are mathematical functions that cut up data into different frequency components and then study each component with a resolution matched to its scale [6]. Wavelets also help us perform multi-resolution analysis of the images [7].

Support vector machine (SVM) is a machine learning technique which is used as a classification tool. It uses kernel function, which acts upon the input data; final summation with an activation function gives the final classification result [8].

In this study, we use wavelet based edge detection and morphological operations to detect and segment lung nodules. We use width-height ratio to initially eliminate non-potential nodules. We compute for mean and standard deviation and use it as training features for SVM. We chose mean and SD as training features because these vary their values between the cancerous and non-cancerous images [8]. Based from the model generated by the SVM training process, we predict which nodules are malignant and which are benign, with Radial Basis Function as the kernel. The reported class performance and sensitivity of using these algorithms together is 92.86%, with an error rate of 0.0714 [8]. The system is able to generate reports for easier interpretation to doctors.

We use lung data from Lung Images Database Consortium (LIDC) to train and test our system. It is a free database of lung images that aims to provide support for computer aided detection techniques for lung sicknesses. The database contains 1018 thoracic CT scans and associated XML-based annotations to stimulate the development of Computer Aided Diagnosis methods for lung nodule detection, classification, and quantitative assessment. The detailed content, together with the types of images, brands of the scanners, and resolutions of CT scan images in the database can be found in chapter three.

B. Statement of the Problem

The best treatment to lung cancer can be applied during its early stages. With the increased number of sections per scan produced with thin-slice multi-detector CT, it will take a lot of time and effort for the health professionals to distinguish and classify the nodules. It is crucial to develop decision support software that will assist them in this difficult task. There is a need for a tool to aid analysis by:

1. Detecting edges in the lungs using multi-scale wavelets.

2. Performing morphological operations on the image and using the detected edges to segment potential nodules.

3. Using SVM in classifying malignant from benign lung images.

4. Generating a report on the processed images.

C. Objectives

This study aims to provide decision support software for health professionals that will help them quickly visualize lung nodules from CT scan images. The system will help the professionals in determining which nodules are cancerous or not by using SVM classifier algorithm. The tool is able to:

1. Import lungs CT scan images (in DICOM format).

2. Obtain input data by:

   a.) Asking the health professional to load a folder which contains the lung images.

   b.) Allowing the health professional to input optional data which include scale, wavelet type (for wavelet transform), and training data (for SVM). If these values are not provided, the default values will be used.

3. Process data by:

   a.) Using multi-scale wavelet transform to detect edges from the lung images.

   b.) Performing morphological operations to the images which include dilation, erosion, and region filling, and using the edge from (a) to segment potential nodules. Using width-height ratio value to immediately remove non-potential nodules.

   c.) Showing classification results using SVM and trying to determine which are cancerous and not.

   d.) Showing the user the segmented potential nodules obtained using the processes mentioned before.

4. Allowing the health professional to input comments/prescriptions which will be used when the results are exported to PDF file.

5. Provide tutorial on the usage of the system.

D. Significance of the Study

It is hard to cure cancer during its latter stages. The tool aims to help in early detection of potential nodules. The implemented decision support software will help health professionals in judging whether a patient has cancerous or non-cancerous nodule(s) in lungs. Though it will still depend on the doctor's judgment, it will help them visualize the location and size of the nodule based on wavelet and SVM algorithms.

E. Scope and Limitations

1. Even though the tool will help health professionals, it does not intend to replace them in deciding whether a nodule is cancerous or not.

2. The tool uses thin CT scan images from Lung Image Database Consortium. These images consist of scans from seven different GE Medical Systems Light Speed scanner models, four different Philips Brilliance scanner models, five different Siemens Definition, Emotion, and Sensation scanner models, and Toshiba Aquilion scanners. Each scan consists of about 100 to 280 images, with tube current of above 100mA. Thick CT scan slices which are about 20 images per scan will not be included.

3. The tool is limited to the following operations. Other operations aside from these will not be included: use of wavelets for edge detection; morphological operations which include

dilation, erosion and region filling and use the result with the edge detected image for segmentation; diffusion for pre-processing; width-height ratio for immediate elimination of non-potential nodules; and SVM with Radial Basis Function (RBF) as the kernel for classification whether the segmented nodule is cancerous or not.

4. The tool does not keep a database of the processed CT scan images and related data because of high memory issues. Processed files will not be remembered after closing the tool. However, the tool has the ability to export the results generated to a PDF file for reference.

5. The tool processes an image of DICOM format, using DCM4CHE 1.4 java library.

6. The tool uses LibSVM [48], an open source library for support vector machines in classifying malignant and benign nodules.

F. Assumptions

1. The user is a health professional.

2. The lung image to be loaded in the system is in DICOM format.

II. Review of Related Literature

A. Lung nodule analysis

There are many studies that tried to develop computer aided detection of lung nodules.

Zhao, Gamsu, Gnserg, Jiang, and Schwartz used morphological operations and local density maximum algorithm to extract potential lung nodules. The method was applied to the detecton of computer simulated small lung nodules. Their method achieved 84.2% sensitivity with five false positive results per scan [9].

Korfiatis, Skiadopoulos, Sakellaropoulos, Kalogerpoulou, and Costaridou combined 2D wavelet edge highlighting, 3D morphological closing, and 3D thresholding for lung segmentation. They concluded that using wavelet is a good pre-processing step for segmentation [10].

Ye, Lin, Dehmeshki, Slabaugh, and Beddoe used fuzzy threshholding, volumetric shape index map, anti-geometric diffusion, modified expectation maximization, rule-based filtering and support vector machine to detect solid and ground grass opacity nodules in lungs. Their proposed method were trained and validated on a clinical dataset of 108 thoracic CT scans which contains 220 nodules (185 solid and 35 GGO). The experimental results indicate 90.2% average detection rate, with 8.2 false positive per scan [11].

Sousa, Silva, Paiva, and Nunes proposed lung nodule detection and classification in six steps: thorax extraction, lung extraction, lung reconstruction, structures extraction, tubular structures elimination, and false positive reduction. They used support vector machine in determining the lung nodules from normal lung structures. They used radial basis function as kernel to SVM. The classification sensitivity achieved was 84.84% [12].

Khaz and Kavitha proposed a new contextual clustering method combined with fuzzy logic for nodule segmentation. Their method was compared to existing segmentation algorithms like Prewitt, Robertz, Sobel, Log, and Zerocross. Its performance provided better segmentation accuracy of 84.7% [13].

Anitha and Sridhar proposed two stage approaches which includes modified adaptive fissure sweep and adaptive thresholding to segment lung lobes and nodules from CT images. The analysis is made on 20 set of images and compared to manual segmentation [14].

Karthikeyan and Ramadoss used fuzzy c-means clustering to segment the lungs. They performed cleaning to remove air, noise and airways. They also used a sequence of morphological operations to smooth the irregular boundary. They concluded that their algorithm works on large number of different cases [15].

Gomathi and Thangaraj’s method consisted of five phases: the extraction of lung region from CT images, the segmentation of lung region, the extraction of feature, and the classification of based on occurrence of cancer. They used Modified Fuzzy Possibilistic C-mean algorithm for segmentation and extreme learning machine in classification of lung cancer [16].

Tidke, Vrishali, and Chakkarwar performed denoising as a pre-processing step. They performed thresholding and morphological operations for segmentation. They extracted features using GLCM. Finally, they used SVM to classify benign from malignant nodules [17].

Aarthy and Ragupathy used wavelets, morphological operations, and support vector machine to detect and classify lung nodules. Their research included 28 lung images and resulted to 92.86% class performance with a sensitivity of 92.86% and error rate of 0.0714 [8].

B. Other General Applications

Wavelet is a very useful mathematical tool which can be applied to different fields. Compared to the previous Fourier Transform which is only based on sine and cosine functions, it can have infinitely many basis functions. Its window size may also vary.

It has many applications. One of these is image processing. Oh, Kaneko, and Makinouchi used both wavelets and SOM to extract features, classify, and retrieve images in a still image database [18]. Zhi and Ming used wavelets to extract face features of a person and use it for identification [19].

Wavelets can also be used for the observation of the equipment's state. Shi, Zhang, and Wei used wavelets alongside with SOM to monitor the present condition of bearing in rotary machines. These are also used for fault detection of rolling element bearings [20]. Kang and Birtwhistle used wavelets to analyze the vibration of signals in power switching equipment to observe its situation [21]. Tao, Wang, and Fan in [22] and Yang, Yuan, and Si in [23] also used wavelets for fault detection.

Wavelets can also be used for sound processing. Seling, Turunen, and Tanttu used these to recognize bird sounds [24]. Hao, Campana, and Keogh used wavelets to produce a visual sound fingerprint based on the unique texture of the sound being produced by animals. Their experiment only involved insects and frogs. But they expect to use their research for all animals [25].

Wavelets can also be used for detecting missing data such as rainfall. Nourani, Parhizkar, Bagnaham, and Sharghi used these with SOM to analyze the rainfall-runoff data [26]. These were used by Janahira and Win to detect water regions by just having the Landsat satellite image [27].

One of the most important applications of wavelets is when it is being used for medical purposes. Semleer, Dettori, and Furst used these to classify tissues in computed tomography based on its texture [28]. Padma and Sukanesh based their research in wavelets for automatic segmentation of brain tumor in CT images using optimal statistical texture features [29]. Tian and Linan used wavelets with SOM for MR brain image segmentation [30]. Kaneko, Gotho, Iseri, Takeshita, Ohki, and Sueda used wavelet transform for QRS complex analysis [31]. Lee, Abibullaev, Kang, Yunhee, and An used these with SOM to analyze attention deficit hyperactivity disorder in EEG [32]. Tawade and Warpe used discrete wavelet transform for epilepsy disorder detection [33]. Vasantha, Bharathi, and Dhamodharan used these to extract, classify, and select medical image features [34]. Kharrat, Gasmi, Messoud, Benamrane, and Abid also classified medical images using wavelets but with the addition of using the optimal feature extraction algorithm and supervised classifier technique [35]. Baranidharan and Ghosh researched on two dimensional image classification neural networks for medical images using wavelets [36]. Tadejko and Rakowski used wavelet and SVM to extract and classify heartbeat [37]. Sun and Dond used discrete wavelet transform to classify tissues using gene expression data [38]. Malek, Sebri, Mabrouk, Torki and Tourki used wavelet with GVF snake segmentation and fuzzy classification to automate breast cancer diagnosis [39].

Support Vector is a machine learning technique which is used as a classification tool. It uses kernel function, which acts upon the input data; final summation with an activation function gives the final classification result [8]. It has many applications.

Samb, Camara, Ndiaye, Slimani, and Esseghir used it for feature selection and classification [40]. Hejazi and Singh used SVM to detect credit data fraud [41]. Okwuashi, McConchie, Nwilo, Isong, Eyoh, Nwanekezie, et al. used SVM to predict future land use change [42]. Venkatesan and Devi used the tool for disease classification [43]. Mandle, Jain, and Shrivastava used SVM for protein structure recognition [44]. Pan, Shen, and Shen used SVM for speech emotion recognition [45].

III. Theoretical Framework

A. Decision Support Software

Decision support software consists of systems and subsystems that help people make decisions based on data that is culled from a wide range of sources [46]. When used in a clinical setting, it will help health professionals in deciding whether a person has a particular sickness (i.e. determining whether a person has cancer).

B. Wavelets

Amara Graps gave us a detailed introduction to wavelets in [6]. Wavelets are mathematical functions that cut up data into different frequency components, and then study each component with a resolution matched to its scale. They have advantages over traditional Fourier methods in analyzing physical situations where the signal contains discontinuities and sharp spikes. The fundamental idea behind wavelets is to analyze according to its scale. Wavelets are functions that satisfy certain mathematical requirements and are used in representing data or other functions.

Mother wavelet or analyzing wavelet is a wavelet prototype function that is being adapted by the wavelet analysis procedure. Temporal analysis is performed with a contracted, high-frequency version of the prototype wavelet, while the frequency analysis is performed with a dilated, low-frequency version of the same wavelet. Because the original signal or function can be represented in terms of a wavelet expansion (using coefficients in a linear combination of the wavelet functions), data operations can be performed using just the corresponding wavelet coefficients. And if we further choose the best wavelets adapted in our data, or truncate the coefficients below the threshold, our data will be sparsely represented. This sparse coding makes wavelets excellent tool in the field of data compression.

Wavelet functions can have infinitely many basis functions and are localized in space, while the former Fourier sine and cosine functions do not.

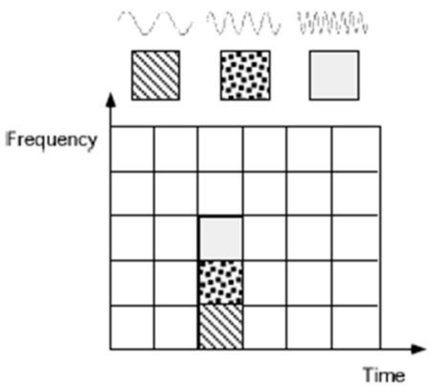


Fig.1. Fourier basis functions, time frequency tiles and coverage of the time-frequency plane

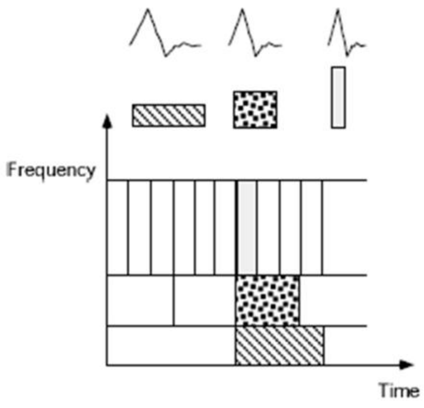


Fig.2. Daubechies wavelet basis functions, time-frequency tiles, and coverage of the time-frequency plane

Figures one and two show a visual comparison of Fourier transform and wavelet transform (daubechies) in terms of time frequency time and plane.

Wavelet transform comprise an infinite set. The different wavelet families make different trade-offs between how compactly the bases functions are localized in space and how smooth they are. Wavelets are classified within a family most often by the number of vanishing moments – extra set of mathematical relationships that must be satisfied, and is directly related to the number of coefficients. The figure above consists of some examples of wavelet families.

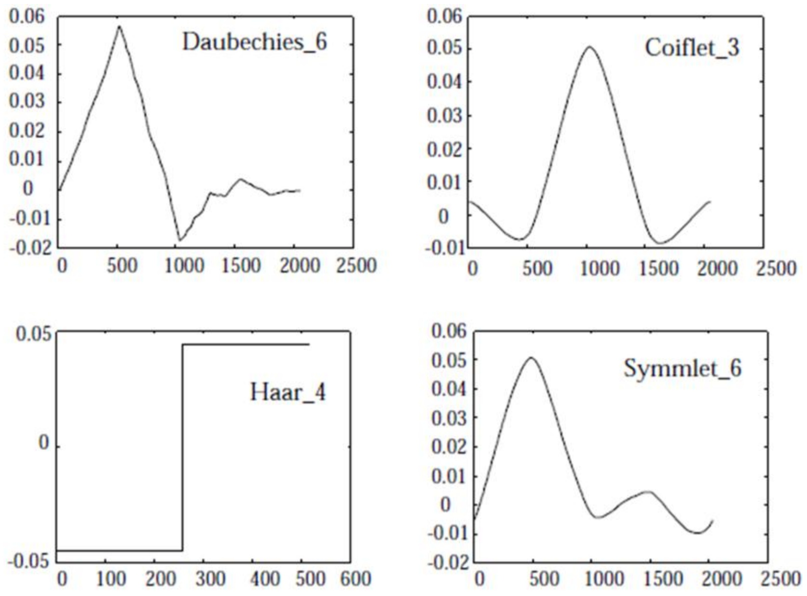


Fig.3. Several different families of wavelets. The number next to the wavelet name represents the number of vanishing moments for the subclass of wavelet

Discrete Wavelet Transform (DWT). The mother function or analyzing wavelet $\Phi(x)$ defines an orthogonal basis:

$$\Phi_{(s,l)}(x) = 2^{\frac{-s}{2}}\Phi(2^{-s}x - l) \qquad (1)$$

The variables s and l are integers that scale and dilate the mother function Φ to generate wavelets. The scale index s indicates the width of the wavelet, while location index l gives its position.

C. Multi-scale Wavelet Edge Detection

An edge in an image is the contour across which the brightness of the image changes abruptly. Edge detection is an important tool in pattern recognition and image segmentation. Sound edge detection can provide valuable information for further image processing such as image segmentation, enhancement, and compression [8].

Two important features of the edge pixels are the following [8]:

1. Edge strength: magnitude of the gradient
2. Edge direction: angle of the gradient

Wavelets overlying the edges yield large wavelet coefficients; wavelets overlying smooth region yield small coefficients.

The decomposition filters at each resolution 2^j are computed by upsampling the filter $h[n]$ by $j \in \mathbb{R}$. The filters $g_j[n]$ are normalized finite difference filters meant to approximate the gradient of the image. The reconstruction image filters are simply the decompositions filters flipped.

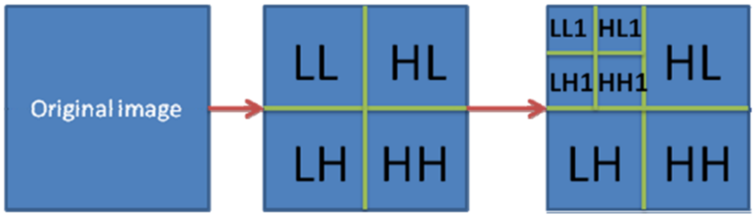


Fig.4.Decomposition of Image

LL is an approximation of the image. It is also the smoothing of the original image which contains most information of the original image. LH preserves the horizontal edge details. HL preserves the vertical edge details. HH preserves the diagonal details which are influenced by noise greatly.

HL means that we used a high pass filter along the rows, and a low pass filter along the column. This process can repeat continuously by putting the first octave's LL sub-image through another set of low pass and high pass filters. These iterative procedures construct the multi-resolution analysis.

Discrete wavelet transform is performed via a cascade of separable convolutions using the above filters. At each resolution 2^j

Image coefficients $\quad a_{j+1}[m] = a_j * \bar{h}_j\bar{h}_j[m]$

Detail Coefficients $\quad \begin{cases} d_{j+1}^1[m] = a_j * \bar{g}_j\delta[m] \\ d_{j+1}^2[m] = a_j * \delta\bar{g}_j[m] \end{cases}$

where $(m_1, m_2) \epsilon \mathbb{Z}^2$.

Wavelet transform is a reversible transform, provided that Equation 1 is satisfied. The reconstruction is possible by using the following reconstruction formula:

$$\sum_{j,k} \gamma(j,k)\psi_{j,k}(t) \qquad (2)$$

Edges of the images are obtained using inverse wavelet transform, while removing the approximation part (LL) of the image.

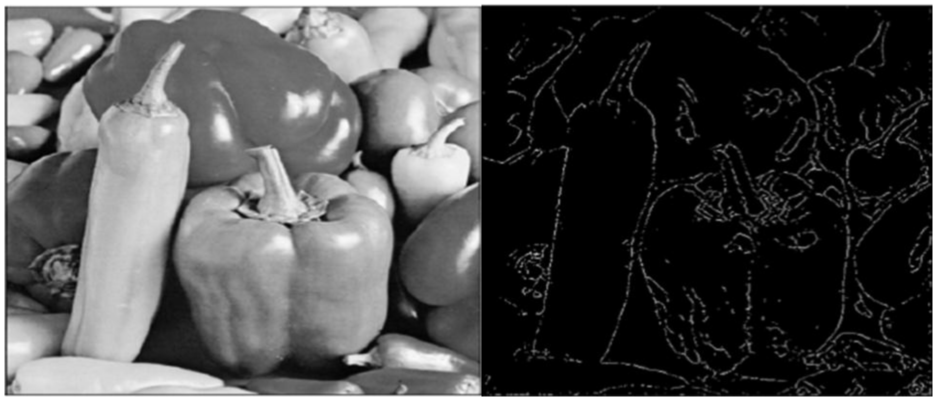


Fig.5.Examples of edge detection of the original image (left) using multi-scale wavelet (right)

D. Support Vector Machine

Noble gave us a brief introduction about Support Vector Machine (SVM) in [47]. It is a computer algorithm that learns by example to assign labels to objects (supervised learning). It is derived from statistical theory by Vladimir Vapnik and his co-workers. Like other classifier algorithms, the concept of treating the objects to be classified as points in a high-dimensional space and finding a line that separates them is also present in SVM. However, it is different from other hyper-plane based classifiers by virtue of how the hyper-plane is selected.

If we are going to define the distance from the separating hyper-plane to the nearest expression vector as the margin of the hyper-plane, then the SVM will select the maximum margin separating hyper-plane. This maximizes the SVM’s ability to predict the correct classification of previously unseen samples.

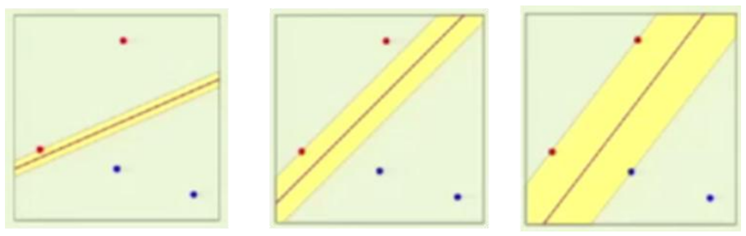


Fig.6.Different lines separating data

Figure 6 shows a linearly separable data. The goal of support vector machine is not only to classify these data correctly but to choose an equation of the line that will produce the biggest margin. A wider margin means less error when classifying data with noise.

In general, SVM uses hyperplane to classify data points. Let $\boldsymbol{x}_n$ be the nearest data point to the plane $\boldsymbol{w}^T\boldsymbol{x} = 0$**.** There are two preliminary technicalities that are being considered in SVM formulation:

1. Normalize $\boldsymbol{w}$: $|\boldsymbol{w}_T\boldsymbol{x}_n| = 1$
2. Pull out $\boldsymbol{w}_0$:

$\boldsymbol{w}$=$(w_1, \dots, w_d)$ apart from b, where b is the bias

The equation of the plane is now $\boldsymbol{w}^T\mathbf{x}+b = 0$

The vector $\boldsymbol{w}$ is perpendicular to the plane in $\boldsymbol{X}$ space. If $\boldsymbol{w}$ is normalized, the distance between $\boldsymbol{x}_n$ and the plane $\boldsymbol{w}^T\boldsymbol{x} + b = 0$ where $|\boldsymbol{w}^T\boldsymbol{x} + b| = 1$ is $\frac{1}{||\boldsymbol{w}||}$

SVM aims to maximize the margin by solving the particular optimization problem:

Maximize $\frac{1}{||\boldsymbol{w}||}$ (3)

subject to $\min |\boldsymbol{w}^T \boldsymbol{x}_n + b| = 1$

$$|\boldsymbol{w}^T \boldsymbol{x}_n + b| = y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b)$$

which is equivalent to

Minimize $\frac{1}{2}\boldsymbol{w}^T\boldsymbol{w}$ (4)

subject to $y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1$

for $n = 1,2, \ldots, N$

By Lagrange formulation, equation becomes

$\sum_{n=1}^{N} \alpha_n (y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) - 1)$ (5)

w.r.t. $\boldsymbol{w}$ and $\boldsymbol{b}$ and maximize w.r.t. each $\alpha_n \geq 0$.

α is also known as the Lagrange multiplier. By differentiating, we can obtain the following equations:

$\nabla_{\boldsymbol{w}} \boldsymbol{\mathcal{L}} = \boldsymbol{w} - \sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} \alpha_{\boldsymbol{n}} y_n x_n = 0$ (6)

$\frac{d\mathcal{L}}{db} = -\sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} \alpha_{\boldsymbol{n}} y_n = 0$ (7)

By substituting the two equations to the original Lagrange, we can obtain

$\sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} \alpha_{\boldsymbol{n}} - \frac{1}{2}\sum_{n=1}^{N} \sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m \, x_n^T x_m$ (8)

Maximize w.r.t. α subject to $\alpha \geq 0$ and $\sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} \alpha_{\boldsymbol{n}} y_n = 0$

We want to have an equation that is free of $\boldsymbol{w}$ and $\boldsymbol{b}$. By quadratic programming,

$$min_{\propto}\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} y_n y_m \alpha_n \alpha_m\, x_n^T x_m - \sum_{n=1}^{N}\alpha_n \qquad (9)$$

$$min_{\alpha}\frac{1}{2}\begin{bmatrix} \alpha^T & y_1 y_1 x_N^T x_1 & \dots & \dots \\ & \dots & \dots & \dots \\ & y_N y_1 x_N^T x_1 & \dots & y_N y_N x_N^T x_N \end{bmatrix}\alpha + (-1^T)\alpha$$

Subject to $y^T\alpha = 0; \quad 0 \le \alpha \le \infty$

which is equal to

$$min_{\alpha}\frac{1}{2}\alpha^T Q\alpha - 1^T\alpha \qquad (10)$$

subject to $y^T\alpha = 0;\ \ \alpha \ge 0$

Quadratic programming gives us the values of α under the Karush-Kuhn-Tucker (KKT) condition

$$\alpha_n\big(y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) - 1\big) = 0, \text{for } n = 1, \dots, N$$

This allows us to solve for w and b which are needed in the decision function for prediction. If $\alpha > 0$, $\boldsymbol{x}_n$ is called a support vector. These are also the points in the data set which are usually the closest to the boundaries of the hyperplane's margin.

$\boldsymbol{w}$ is computed as

$$\boldsymbol{w} = \sum_{\boldsymbol{n=1}}^{\boldsymbol{N}}\alpha_{\boldsymbol{n}} y_n x_n \qquad (11)$$

Using any support vector, b can be calculated in the equation

$$y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) = 1 \qquad (12)$$

Having the values of $\boldsymbol{w}, \alpha$ and b, we can now compute the decision function:

$$sgn(\sum_{i=1}^{l} y_i \alpha_i K\left(\boldsymbol{x}_{i,}, \boldsymbol{x}_j\right) + b) \tag{13}$$

In general, which includes data that are not linearly separable, the following formulation will be used [48]. Introducing ξ in the equation allows violators. By using kernel functions, SVM can process non-linear data.

Given training vectors $\boldsymbol{x}_i \in \mathcal{R}^n, i = 0, \dots, l$, in two classes, and an indicator vector $\boldsymbol{y} \in \mathcal{R}^n$ such that $y_i \in \{1, -1\}$, SVM solves the following primal optimization problem.

$$min_{w,b,\xi} \ \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{l}\xi_i \tag{14}$$

$$\text{subject to } y_i(\boldsymbol{w}^T\phi(\boldsymbol{x}_i) + b) \geq 1 - \xi_i,$$

$$\xi_i \geq 0, \ i = 1, \dots, l$$

where $\phi(\boldsymbol{x}_i)$ maps $\boldsymbol{x}_i$ into a higher-dimensional space and $C > 0$ is the regularization parameter. Due to the possible high dimensionality of the vector variable $\boldsymbol{w}$, we usually solve the following dual problem.

$$min_{\propto} \ \frac{1}{2} \propto^T Q \propto - \boldsymbol{e^T} \propto \tag{15}$$

$$\text{subject to } \boldsymbol{y}^T \propto = 0,$$

$$0 \leq \propto_i \leq C, \ i = 1, \dots, l$$

Where $e = [1 \dots 1]^T$ is the vector of all ones, Q is an l by l positive semi-definite matrix, $Q_{ij} = y_i y_j K(\boldsymbol{x}_{i,}, \boldsymbol{x}_j)$, and $K(\boldsymbol{x}_{i,}, \boldsymbol{x}_j) \equiv \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$ is the kernel function.

SVM uses different types of kernel. Three of these are linear, polynomial, and radial basis function (RBF) which are given below:

$$K\left(\boldsymbol{x}_{i,},\ \boldsymbol{x}_j\right) = \boldsymbol{x}_i^T x_j \tag{16}$$

$$K\left(\boldsymbol{x}_{i,},\ \boldsymbol{x}_j\right) = (\gamma \boldsymbol{x}_i^T x_j + r)^d \tag{17}$$

$$K\left(\boldsymbol{x}_{i,},\ \boldsymbol{x}_j\right) = exp(-\boldsymbol{\gamma}\ ||x_i - x_j||)^2 \tag{18}$$

The kernel function that will be used in the system is Radial Basis Function (RBF).

After (3) is solved, using the primal-dual relationship, the optimal $\boldsymbol{w}$ satisfies

$$\boldsymbol{w} = \textstyle\sum_{i=1}^{l} y_i \alpha_i \phi(\boldsymbol{x}_i) \tag{19}$$

Each new point x_i is classified by evaluating the decision function

$$sgn(\boldsymbol{w}^T \phi(\boldsymbol{x}) + b) = sgn(\textstyle\sum_{i=1}^{l} y_i \alpha_i K(\boldsymbol{x}_{i,}, \boldsymbol{x}_j) + b) \tag{20}$$

The accuracy is computed as

$$\frac{number\ of\ correctly\ predicted\ data}{number\ of\ total\ testing\ data} \times 100 \qquad (21)$$

Provided below are some cases that can be solved by SVM which were summarized by Fletcher in [49]. The pattern of training and prediction is similar to the above except for some additional parameters. SVM can be used in the following cases:

i. *Linearly separable binary classification.*

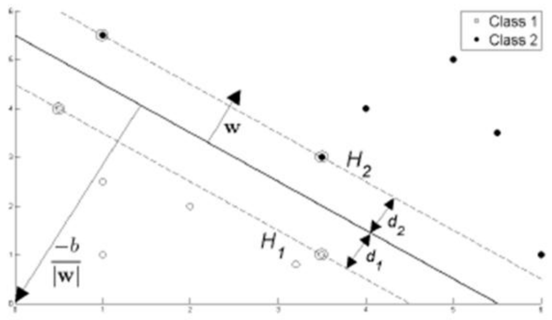


Fig.7.Hyperplane through two linearly separable

ii. *Binary classification for data that is not fully linearly separable.*

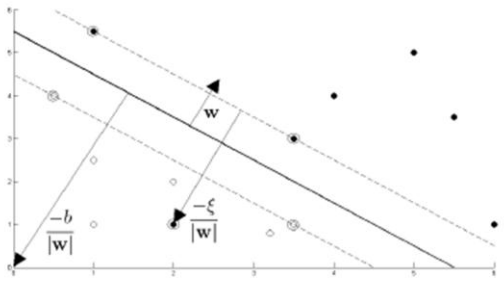


Fig.8.Hyper-plane through two non-linearly separable classes

iii. *SVM for regression.*

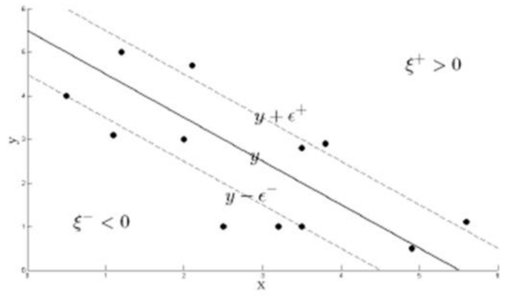


Fig.9.Regression with Ɛ-sensitive tubes

iv. *Nonlinear SVM.*

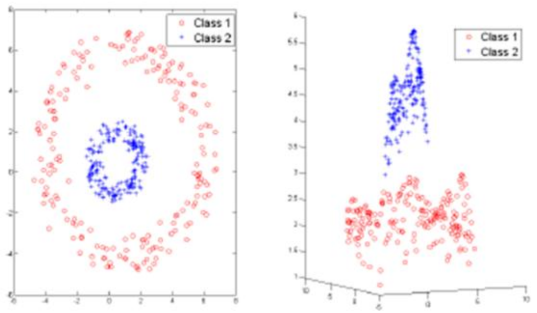


Fig.10.Dichotomous data re-mapped using radial basis kernel

E. Morphological Operations

Morphological operations include dilation, erosion, filling, opening, closing, and others. These operations are carried out to enhance the edges by removing the distortions produced from multi-scale wavelets. Three morphological operations will be used in the system: dilation, region filling and erosion.

Rhody and Carlson in [50] gave us an overview about basic morphological image processing. The structuring element can have any shape.

Dilation, which is written as $A \oplus B$ is illustrated below. The steps will be explained in chapter four:

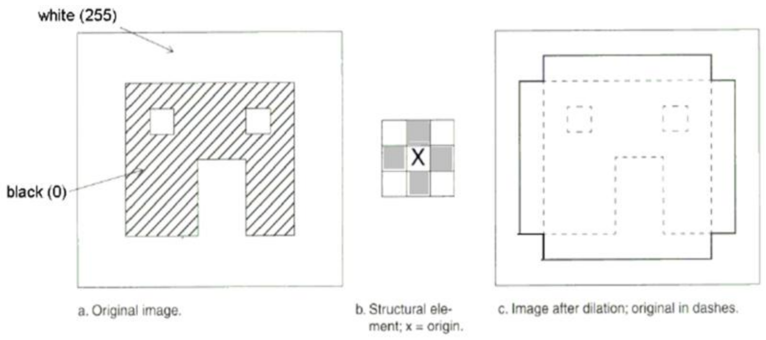

white (255)
black (0)
X
a. Original image.
b. Structural element; x = origin.
c. Image after dilation; original in dashes.

Fig.11.Illustration of dilation process

Erosion, which is written as $A \ominus B$, is similar to dilation, except that it turns pixels to 'white' not 'black'.

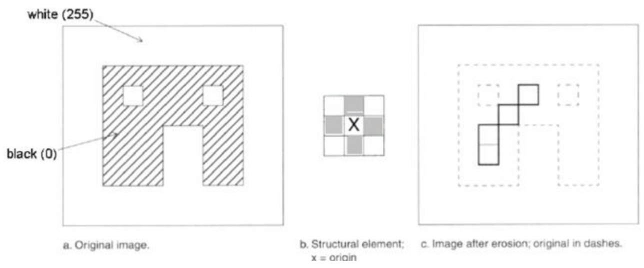

white (255)
black (0)
X
a. Original image.
b. Structural element; x = origin
c. Image after erosion; original in dashes.

Fig.12.Illustration of erosion process

An example of the effect of erosion followed by dilation is the following:

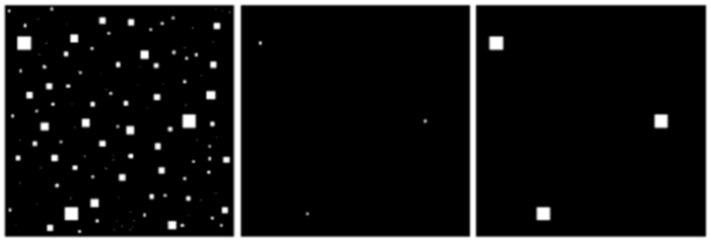


Fig.13. (left) Image of squares of size 1,3,5,7,9, and 15 pixels
(middle) Erosion of left image with a square structuring element of 1's 13 pixels
(right) Dilation of middle image with same structuring image

Region filling is filling the bounded regions with the background color. An illustration follows:

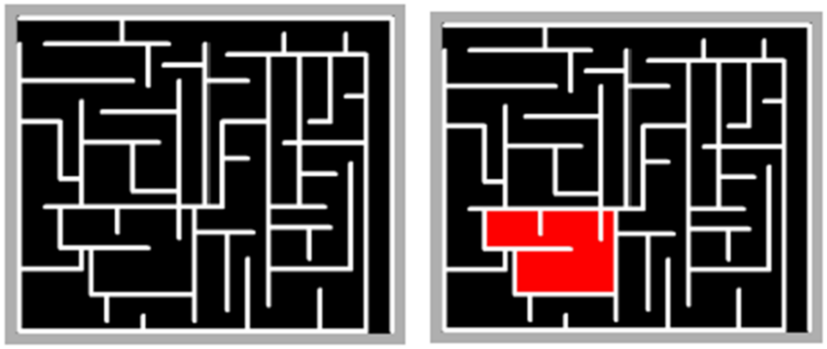


Fig.14.Illustration of region filling algorithm

F. Segmentation

Segmentation is the process of dividing into segments [51]. It is the most crucial and challenging step in the analysis of lung nodules. Nodules are frequently attached to other structures, including the local pulmonary vasculature and the pleural surface adjoining thoracic wall. Segmentation is the final step in detection of suspicious region which is concerned with dividing the image into meaningful region. Segmented image is obtained by using the edge detected output with the morphological filter output [8].

G. Lung Image Database Consortium (LIDC)

The purpose of Lung Image Database Consortium (LIDC) and Image Database Resource Initiative (IDRI) is to provide a repository of computed tomography (CT) scans which can be publicly available reference for the medical imaging research community. The database contains 1018 thoracic CT scans and associated XML-based annotations to stimulate the development of Computer Aided Diagnosis methods for lung nodule detection, classification, and quantitative assessment. For complete reference of LIDC, see [52].

Seven academic centers and eight medical imaging companies collaborated to identify, address, and resolve challenging organizational, technical and clinical issues to provide a solid foundation for a robust database. LIDC/IRDI Database contains 1018 cases, each of which includes images from a clinical thoracic CT scan and an associated XML file that records the results of a two-phase image annotation process performed by four experienced thoracic radiologists. Each radiologist independently reviewed each CT scan and mark lesions belonging to one of the three categories:

i. Nodule $\geq$ 3mm

ii. Nodule $<$ 3mm

iii. Non-nodule $\geq$ 3mm

The radiologists then reviewed their own marks independently along with the marks of the other three radiologists without knowing who marked which. The goal of the process is to render final opinion while identifying as completely as possible all lung nodules in each CT scan without requiring forced consensus.

The Database contains 7371 lesions marked nodule by at least one radiologist. 2669 of these lesions were marked ≥ 3mm by at least one radiologist, of which 928 (34.7%) received such marks from all four radiologists. These 2669 lesions include nodule outlines and subjective nodule characteristic ratings.

A range of scanner manufacturers and models was represented in 670 scans from seven different GE Medical Systems Light Speed scanner models, 74 scans from four different Philips Brilliance scanner models, 205 scans from five different Siemens Definition, Emotion, and Sensation scanner models, and 69 scans from Toshiba Aquilion scanners (the mention of commercial equipment in LIDC-IDRI paper is intended to specify the conditions of the present study and is not an endorsement by the LIDC/IDRI Research Group). The tube peak potential energies used for scan acquisition were as follows: 120 kV (n=818), 130 kV (n=3), 135 kV (n=69), and 140 kV (n=100). Tube current ranged from 40 to 627 mA (mean: 222.1 mA). Slice thicknesses were 0.6 mm (n=7), 0.75 mm (n=30), 0.9 mm (n=2), 1.0 mm (n=58), 1.25 mm (n=349), 1.5 mm (n=5), 2.0 mm (n=124), 2.5 mm (n=322), 3.0 mm (n=117), 4.0 mm (n=1), and 5.0 mm (n=3). Reconstruction interval ranged from 0.45 to 5.0 mm (mean: 1.74 mm). The in-plane pixel size ranged from 0.461 to 0.977 mm (mean: 0.688 mm). While the convolution kernels used for image reconstruction differ among manufacturers, these convolution kernels may be classified broadly as “soft” (n=67), “standard/non-enhancing” (n=560), “slightly enhancing” (n=264), and “over-enhancing” (n=127) (in order of increasing spatial frequencies accentuated by each class).

H. Computed Tomography (CT)/Computerized Axial Tomography (CAT) Scan

CT scan is an x-ray procedure that combines many x-ray images with the aid of a computer to generate cross-sectional views and, if needed, three-dimensional images of the internal organs and structures of the body [53].

Using thoracic CT scan, we can analyze the scanned body part to be able to see nodules. A nodule is an abnormality that leads to lung cancer, characterized by a small round or oval shaped growth on the lung that appears as a white shadow on an X-ray or CT-scan [8].

There are two well-known types of CT scan: the sequential CT and the spiral CT. In sequential CT, a cross-sectional image is produced by scanning a transverse slice of the body from different angular positions while the tube and detector rotate 360^0 around the patient while the table is remaining to be stationary. On the other hand, in spiral CT, the patient on the table is moved continuously through the scan field in the z direction while the gantry performs multiple 360^0 rotations in the same direction. Spiral CT is usually preferred because it has shorter scan times. It also results to complete coverage of organs in single respiratory position, and additional diagnostic information due to improved resolution (thinner slices) and 3D visualizations in routine. The thinner the slices, the more detailed it will be. This technique allows doctors to see more comprehensive images.

A CT scan system comprises of several components which include the following: the scanning unit (i.e. gantry) with tube and detector system, the patient table, the image processor for reconstruction of image, and the console, which serves as the interface which is designed for controlling the machine. The x-ray in scanning unit functions as a transmitter. The tube converts the incident X-rays of varying intensity to electric signals. These signals are amplified by downstream electronic components and converted to digital pulses. These detectors dramatically improve image quality.

With CT scanning, numerous x-ray beams and a set of electronic x-ray detectors rotate around the patient. As an illustration, CT imaging is like looking into a loaf of bread and cutting it to thin slices. When the image slices are reassembled by computer software, the result is a very detailed multi-dimensional view of the body’s interior.

In present, there are refinements in detector technology to allow CT scanners to obtain multiple slices. These are called multi-slice CT or multi-detector CT. It results to faster scanning and produces more detailed view of the body part being scanned.

Some algorithms were developed and being used in CT scanners to make image processing faster and cheaper in memory space. Back projection and reprojection are examples of these. One of the leading scanner vendors, General Electric (GE) developed an algorithm that will slash the compute time for low radiation dose scans from 100 hours per image to less than an hour. However, these algorithms are not yet capable of automated segmentation and classification of nodules.

IV. Design and Implementation

A. System Design

The Lung Nodule Detector and Classifier Tool (LNDCT) is able to detect and classify nodules based on the SVM model loaded. The processes include edge detection, morphological operations, segmentation and SVM classification. The system is able to generate reports based on the processed CT scan images. It has a tutorial for the users. LNDCT is able to output these if the user will input the needed parameters: lung images, SVM algorithm and training data (optional). The context diagram is shown on Figure 15.

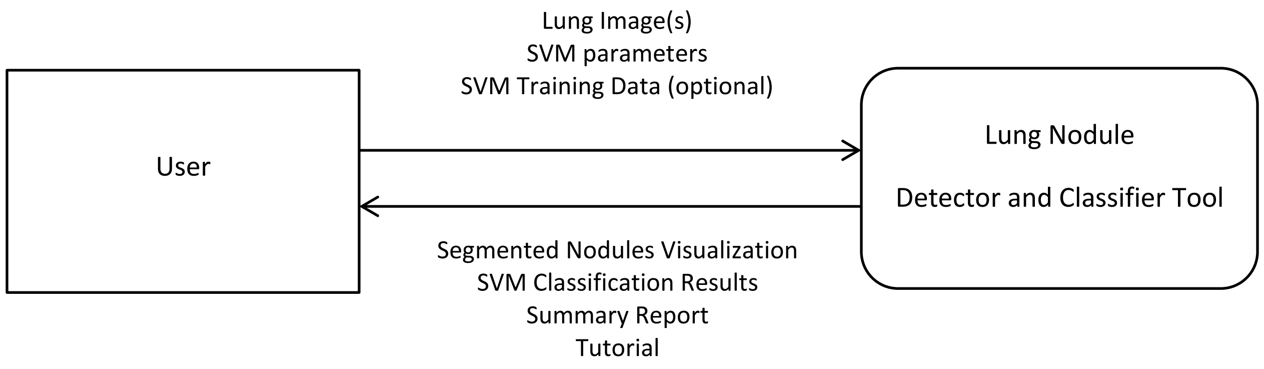

Lung Image(s)
SVM parameters
SVM Training Data (optional)
User
Lung Nodule
Detector and Classifier Tool
Segmented Nodules Visualization
SVM Classification Results
Summary Report
Tutorial

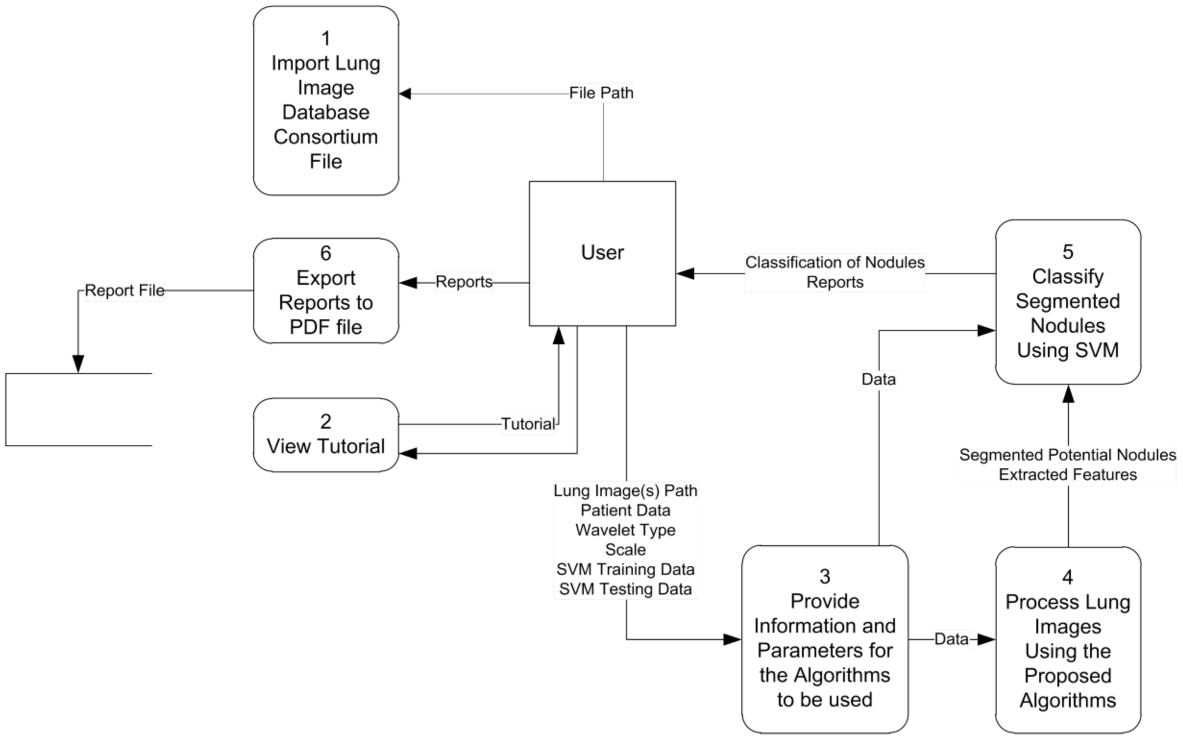
Fig.15.Context Diagram, LNDCT

The tool is implemented in an object oriented manner. The graphical user interface is divided into three major components.

User Input Panel. This is where the user inputs all the needed data for the system. Under the “files” tab, the user is able to explore and choose the loaded images.

Result Panel. All results of the processes are shown here. These include the segmented image, and the computed results.

Information Panel. All information is shown here. The image SOP UID, series instance UID, and the information about the processes is shown here.



1 Import Lung Image Database Consortium File
File Path
User
6 Export Reports to PDF file
Reports
Report File
Classification of Nodules Reports
5 Classify Segmented Nodules Using SVM
Data
2 View Tutorial
Tutorial
Segmented Potential Nodules Extracted Features
Lung Image(s) Path
Patient Data
Wavelet Type
Scale
SVM Training Data
SVM Testing Data
3 Provide Information and Parameters for the Algorithms to be used
Data
4 Process Lung Images Using the Proposed Algorithms

Fig.16.Top Level Data Flow Diagram, Lung Nodule Detector

The Lung Nodule Detector and Classifier Tool has six main processes. These are the following: import a Lung Image Database Consortium (LIDC) file (lung images in DICOM format); view tutorial; provide information and parameters for the algorithms to be used; process lung image using the proposed algorithms;   classify segmented nodules using SVM; and export reports to PDF file. Viewing the tutorial is a process which can be skipped by the user. Providing supplementary information involves getting the needed parameters for the wavelet transform and SVM training and prediction process. Processing the lung image file involves these main processes: edge detection using multi-scale wavelet transform; and segmentation by applying morphological operations to the edge detected image. The more detailed version of the process is shown in figure 17. Classification of segmented lung nodules involves training and prediction of SVM. Its sub-explosion is shown in figure 20. Exporting reports means being able to export the results from the previous processes to PDF file.

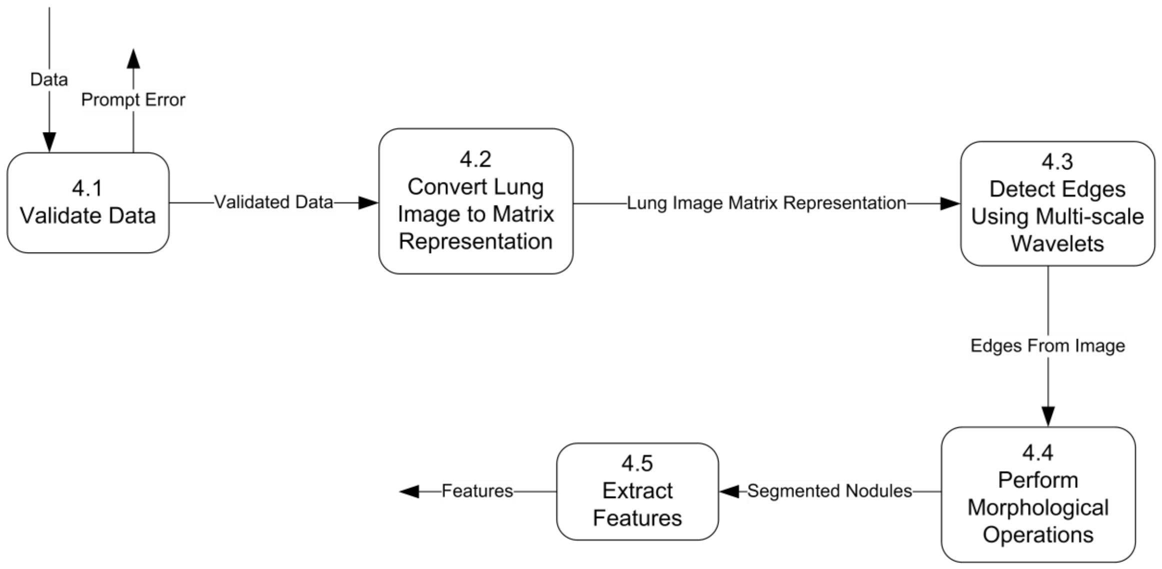

Data
Prompt Error
4.1 Validate Data
Validated Data
4.2 Convert Lung Image to Matrix Representation
Lung Image Matrix Representation
4.3 Detect Edges Using Multi-scale Wavelets
Edges From Image
4.4 Perform Morphological Operations
Segmented Nodules
4.5 Extract Features
Features

Fig.17.Sub-explosion of process 4, LNDCT

Figure 17 shows the detailed process of image processing. The data inputted is validated. It prompts an error message if the data is invalid. The lung image from the validated data is converted to its matrix representation. The image undergoes edge detection process using multi-scale wavelets. The segmented lung nodule image is a result of applying morphological operations on the edge detected image. Features which include mean and standard deviation are extracted from the image. These undergo process five which is shown in figure 20.

The sub-explosion of the multi-scale edge detection is shown in figure 18.

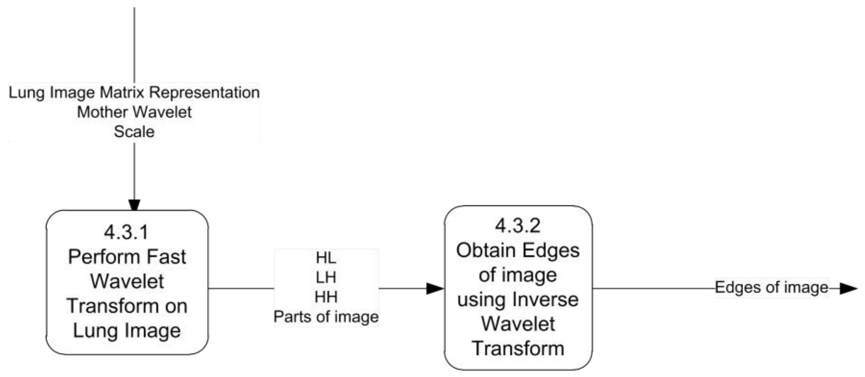

Lung Image Matrix Representation
Mother Wavelet
Scale
4.3.1
Perform Fast Wavelet Transform on Lung Image
HL
LH
HH
Parts of image
4.3.2
Obtain Edges of image using Inverse Wavelet Transform
Edges of image

Fig.18.Sub-explosion of Multi-scale Edge Detection Process, LNDCT

The matrix representation of the lung image, wavelet type and scale values are the required inputs for the edge detection process. The default scale is 2 because it was the scale used by Aarthy and Ragupathy in [8] that yields to good result. The matrix representation of the lung image will undergo fast wavelet transform. The default wavelet type is quadratic spline because it is the wavelet family commonly used for edge detection. The quadratic spline wavelet filter is $H(\omega) = \frac{1}{8}(e^{-j\omega} + 3 + 3e^{j\omega} + e^{2j\omega})$, which produces the coefficients $h_{-1} = \frac{\sqrt{2}}{8}$, $h_0 = \frac{3\sqrt{2}}{8}$, $h_1 = \frac{3\sqrt{2}}{8}$, and $h_2 = \frac{\sqrt{2}}{8}$. Similarly, $G(\omega) = 2e^{j\omega} - 2$, which produces the coefficients $g_0 = -2\sqrt{2}$ and $g_1 = 2\sqrt{2}$.

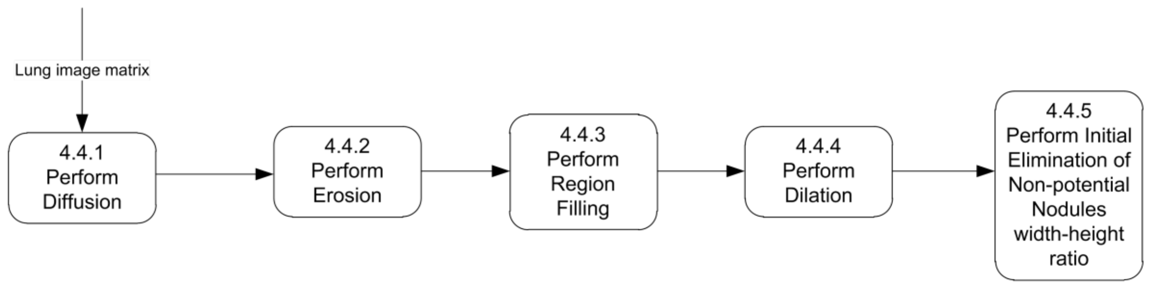

Lung image matrix
4.4.1 Perform Diffusion
4.4.2 Perform Erosion
4.4.3 Perform Region Filling
4.4.4 Perform Dilation
4.4.5 Perform Initial Elimination of Non-potential Nodules width-height ratio

Fig19.Sub-explosion of the Morphological Operation Process, LNDCT

The sub-explosion of the morphological operation process is shown in figure 19. Dilation, region filling, and erosion are used to smoothen and immediately eliminate unnecessary parts of the image. The algorithms are also given below. Sphere is the structuring element used in erosion since it produced the most desired result in [8].

1. *If the origin of the structuring element coincides with a 'white' pixel in the image, there is no change; move to the next pixel.*
2. *If the origin of the structuring element coincides with a 'black' image, make black all pixels from the image covered by the structuring element.*

Algorithm 1. Dilation

1. *If the origin of the structuring element coincides with a 'white' pixel in the image, there is no change; move to the next pixel.*
2. *If the origin of the structuring element coincides with a 'black' pixel in the image, and at least one of the 'black' pixels in the structuring element falls over a white pixel in the image, then change the 'black' pixel in the image (corresponding to the position on which the center of the structuring element falls) from 'black' to a 'white'.*

Algorithm 2. Erosion

```
1. Let $B$ be a structuring element of type $N_4$ or $N_8$ depending on the desired connectivity.
2. Select a pixel $p$ inside a background color region.
3. Initialize $X_0$ to be an array of background pixels except $X_0[p] = 1$.
4. Perform the iteration

        $X_k = (X_{k-1} \oplus B) \cap A^c$ for $k = 1,2,3,\ldots$ until $X_k = X_{k-1}$
```

Algorithm 3. Region Filling

Diffusion diffuses across image edges. The smoothness of image edges enables accurate calculation of voxel based geometric features [11].

The resulting image goes through initial elimination process for the nodules. This will eliminate the obvious non-nodule objects based on width-height ratio.

Figure 20 shows the top level data flow of SVM. It is separated to two major parts. Given a training data set S, which includes C and γ, SVM will be trained. The default value of γ is 1 because it the value indicated in the paper which produced good results. The kernel function that will be used is RBF. After training the SVM, it produces a model, together with other parameters that are basis for the prediction. The testing part acts upon the testing data using the model from the training. If the user skipped the training part, SVM will use the default model which is already embedded in the system. These processes determine the classification of the data.

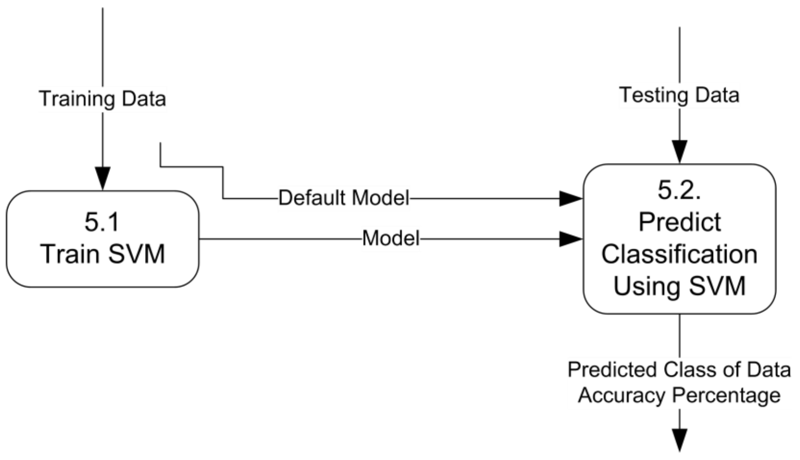


Fig.20.Sub-explosion of SVM process, LNDCT

The training and testing data include the value of mean and standard deviation computed from the segmented lung nodule image. Aarthy and Ragupathy computed for other features which include contrast and entropy [8]. But since mean and standard deviation vary their values between the cancerous and non-cancerous images, these were selected for training and testing of data. The formulae are given in equation (22) and (23). A binary classification is being used here, in which a hyper-plane classifies the given data into two different classes. In our application of SVM, the features, which consist of mean and variance, are denoted as $\boldsymbol{x}$. The classes (which are cancerous and non-cancerous) are denoted as $\boldsymbol{y}$. Since we are only classifying two classes, 0 will denote non-cancerous nodule while 1 will denote cancerous nodule. Based from the model obtained from SVM training, cancerous and non-cancerous nodules are decided by the decision function in (20). If the value of that equation is 0, then it is classified as non-cancerous. If its value is 1, it is classified as cancerous.

$$mean:\ \ \bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{22}$$

$$sd = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{23}$$

where the $mean$ denotes the average value of all the pixels and n is the number of elements in the sample.

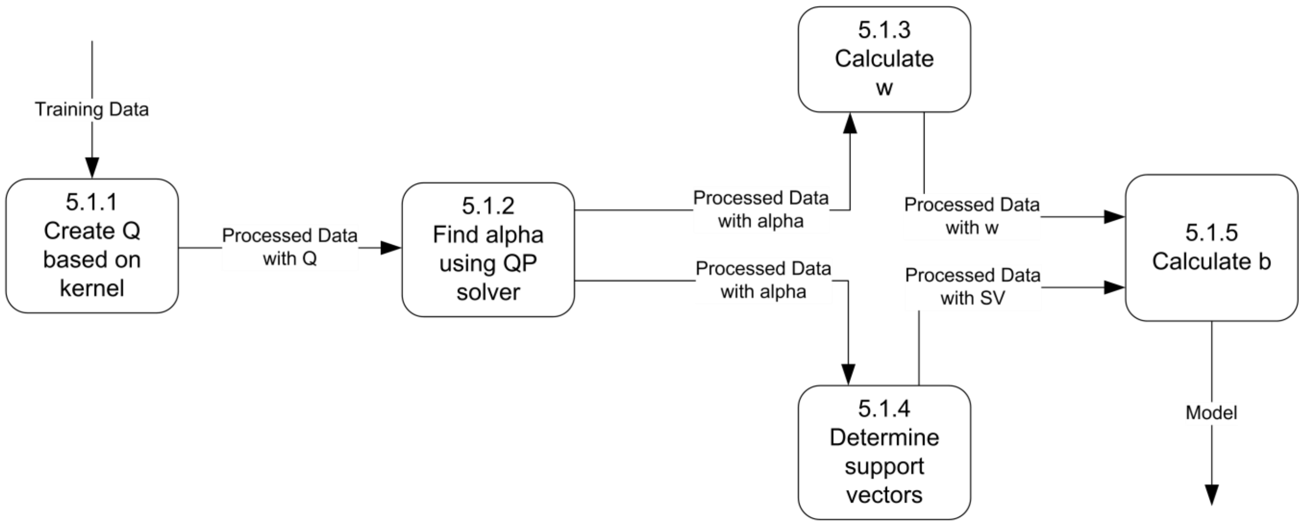

Training Data
5.1.1 Create Q based on kernel
Processed Data with Q
5.1.2 Find alpha using QP solver
Processed Data with alpha
Processed Data with alpha
5.1.3 Calculate w
5.1.4 Determine support vectors
Processed Data with w
Processed Data with SV
5.1.5 Calculate b
Model

Fig.21.Sub-explosion of Training Process, LNDCT

. Figure 21 shows the sub-explosion of the training process. The formulae being used are discussed in chapter three. Based from the training data which includes the RBF kernel and the regularization parameter C, Q matrix will be computed. Then, α will be solved using quadratic programming. Given the training data and α, w will be calculated. The α's that are greater than zero and less than C will be determined as support vectors. Lastly, b is calculated. These values comprise the model which are used in prediction of cancerous and non-cancerous nodules

Decomposition Methods. The main difficulty in solving problem is that Q is a dense matrix and may be too large to be stored. A decomposition method modifies only a subset of α per iteration, so only some columns of Q are needed. This subset of α variables, denoted as B leads to smaller optimization sub-problem. We consider an SMO-type decomposition method mentioned in [48].

Algorithm 7(An SMO-type decomposition method in Fan et al., 2005)

1. Find α^1 as the initial feasible solution. Set $k = 1$.

2. If α^k is a stationary point of problem (3), stop. Otherwise, find a two element working set $B = \{i, j\}$ by WSS 1 which will be described later. Define $N \equiv \{1, \ldots, l\} \backslash B$. Let $\propto_B^k$ and $\propto_N^k$ be sub-vectors of $\propto^k$ corresponding to B and N, respectively.

3. Solve the following sub-problem with the variable $\propto_B = [\propto_i \ \ \propto_j]^T$.

$$min_{\propto_i, \propto_j} \ \frac{1}{2}[\propto_i \propto_j] \begin{bmatrix} Q_{ii} & Q_{ij} \\ Q_{ij} & Q_{jj} \end{bmatrix} \begin{bmatrix} \propto_i \\ \propto_j \end{bmatrix} + (\boldsymbol{p}_B + Q_{BN} \propto_N^k)^T \begin{bmatrix} \propto_i \\ \propto_j \end{bmatrix} \qquad (24)$$

subject to $0 \leq \alpha_i, \alpha_j \leq$ C,

$y_i \propto_i + y_j \propto_j = \Delta - \boldsymbol{y}_N^T \alpha_N^k$

4. Set α_B^{k+1} to be the optimal solution of sub-problem (3) and (4), and $\alpha_N^{k+1} \equiv \alpha_N^k$. Set $k \leftarrow k + 1$ and go to Step 2.

Stopping Criteria. The Karush-Kuhn-Tucker (KKT) optimally condition problem in (3) implies that a feasible α is a stationary point in (3) if and only if there exists a number b and two nonnegative vectors λ and ξ such that

$$\nabla f(\alpha) + b\boldsymbol{y} = \lambda - \xi,$$

$$\nabla f(\alpha) + by_i \begin{cases} \geq 0 \; if \; \alpha_i < C, \\ \leq 0 \; if \; \alpha_i > 0, \end{cases} \qquad (25)$$

where $\nabla f(\alpha) \equiv Q\alpha + p$ is the gradient of $f(\alpha)$.

since $y_i = \pm 1$, the condition (25) is equivalent to that there exists b such that

$$m(\alpha) \leq b \leq M(\alpha),$$

where

$$m(\alpha) \equiv max_{i\epsilon I_{up}(\alpha)} - y_i \nabla_i f(\alpha) \text{ and } M(\alpha) \equiv min_{i\epsilon I_{low}(\alpha)} - y_i \nabla_i f(\alpha),$$

and

$$I_{up}(\alpha) \equiv \{t|\alpha_t < C, y_t = 1 \; or \; \alpha_t > 0, y_t = -1\}, \text{ and}$$

$$I_{low}(\alpha) \equiv \{t|\alpha_t < C, y_t = -1 \; or \; \alpha_t > 0, y_t = 1\}$$

That is, a feasible α is a stationary point of problem (3) if and only if

$$m(\alpha) \leq M(\alpha) \qquad (26)$$

A suitable stopping condition is

$$m(\alpha^k) - M(\alpha^k) \leq \epsilon \qquad (27)$$

where ϵ is tolerance.

WSS 1 (Fan et al.)

1. For all $t, s,$ define

$$a_{ts} \equiv K_{tt} + K_{ss} - 2K_{ts},\ b_{ts} \equiv -y_t \nabla_t f(a^k) + y_s \nabla_s f(a^k) > 0 \tag{28}$$

and

$$\bar{a}_{ts} \equiv \begin{cases} a_{ts} & if\ a_{ts} > 0 \\ \tau & otherwise \end{cases}$$

Select

$$i\ \epsilon\ argmax_t\left\{-y_t \nabla_t f(a^k) \middle| t\ \epsilon\ I_{up}(\alpha^k)\right\},$$

$$j\epsilon\ argmin_t\left\{-\frac{b_{it}^2}{\bar{a}_{it}} \middle| t\ \epsilon\ I_{low}(\alpha^k), -y_t \nabla_t f(a^k) < -y_i \nabla_i f(a^k)\right\}$$

2. Return $B = \{i, j\}$

The procedure selects a pair $\{i, j\}$ approximately minimizing the function value; see the term $-\frac{b_{it}^2}{\bar{a}_{it}}$

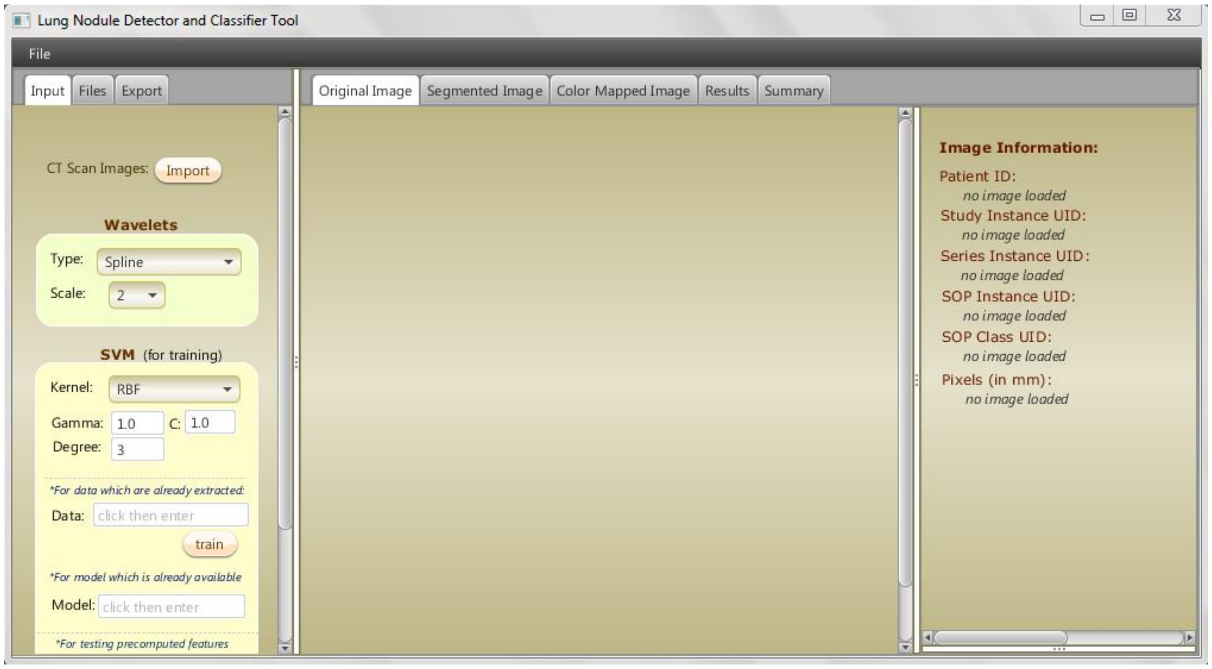
B. Technical Architecture

LNDCT tool is implemented using Java, it runs on most operating systems given that the latest Java Runtime Environment is pre-installed in the system. LNDCT uses the latest library that is called JavaFX, which is already packaged with the latest JRE release.

Since image processing and machine learning algorithms are very expensive in memory, it is recommended for the computer to have at least 2 GB of RAM, and at least 2 GB free space. For more detailed and better visualization, a video card is also recommended.

V. Results

LNDCT main screen is divided into three major parts: (a) the left panel is concerned about user defined operations (inputting parameters, exploring the loaded files, and exporting the results). (b) The biggest portion in the middle is the output window. It is where the processed images will be displayed. (c) The panel on the right will contain the information about the image loaded in the middle portion.

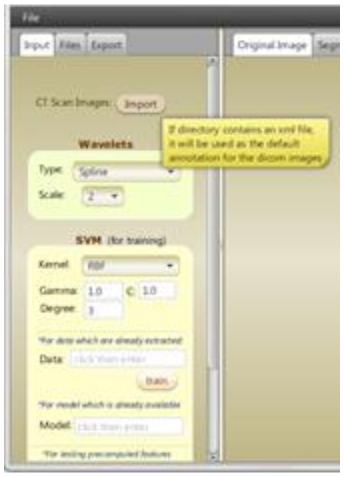


Type: Spline
Scale: 2
Kernel: RBF
Gamma: 1.0 C: 1.0
Degree: 3
*For data which are already extracted:
Data:
*For model which is already available
Model:
*For testing precomputed features

Image Information:
Patient ID: no image loaded
Study Instance UID: no image loaded
Series Instance UID: no image loaded
SOP Instance UID: no image loaded
SOP Class UID: no image loaded
Pixels (in mm): no image loaded

Fig.22 Main Screen Window, LNDCT

To start using LNDCT, the user must choose a folder which contains the dicom files. A notification message will pop up when the mouse is hovered to the import button (see figure 23). These will be processed by group, and will be cross-validated to an xml file (known data) if it exists in the directory loaded. The xml file should follow the correct format used by the Lung Image Database Consortium. Figure 24 shows how to load a folder in LNDCT. An example xml file from LIDC is shown in Figure 25.



If directory contains an xml file, it will be used as the default annotation for the dicom images

CT Scan Images: Import

Type: Spline
Scale: 2

Kernel: RBF
Gamma: 1.0 C: 1.0
Degree: 3

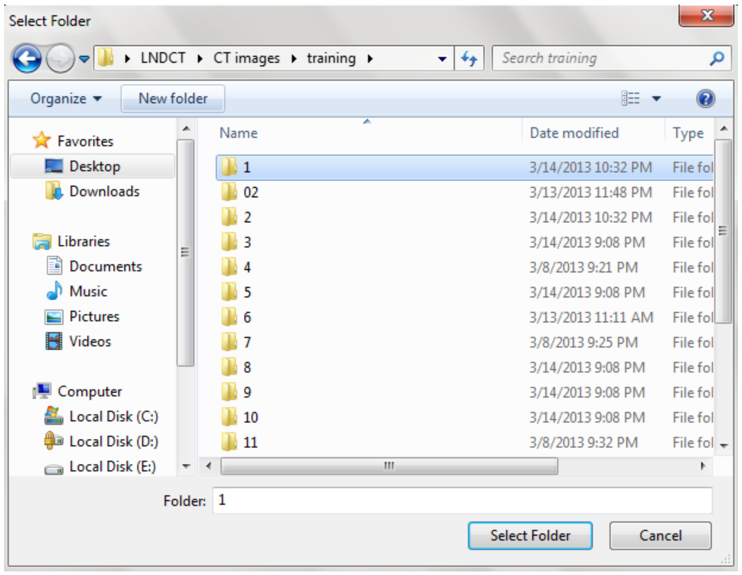
*For data which are already extracted
Data: click then enter

*For model which is already available
Model: click then enter

*For testing precomputed features

Fig.23 A notification message pops up when a mouse is hovered to import button, LNDCT

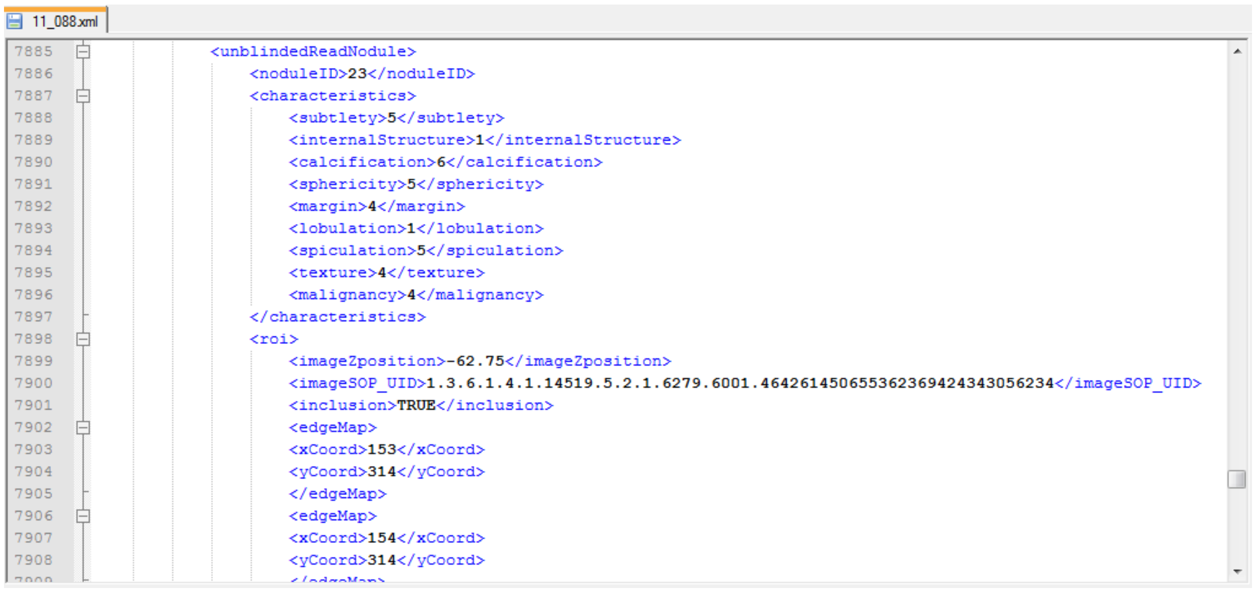
If the folder also contains an SVM model file, it will be used for prediction if the detected nodules are cancerous or not, unless another model file is loaded.



Name	Date modified	Type
1	3/14/2013 10:32 PM	File fol
02	3/13/2013 11:48 PM	File fol
2	3/14/2013 10:32 PM	File fol
3	3/14/2013 9:08 PM	File fol
4	3/8/2013 9:21 PM	File fol
5	3/14/2013 9:08 PM	File fol
6	3/13/2013 11:11 AM	File fol
7	3/8/2013 9:25 PM	File fol
8	3/14/2013 9:08 PM	File fol
9	3/14/2013 9:08 PM	File fol
10	3/14/2013 9:08 PM	File fol
11	3/8/2013 9:32 PM	File fol

Folder: 1

Fig.24 Import images dialog box, LNDCT

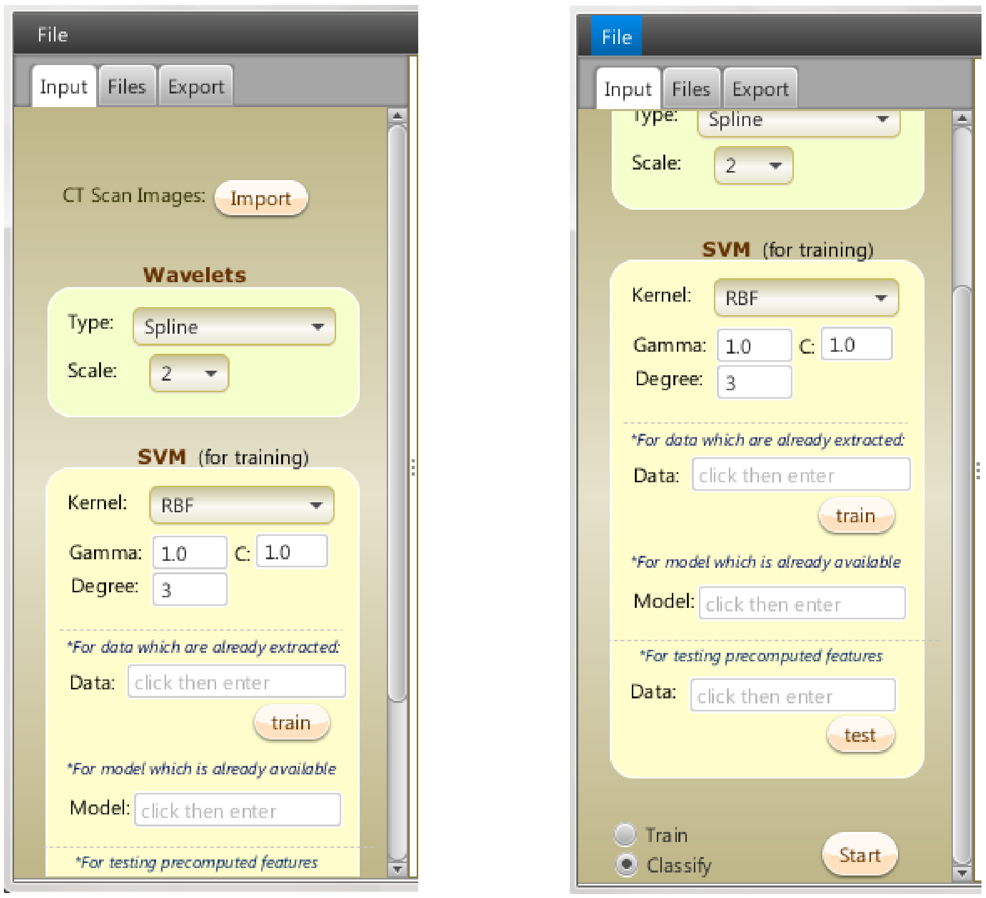


```
<unblindedReadNodule>
    <noduleID>23</noduleID>
    <characteristics>
        <subtlety>5</subtlety>
        <internalStructure>1</internalStructure>
        <calcification>6</calcification>
        <sphericity>5</sphericity>
        <margin>4</margin>
        <lobulation>1</lobulation>
        <spiculation>5</spiculation>
        <texture>4</texture>
        <malignancy>4</malignancy>
    </characteristics>
    <roi>
        <imageZposition>-62.75</imageZposition>
        <imageSOP_UID>1.3.6.1.4.1.14519.5.2.1.6279.6001.464261450655362369424343056234</imageSOP_UID>
        <inclusion>TRUE</inclusion>
        <edgeMap>
        <xCoord>153</xCoord>
        <yCoord>314</yCoord>
        </edgeMap>
        <edgeMap>
        <xCoord>154</xCoord>
        <yCoord>314</yCoord>
```

Fig.25 Example known data in xml format, LIDC

Each CT scan of a patient in LIDC database contains about 100 to 600 images. Each patient has a single xml file that contains all the information needed for reference and cross-validation. It includes the series instance UID, study instance UID, image SOP UID (which is unique across images per scan per patient), the coordinates of the nodules, and their malignancy.

The default values of the parameters are based on the paper previously discussed. These can be changed by the user freely. The parameters which can be changed include the following: the type of wavelet and scale to be used and the kernel for SVM.


File

Input | Files | Export

CT Scan Images: Import

Wavelets

Type: Spline

Scale: 2

SVM (for training)

Kernel: RBF

Gamma: 1.0 C: 1.0

Degree: 3

For data which are already extracted:

Data: click then enter

train

For model which is already available

Model: click then enter

For testing precomputed features

Data: click then enter

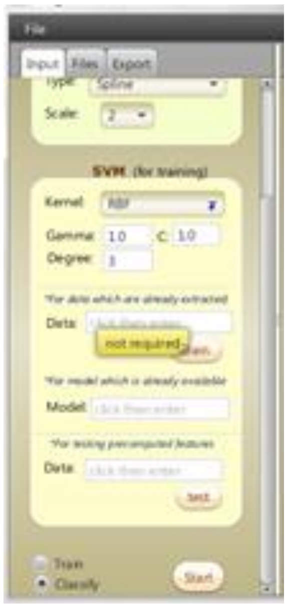
test

Train [✓ Classify]

Start

Fig.26 Input panel, LNDCT

In the SVM section, the user has the power to import a data file (which already contains the extracted features) and train it separately. The user can also import an SVM model to be used for prediction. These are not required as shown in figure 27.

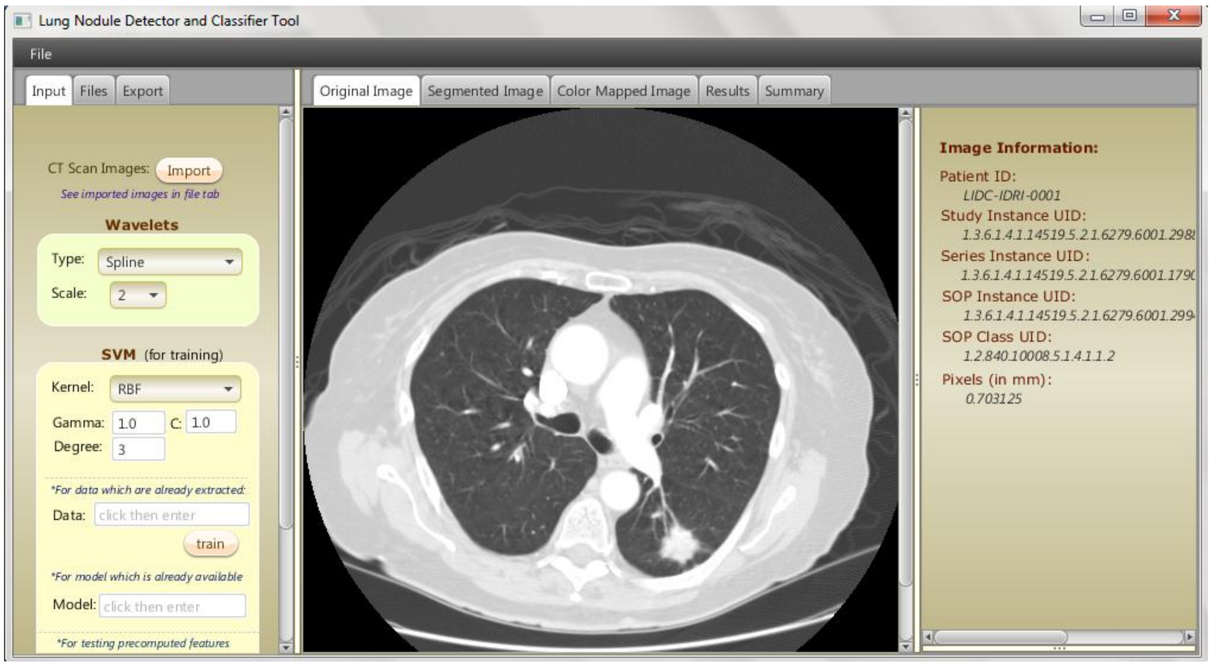

File
Input
Files
Export
SVM (for training)
*For data which are already extracted
not required
*For model which is already available
*For testing precomputed features
Test
Start

Type: Spline
Scale: 2
Kernel: RBF
Gamma: 1.0
C: 1.0
Degree: 3
Data: click then enter
Model: click then enter
Data: click then enter
Train
[✓ Classify]

Fig.27 SVM (part of input panel), LNDCT

Training is not required in LNDCT, as long as an SVM model is already available. However, LNDCT allows users to train data using a file which already contains extracted features, or by loading images and choosing the train radio button (see figure 26).

An example folder loaded in LNDCT is shown in figure 28.



CT Scan Images:
See imported images in file tab

Wavelets

Type: Spline
Scale: 2

SVM (for training)

Kernel: RBF
Gamma: 1.0 C: 1.0
Degree: 3

**For data which are already extracted:*
Data: click then enter

**For model which is already available*
Model: click then enter

**For testing precomputed features*

Image Information:

Patient ID:
LIDC-IDRI-0001
Study Instance UID:
1.3.6.1.4.1.14519.5.2.1.6279.6001.298
Series Instance UID:
1.3.6.1.4.1.14519.5.2.1.6279.6001.179
SOP Instance UID:
1.3.6.1.4.1.14519.5.2.1.6279.6001.299
SOP Class UID:
1.2.840.10008.5.1.4.1.1.2
Pixels (in mm):
0.703125

Fig.28, Example of loaded dicom image, LNDCT

After the images are loaded, the information panel will display the metadata information embedded in the dicom file. These include the patient ID, study instance UID, series instance UID, SOP instance UID, and pixel spacing in millimeters. The information of the image in figure 28 is zoomed in in figure 29.

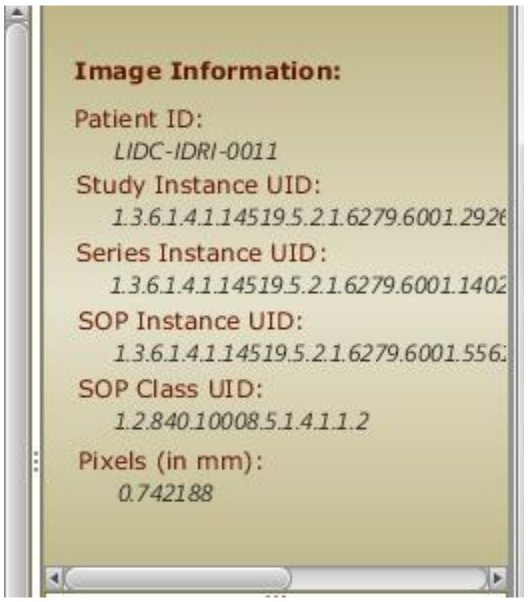
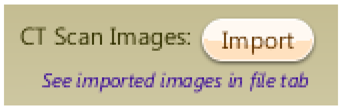


Image Information:

Patient ID:
LIDC-IDRI-0011
Study Instance UID:
1.3.6.1.4.1.14519.5.2.1.6279.6001.292
Series Instance UID:
1.3.6.1.4.1.14519.5.2.1.6279.6001.140
SOP Instance UID:
1.3.6.1.4.1.14519.5.2.1.6279.6001.556
SOP Class UID:
1.2.840.10008.5.1.4.1.1.2
Pixels (in mm):
0.742188

Fig.29 Information on the loaded dicom image, LNDCT

A notification message will also appear in the bottom of the import button after successfully loading images. It is shown in figure 30.



See imported images in file tab

Fig.30 Notification message after importing a folder, LNDCT

Under the files tab, in the input panel, the user can choose which image to display in the output panel. The information loaded will change as well.

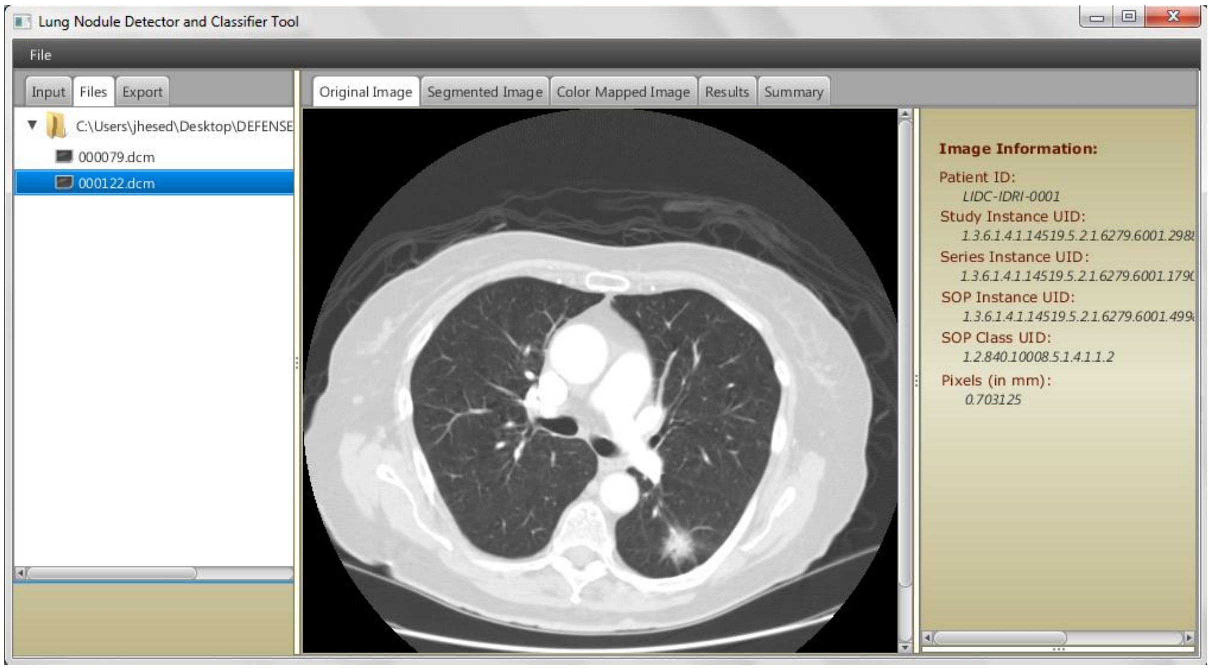
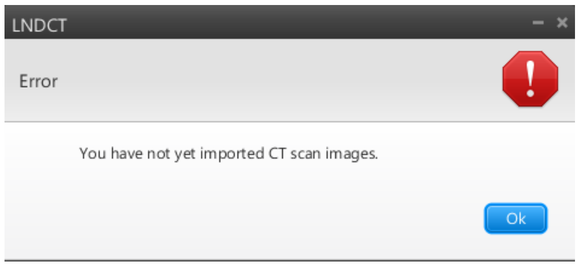


Image Information:
Patient ID:
LIDC-IDRI-0001
Study Instance UID:
1.3.6.1.4.1.14519.5.2.1.6279.6001.298
Series Instance UID:
1.3.6.1.4.1.14519.5.2.1.6279.6001.179
SOP Instance UID:
1.3.6.1.4.1.14519.5.2.1.6279.6001.499
SOP Class UID:
1.2.840.10008.5.1.4.1.1.2
Pixels (in mm):
0.703125

Fig.31 Changing image, LNDCT

In training LNDCT, choose the "train" radio button and click start. The tool will automatically segment the potential nodules and extract their features. LNDCT will then write the features of the nodules in a file named "data_training.data" in the same directory loaded. A model file will be created based from this training data which will be used for future classifications. The file will be named "data_training.data.model" which will also be located in the same folder.



Error

You have not yet imported CT scan images.

Fig.32 Sample of system error message, LNDCT

Internal pre-processing steps like diffusion, binarization, and edge detection are shown on figure 33. It will undergo morphological operations. Erosion was applied to immediately eliminate very small particles in the image. Edges obtained from wavelets will be used to segment the nodules attached in some regions. Region filling was done to remove the lung mask. Dilation was applied to smoothen the extracted potential nodules. Candidate nodules where the value of width-height ratio is greater than 1.5 will be eliminated. The segmented nodules are shown in figure 34.

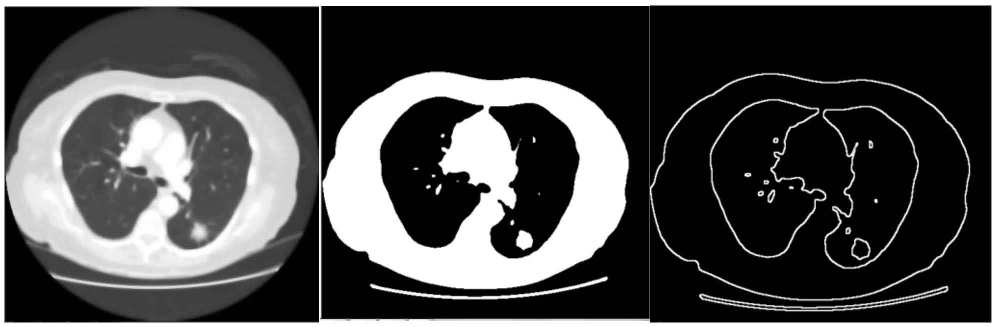
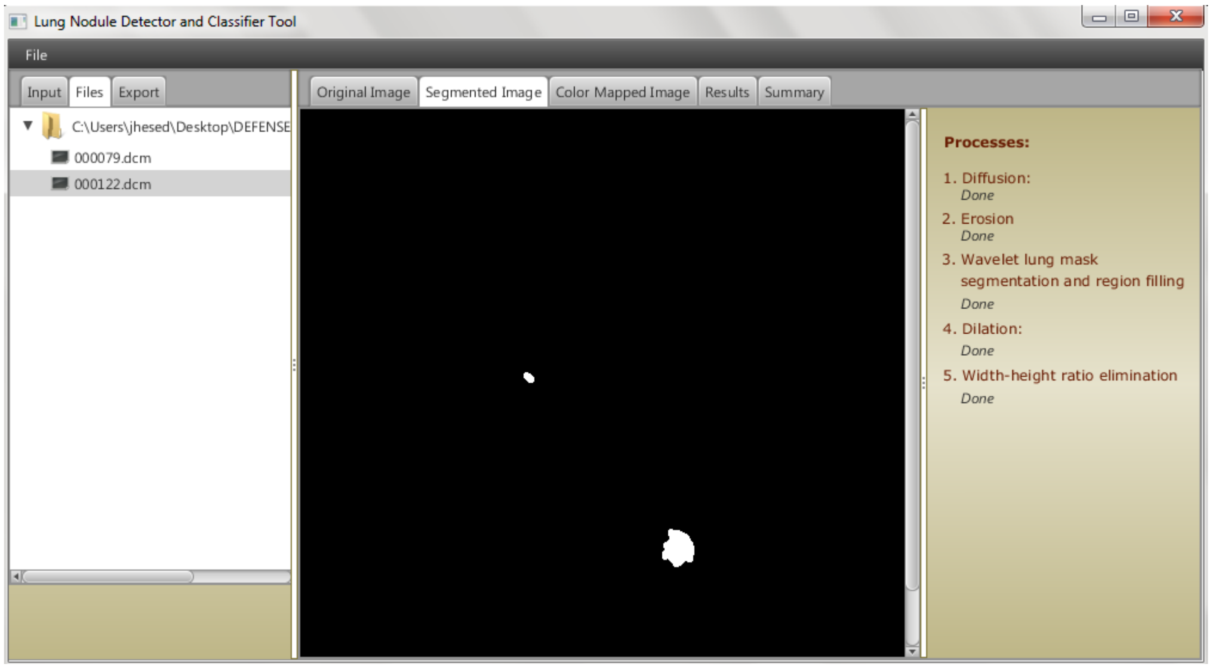


Fig.33 Internal pre-processing steps; diffusion (left), binarization (center), wavelet edge detection (right), LNDCT



Processes:

1. Diffusion:
   Done
2. Erosion
   Done
3. Wavelet lung mask segmentation and region filling
   Done
4. Dilation:
   Done
5. Width-height ratio elimination
   Done

Fig.34 Segmented nodule, LNDCT

The processes the image undergone can be seen in the information panel at the right. Because of the very big possibility that there may be multiple nodules in a single image, the nodules are mapped according to the colors given to them. This can be seen in "color mapped image" tab as shown in figure 35. The colors of the predicted malignant and benign nodules are also indicated in the information panel on the right.

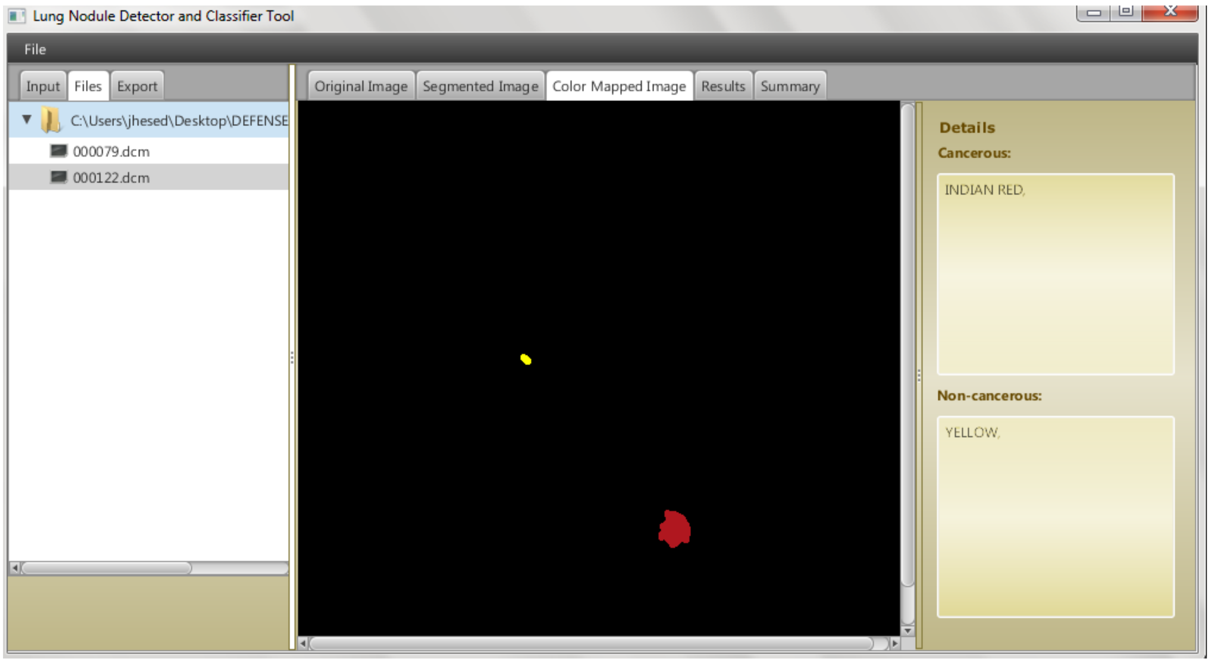
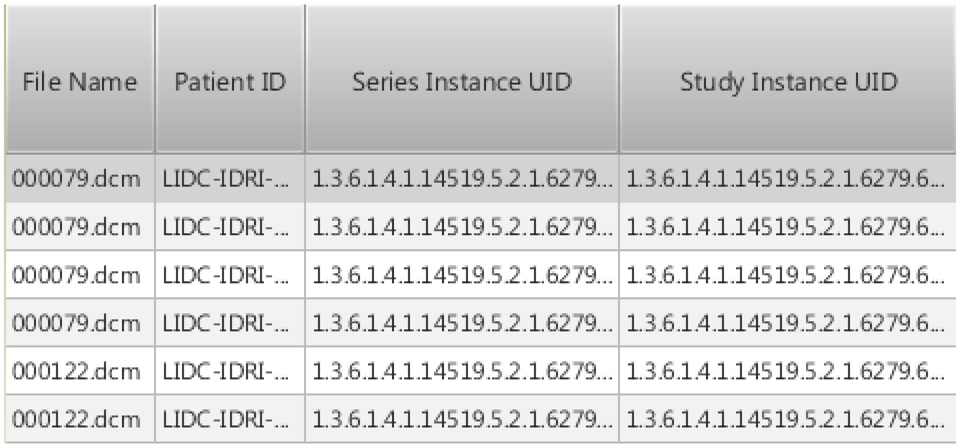


Fig.35 Color mapped nodules (example), LNDCT

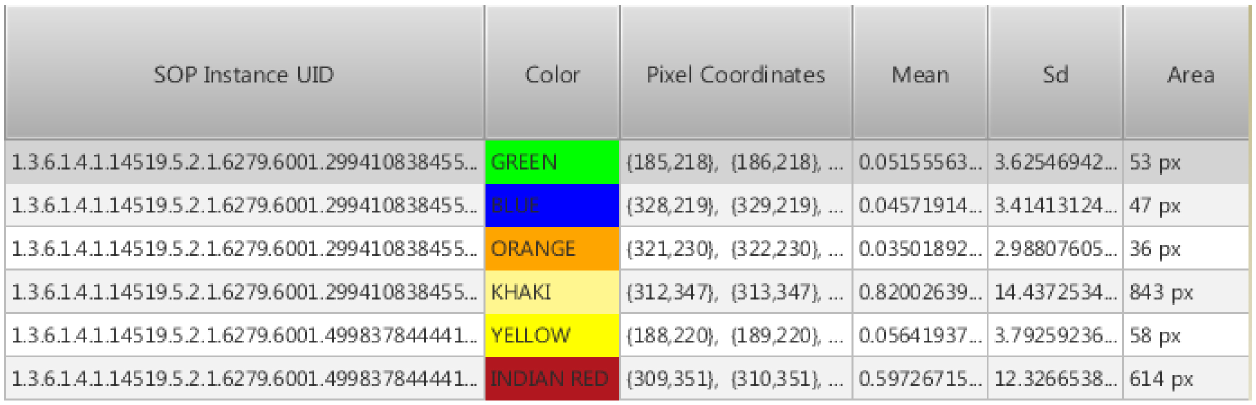
The extracted nodules will be compared to the parsed xml file using the pixel coordinates and the image SOP UID, which are unique for every image.

The results, including the computed information for each nodule, can be found under the results tab. These can be seen in figures 36-38.



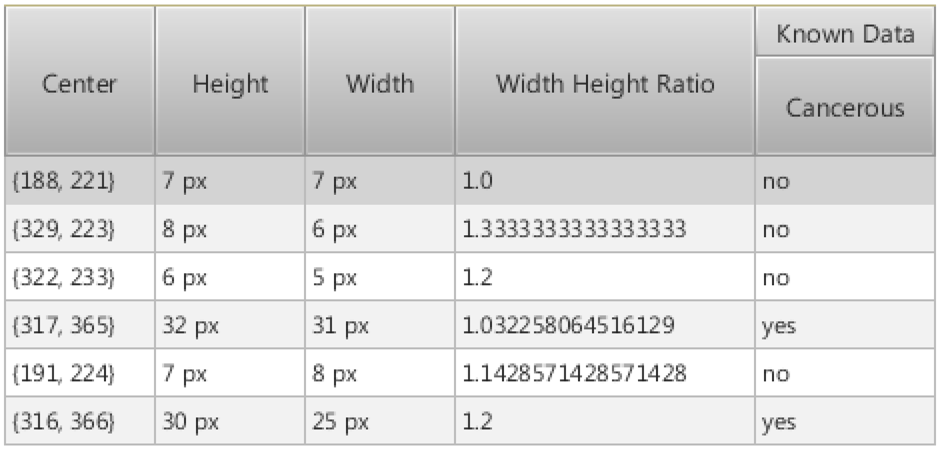
File Name	Patient ID	Series Instance UID	Study Instance UID
000079.dcm	LIDC-IDRI-...	1.3.6.1.4.1.14519.5.2.1.6279...	1.3.6.1.4.1.14519.5.2.1.6279.6...
000079.dcm	LIDC-IDRI-...	1.3.6.1.4.1.14519.5.2.1.6279...	1.3.6.1.4.1.14519.5.2.1.6279.6...
000079.dcm	LIDC-IDRI-...	1.3.6.1.4.1.14519.5.2.1.6279...	1.3.6.1.4.1.14519.5.2.1.6279.6...
000079.dcm	LIDC-IDRI-...	1.3.6.1.4.1.14519.5.2.1.6279...	1.3.6.1.4.1.14519.5.2.1.6279.6...
000122.dcm	LIDC-IDRI-...	1.3.6.1.4.1.14519.5.2.1.6279...	1.3.6.1.4.1.14519.5.2.1.6279.6...
000122.dcm	LIDC-IDRI-...	1.3.6.1.4.1.14519.5.2.1.6279...	1.3.6.1.4.1.14519.5.2.1.6279.6...

Fig.36 Results tab 1, LNDCT



SOP Instance UID	Color	Pixel Coordinates	Mean	Sd	Area
1.3.6.1.4.1.14519.5.2.1.6279.6001.299410838455...	GREEN	{185,218}, {186,218}, ...	0.05155563...	3.62546942...	53 px
1.3.6.1.4.1.14519.5.2.1.6279.6001.299410838455...	BLUE	{328,219}, {329,219}, ...	0.04571914...	3.41413124...	47 px
1.3.6.1.4.1.14519.5.2.1.6279.6001.299410838455...	ORANGE	{321,230}, {322,230}, ...	0.03501892...	2.98807605...	36 px
1.3.6.1.4.1.14519.5.2.1.6279.6001.299410838455...	KHAKI	{312,347}, {313,347}, ...	0.82002639...	14.4372534...	843 px
1.3.6.1.4.1.14519.5.2.1.6279.6001.499837844441...	YELLOW	{188,220}, {189,220}, ...	0.05641937...	3.79259236...	58 px
1.3.6.1.4.1.14519.5.2.1.6279.6001.499837844441...	INDIAN RED	{309,351}, {310,351}, ...	0.59726715...	12.3266538...	614 px

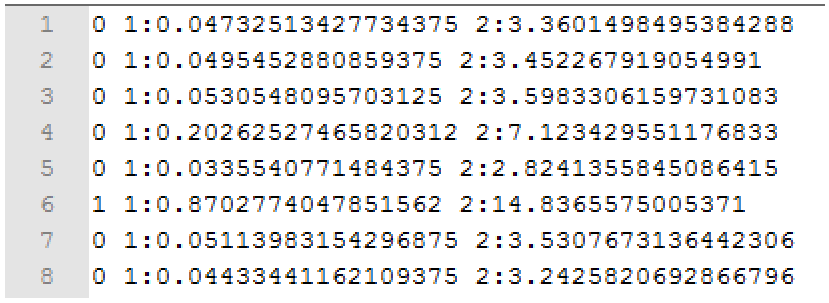
Fig.37 Results tab 2, LNDCT



Center	Height	Width	Width Height Ratio	Known Data
				Cancerous
{188, 221}	7 px	7 px	1.0	no
{329, 223}	8 px	6 px	1.3333333333333333	no
{322, 233}	6 px	5 px	1.2	no
{317, 365}	32 px	31 px	1.032258064516129	yes
{191, 224}	7 px	8 px	1.1428571428571428	no
{316, 366}	30 px	25 px	1.2	yes

Fig.38 Results tab 3, LNDCT

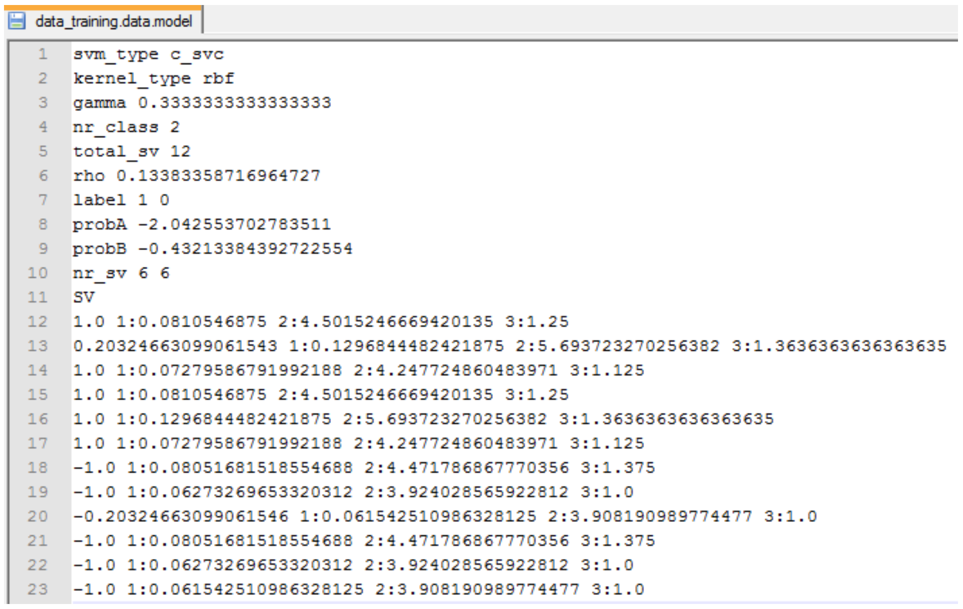
When training, the values in table column predicted data and prediction are left NA or -0.0. The system will save a copy of the training data and model as previously discussed and as shown in figure 39. If the value of the first number for each line is 1, it is malignant. If it is 0, it is not. The features are listed in $number{:}feature$ format. The first feature is mean, followed by standard deviation.



```
0 1:0.04732513427734375 2:3.3601498495384288
0 1:0.0495452880859375 2:3.452267919054991
0 1:0.0530548095703125 2:3.5983306159731083
0 1:0.20262527465820312 2:7.123429551176833
0 1:0.0335540771484375 2:2.8241355845086415
1 1:0.8702774047851562 2:14.8365575005371
0 1:0.05113983154296875 2:3.5307673136442306
0 1:0.04433441162109375 2:3.2425820692866796
```

Fig.39 training data saved in data file, LNDCT

A model file will be generated as shown in figure 40. This will be used for future prediction of the system.



```
svm_type c_svc
kernel_type rbf
gamma 0.3333333333333333
nr_class 2
total_sv 12
rho 0.13383358716964727
label 1 0
probA -2.042553702783511
probB -0.43213384392722554
nr_sv 6 6
SV
1.0 1:0.0810546875 2:4.5015246669420135 3:1.25
0.20324663099061543 1:0.1296844482421875 2:5.693723270256382 3:1.3636363636363635
1.0 1:0.07279586791992188 2:4.247724860483971 3:1.125
1.0 1:0.0810546875 2:4.5015246669420135 3:1.25
1.0 1:0.1296844482421875 2:5.693723270256382 3:1.3636363636363635
1.0 1:0.07279586791992188 2:4.247724860483971 3:1.125
-1.0 1:0.08051681518554688 2:4.471786867770356 3:1.375
-1.0 1:0.06273269653320312 2:3.924028565922812 3:1.0
-0.20324663099061546 1:0.061542510986328125 2:3.908190989774477 3:1.0
-1.0 1:0.08051681518554688 2:4.471786867770356 3:1.375
-1.0 1:0.06273269653320312 2:3.924028565922812 3:1.0
-1.0 1:0.061542510986328125 2:3.908190989774477 3:1.0
```

Fig.40 Model file, LNDCT thru LibSVM

Training can also be done by using pre-computed feature values. Click on the "data" text field and press enter. A pop-up dialog will appear. Choose the data file and open it. Its location will be placed in the text field. Click the "train button" and LNDCT will generate a model file in the same location loaded.

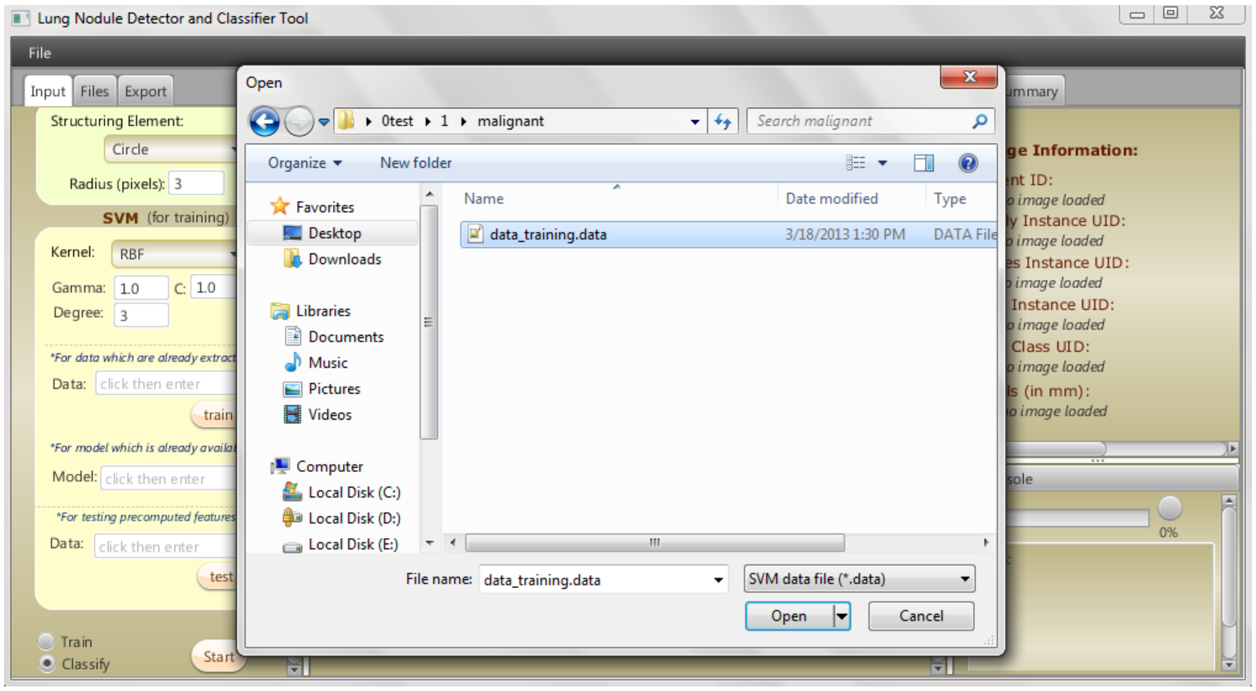
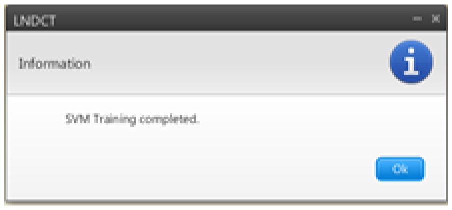


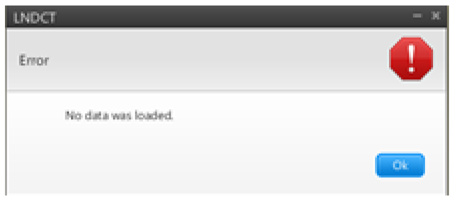
Fig.41 Example of loading pre-computed data features, LNDCT



SVM Training completed.

Fig.42 Training successful notification message, LNDCT

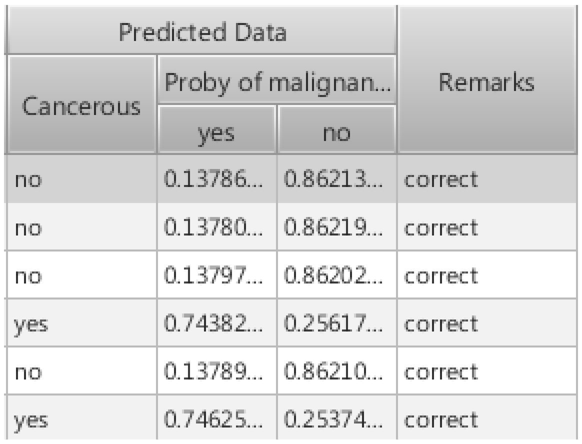
Pre-computed values of features can also be tested using the same steps but by clicking the “test” button and the text field above it. If in case the “train/test” button was clicked, but no data was loaded, an error message will pop-up as shown in figure 43.



No data was loaded.

Fig.43 training without loading pre-computed data error message, LNDCT

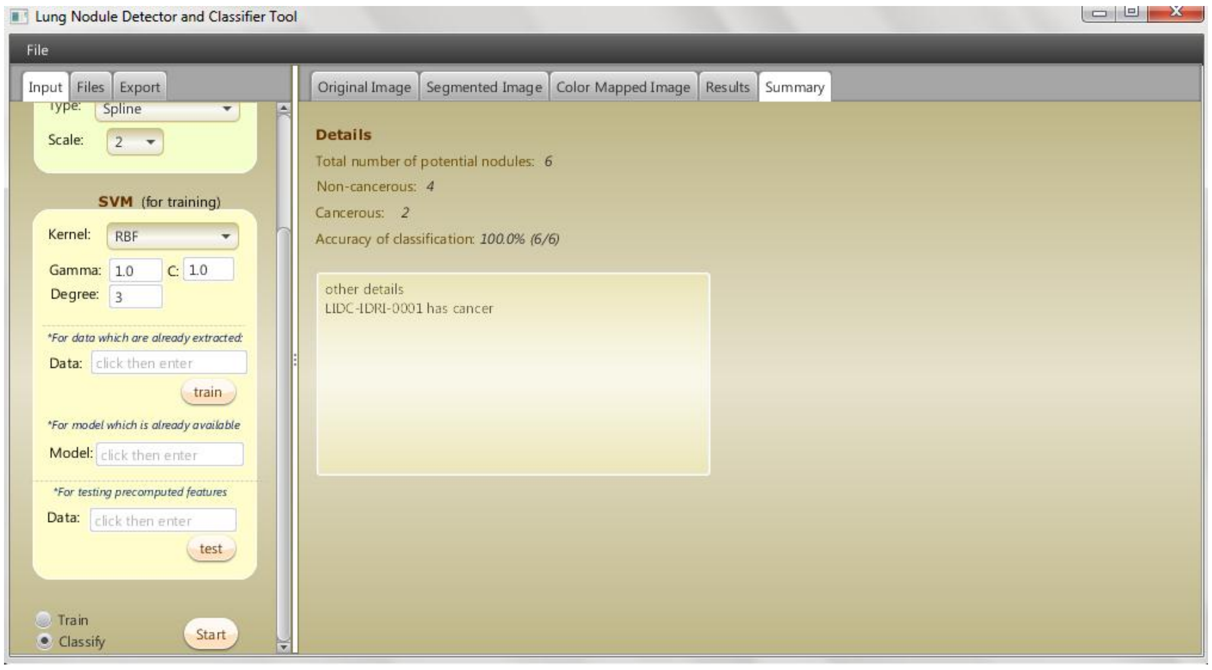
Using LNDCT as a lung nodule detector and classifier tool is the similar when training. But the "predicted data" and "remarks" under results tab will have values as shown in figure 44.



Predicted Data			Remarks
Cancerous	Proby of malignan...		
	yes	no	
no	0.13786...	0.86213...	correct
no	0.13780...	0.86219...	correct
no	0.13797...	0.86202...	correct
yes	0.74382...	0.25617...	correct
no	0.13789...	0.86210...	correct
yes	0.74625...	0.25374...	correct

Fig.44 Classification results, LNDCT

The summary of the whole process is shown under the "summary" tab. It includes the details about the total number of cancerous and non-cancerous nodules, the accuracy of classification based from the xml cross validation file, and the diagnosis for the patient. These details are illustrated in figure 45.



Type: Spline
Scale: 2

SVM (for training)

Kernel: RBF
Gamma: 1.0 C: 1.0
Degree: 3

*For data which are already extracted:
Data: click then enter

*For model which is already available
Model: click then enter

*For testing precomputed features
Data: click then enter

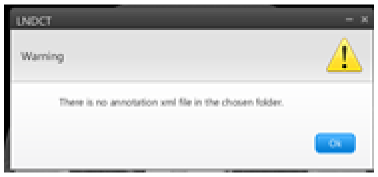
Train
● Classify

Details
Total number of potential nodules: *6*
Non-cancerous: *4*
Cancerous: *2*
Accuracy of classification: *100.0% (6/6)*

other details
LIDC-IDRI-0001 has cancer

Fig.45 Summary tab, LNDCT

Xml files should be placed in the same folder which contains the dicom files. These will be used for cross-validation of the predicted data and known data. It will also be used to categorize whether the nodules are cancerous or not in the training phase. If the folder loaded does not contain the xml file, a warning message will pop-up as shown in figure 46.

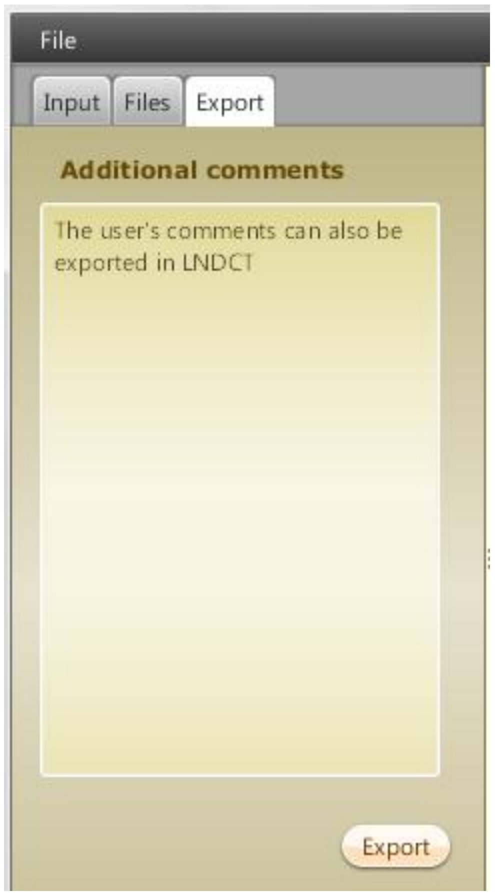


Warning

There is no annotation xml file in the chosen folder.

Fig.46 Warning message, LNCDT

Both the training and testing results can be exported in pdf format. To do this, access the export tab as shown in figure 47, and click export button.



The user's comments can also be exported in LNDCT

Fig.47 Example export, LNDCT

The generated PDF file will be placed in the same folder. Its name will be "LNDCT report <date>". The PDF file will contain the original and the color-mapped segmented images, and the computed values for each nodule. The information about the colors of the cancerous and non-cancerous nodules based on the color-mapped image is also included (see the circled portion in figure 49). The PDF file automatically pops-up after the export process was finished.

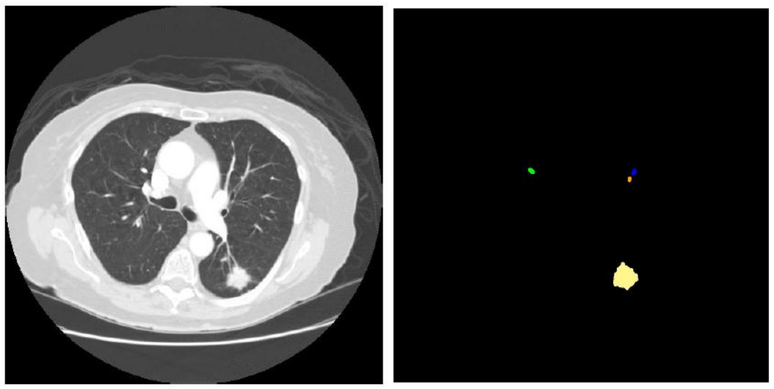
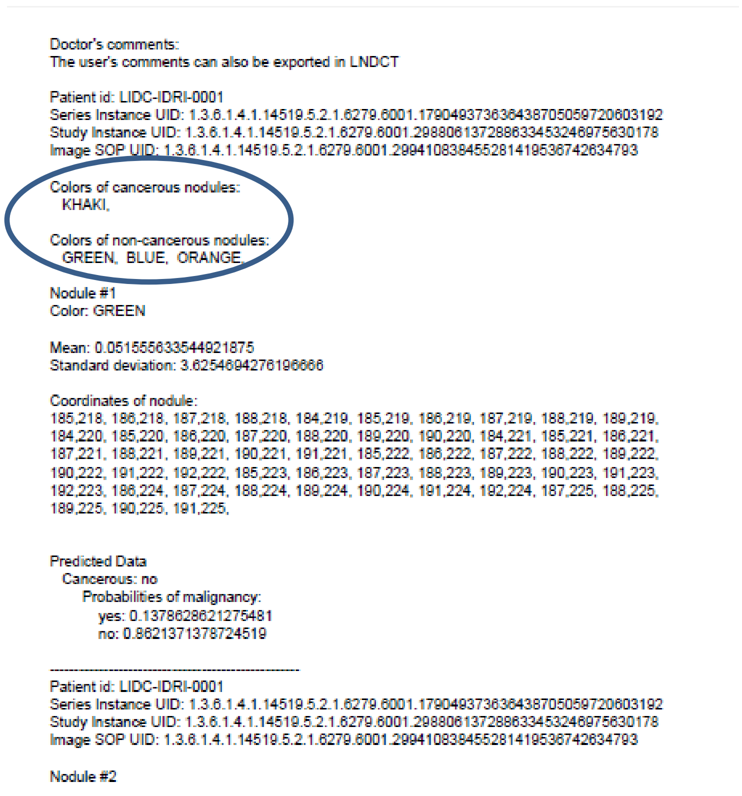


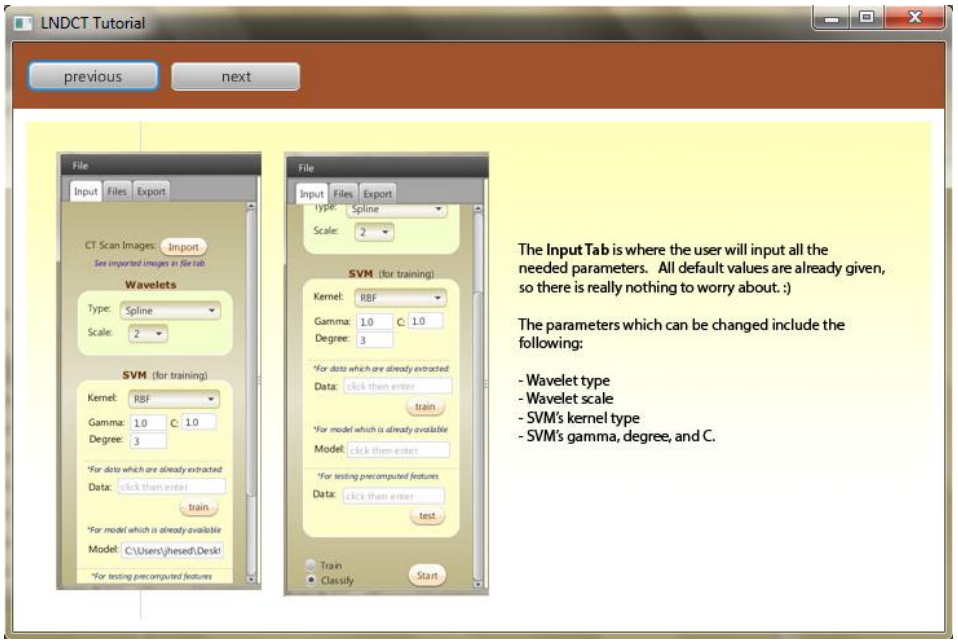
Fig.48 Exported Original image (left) and color-mapped image (right), LNDCT



```
Doctor's comments:
The user's comments can also be exported in LNDCT

Patient id: LIDC-IDRI-0001
Series Instance UID: 1.3.6.1.4.1.14519.5.2.1.6279.6001.179049373636438705059720603192
Study Instance UID: 1.3.6.1.4.1.14519.5.2.1.6279.6001.298806137288633453246975630178
Image SOP UID: 1.3.6.1.4.1.14519.5.2.1.6279.6001.299410838455281419536742634793

Colors of cancerous nodules:
  KHAKI,

Colors of non-cancerous nodules:
  GREEN,  BLUE,  ORANGE,

Nodule #1
Color: GREEN

Mean: 0.051555633544921875
Standard deviation: 3.6254694276196666

Coordinates of nodule:
185,218, 186,218, 187,218, 188,218, 184,219, 185,219, 186,219, 187,219, 188,219, 189,219,
184,220, 185,220, 186,220, 187,220, 188,220, 189,220, 190,220, 184,221, 185,221, 186,221,
187,221, 188,221, 189,221, 190,221, 191,221, 185,222, 186,222, 187,222, 188,222, 189,222,
190,222, 191,222, 192,222, 185,223, 186,223, 187,223, 188,223, 189,223, 190,223, 191,223,
192,223, 186,224, 187,224, 188,224, 189,224, 190,224, 191,224, 192,224, 187,225, 188,225,
189,225, 190,225, 191,225,


Predicted Data
  Cancerous: no
      Probabilities of malignancy:
         yes: 0.1378628621275481
         no: 0.8621371378724519

--------------------------------------------------
Patient id: LIDC-IDRI-0001
Series Instance UID: 1.3.6.1.4.1.14519.5.2.1.6279.6001.179049373636438705059720603192
Study Instance UID: 1.3.6.1.4.1.14519.5.2.1.6279.6001.298806137288633453246975630178
Image SOP UID: 1.3.6.1.4.1.14519.5.2.1.6279.6001.299410838455281419536742634793

Nodule #2
```

Fig.49 Example of generated file, LNDCT

For beginners, a tutorial can also be accessed under the File menu. Click on tutorial, and the tutorial window will appear. Click the "next" button to see the next page of the tutorial. These are shown from figure 50 to 63.

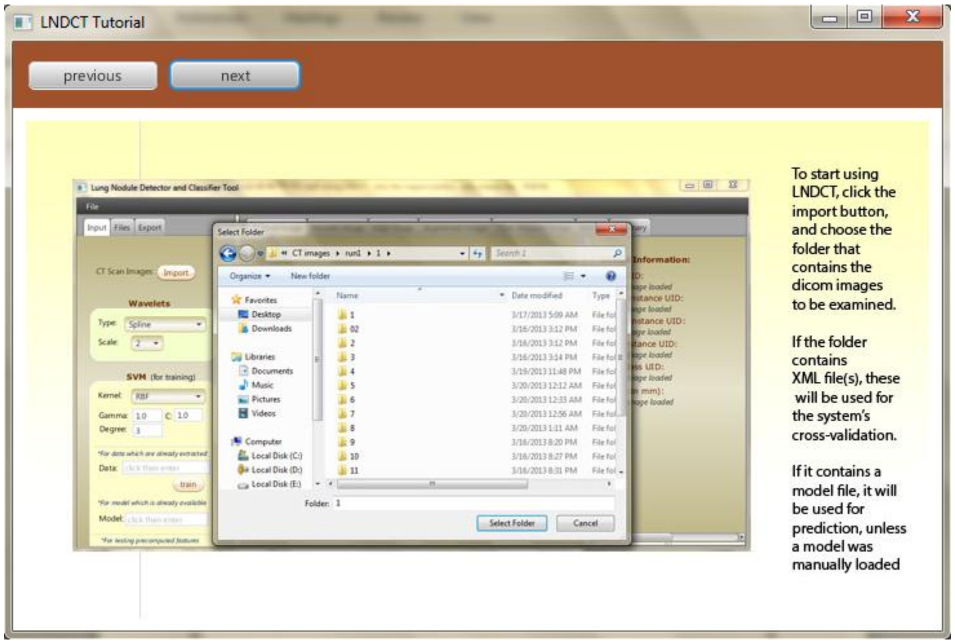


The **Input Tab** is where the user will input all the needed parameters. All default values are already given, so there is really nothing to worry about. :)

The parameters which can be changed include the following:

- Wavelet type
- Wavelet scale
- SVM's kernel type
- SVM's gamma, degree, and C.

Fig.50 Page 1 of Tutorial, LNDCT

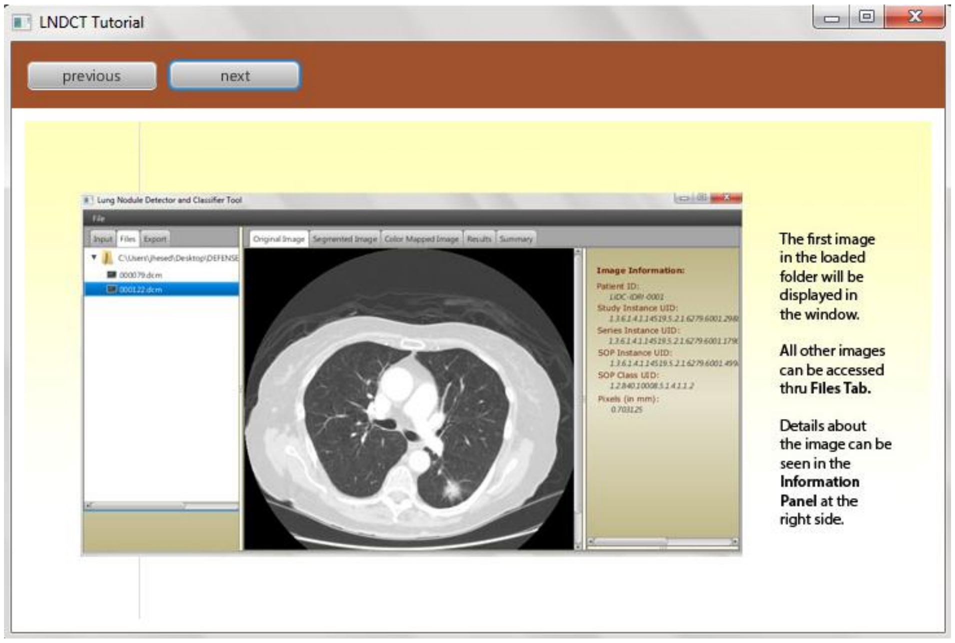


To start using LNDCT, click the import button, and choose the folder that contains the dicom images to be examined.

If the folder contains XML file(s), these will be used for the system's cross-validation.

If it contains a model file, it will be used for prediction, unless a model was manually loaded

Fig.51 Page 2 of Tutorial, LNDCT

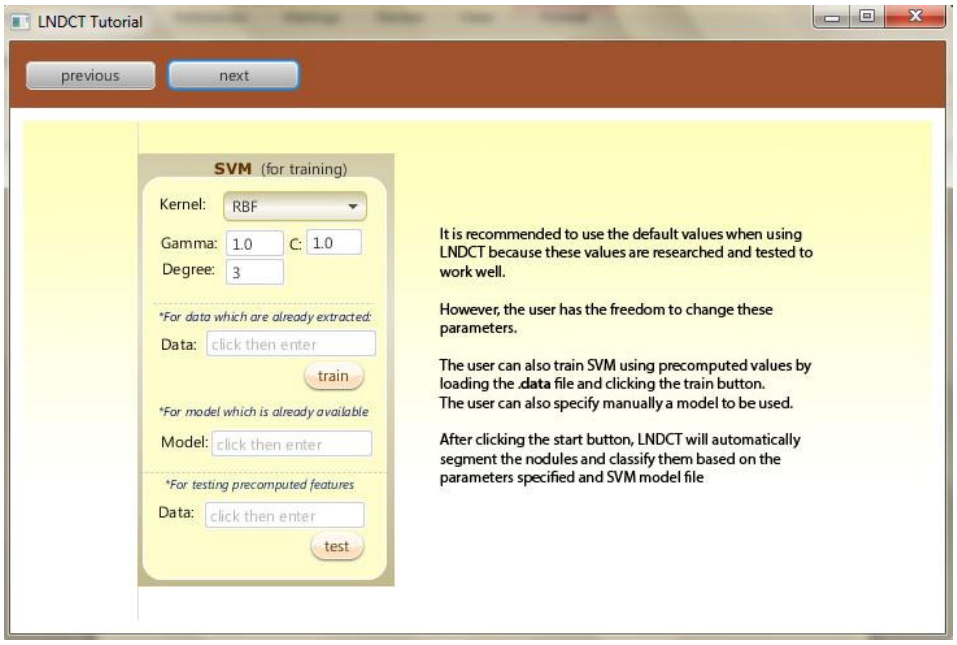


The first image in the loaded folder will be displayed in the window.

All other images can be accessed thru **Files Tab.**

Details about the image can be seen in the **Information Panel** at the right side.

Fig.52 Page 3 of Tutorial, LNDCT



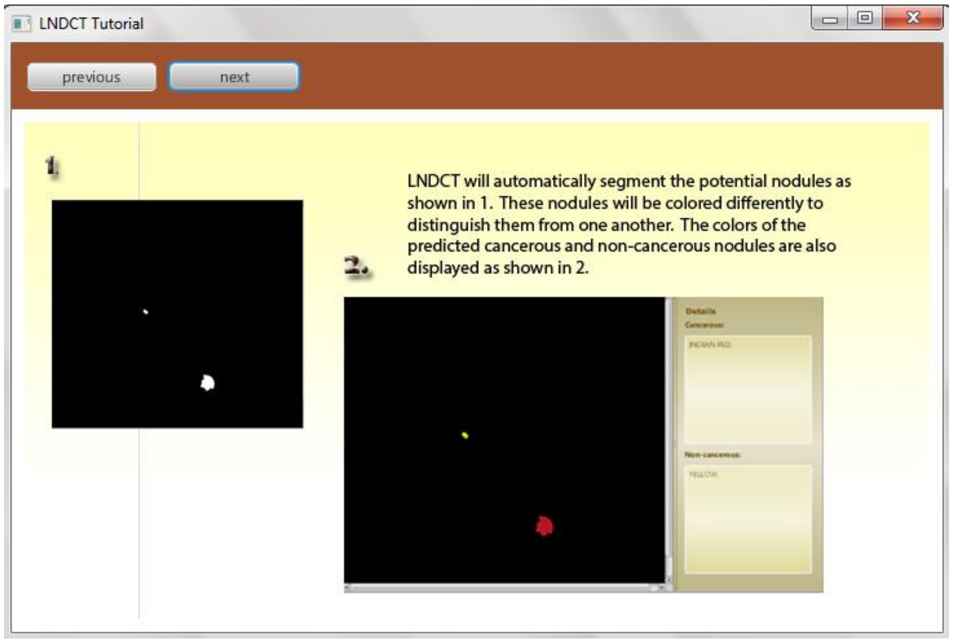
It is recommended to use the default values when using LNDCT because these values are researched and tested to work well.

However, the user has the freedom to change these parameters.

The user can also train SVM using precomputed values by loading the **.data** file and clicking the train button.
The user can also specify manually a model to be used.

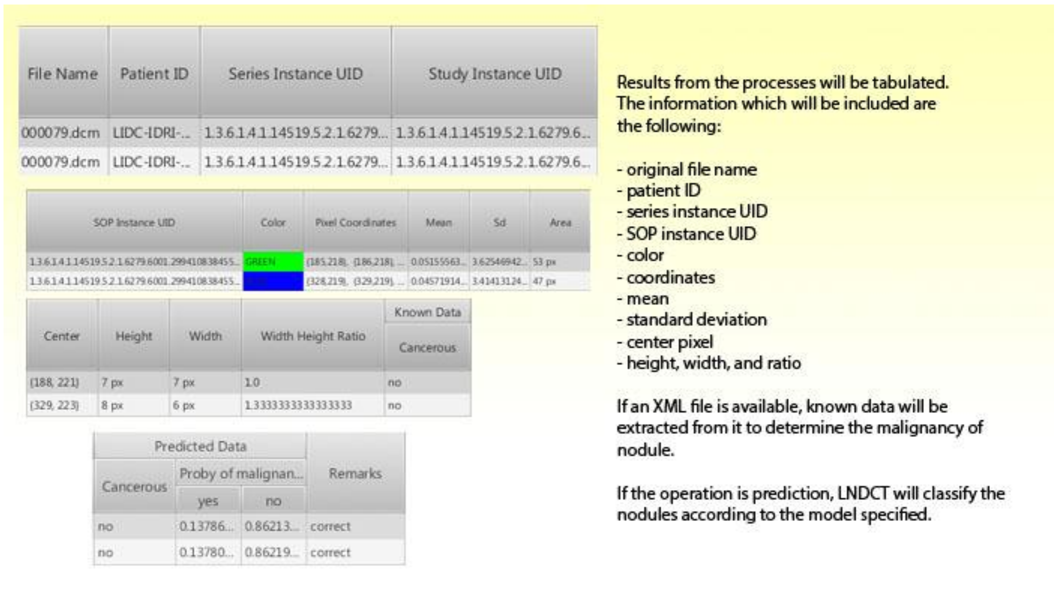
After clicking the start button, LNDCT will automatically segment the nodules and classify them based on the parameters specified and SVM model file

Fig.53 Page 4 of Tutorial, LNDCT



LNDCT will automatically segment the potential nodules as shown in 1. These nodules will be colored differently to distinguish them from one another. The colors of the predicted cancerous and non-cancerous nodules are also displayed as shown in 2.

Fig.54 Page 5 of Tutorial, LNDCT



File Name	Patient ID	Series Instance UID	Study Instance UID
000079.dcm	LIDC-IDRI-...	1.3.6.1.4.1.14519.5.2.1.6279...	1.3.6.1.4.1.14519.5.2.1.6279.6...
000079.dcm	LIDC-IDRI-...	1.3.6.1.4.1.14519.5.2.1.6279...	1.3.6.1.4.1.14519.5.2.1.6279.6...

SOP Instance UID	Color	Pixel Coordinates	Mean	Sd	Area
1.3.6.1.4.1.14519.5.2.1.6279.6001.299410838455...	🟩 GREEN	(185,218), (186,218), ...	0.05155563...	3.62546942...	53 px
1.3.6.1.4.1.14519.5.2.1.6279.6001.299410838455...	🟦	(328,219), (329,219), ...	0.04571914...	3.41413124...	47 px

Center	Height	Width	Width Height Ratio	Known Data
				Cancerous
(188, 221)	7 px	7 px	1.0	no
(329, 223)	8 px	6 px	1.3333333333333333	no

Predicted Data			Remarks
Cancerous	Proby of malignan...		
	yes	no	
no	0.13786...	0.86213...	correct
no	0.13780...	0.86219...	correct

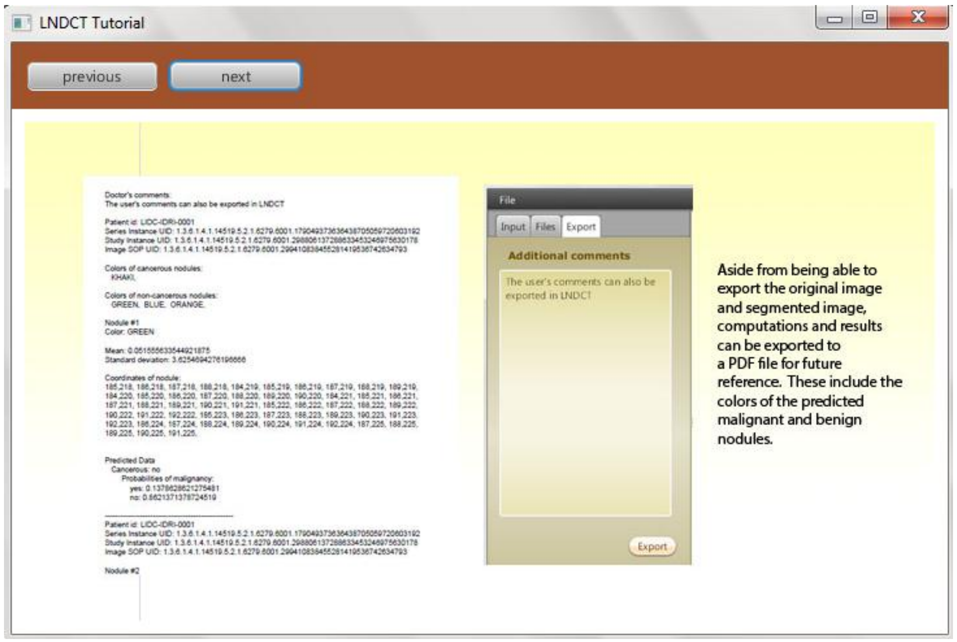
Results from the processes will be tabulated. The information which will be included are the following:

- original file name
- patient ID
- series instance UID
- SOP instance UID
- color
- coordinates
- mean
- standard deviation
- center pixel
- height, width, and ratio

If an XML file is available, known data will be extracted from it to determine the malignancy of nodule.

If the operation is prediction, LNDCT will classify the nodules according to the model specified.

Fig. 55 Page 6 of Tutorial, LNDCT



Doctor's comments:
The user's comments can also be exported in LNDCT

Patient id: LIDC-IDRI-0001
Series Instance UID: 1.3.6.1.4.1.14519.5.2.1.6279.6001.179049373636438705059720603192
Study Instance UID: 1.3.6.1.4.1.14519.5.2.1.6279.6001.298806137288633453246975630178
Image SOP UID: 1.3.6.1.4.1.14519.5.2.1.6279.6001.299410838455281419536742634793

Colors of cancerous nodules:
KHAKI,

Colors of non-cancerous nodules:
GREEN, BLUE, ORANGE,

Nodule #1
Color: GREEN

Mean: 0.051555633544921875
Standard deviation: 3.6254694276196666

Coordinates of nodule:
185,218, 186,218, 187,218, 188,218, 184,219, 185,219, 186,219, 187,219, 188,219, 189,219,
184,220, 185,220, 186,220, 187,220, 188,220, 189,220, 190,220, 184,221, 185,221, 186,221,
187,221, 188,221, 189,221, 190,221, 191,221, 185,222, 186,222, 187,222, 188,222, 189,222,
190,222, 191,222, 192,222, 185,223, 186,223, 187,223, 188,223, 189,223, 190,223, 191,223,
192,223, 186,224, 187,224, 188,224, 189,224, 190,224, 191,224, 192,224, 187,225, 188,225,
189,225, 190,225, 191,225,

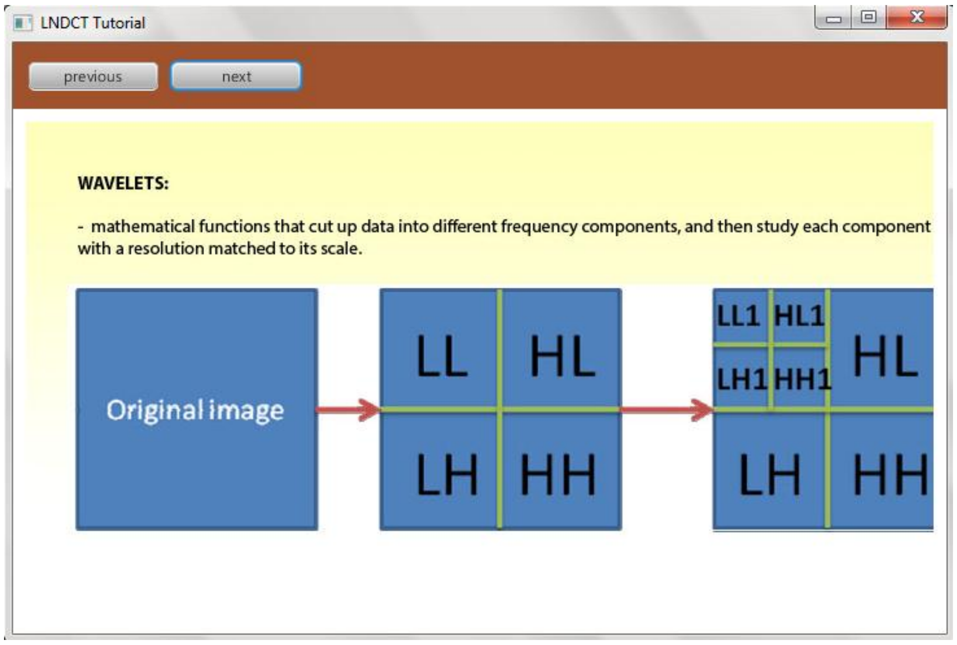
Predicted Data
Cancerous: no
Probabilities of malignancy:
yes: 0.1378628621275481
no: 0.8621371378724519

Patient id: LIDC-IDRI-0001
Series Instance UID: 1.3.6.1.4.1.14519.5.2.1.6279.6001.179049373636438705059720603192
Study Instance UID: 1.3.6.1.4.1.14519.5.2.1.6279.6001.298806137288633453246975630178
Image SOP UID: 1.3.6.1.4.1.14519.5.2.1.6279.6001.299410838455281419536742634793

Nodule #2

Aside from being able to export the original image and segmented image, computations and results can be exported to a PDF file for future reference. These include the colors of the predicted malignant and benign nodules.

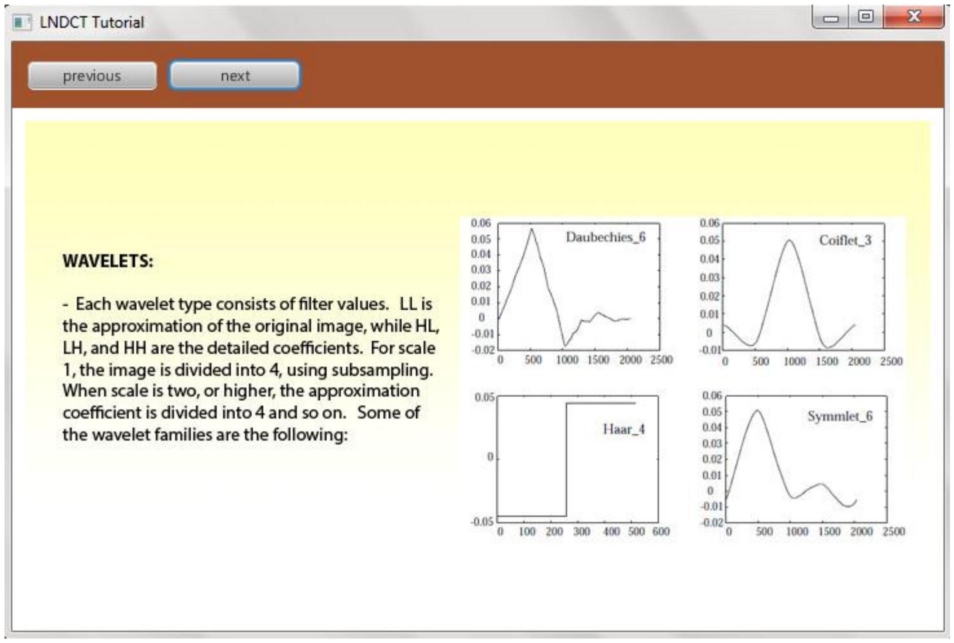
Fig.56 Page 7 of Tutorial, LNDCT



WAVELETS:

- mathematical functions that cut up data into different frequency components, and then study each component with a resolution matched to its scale.

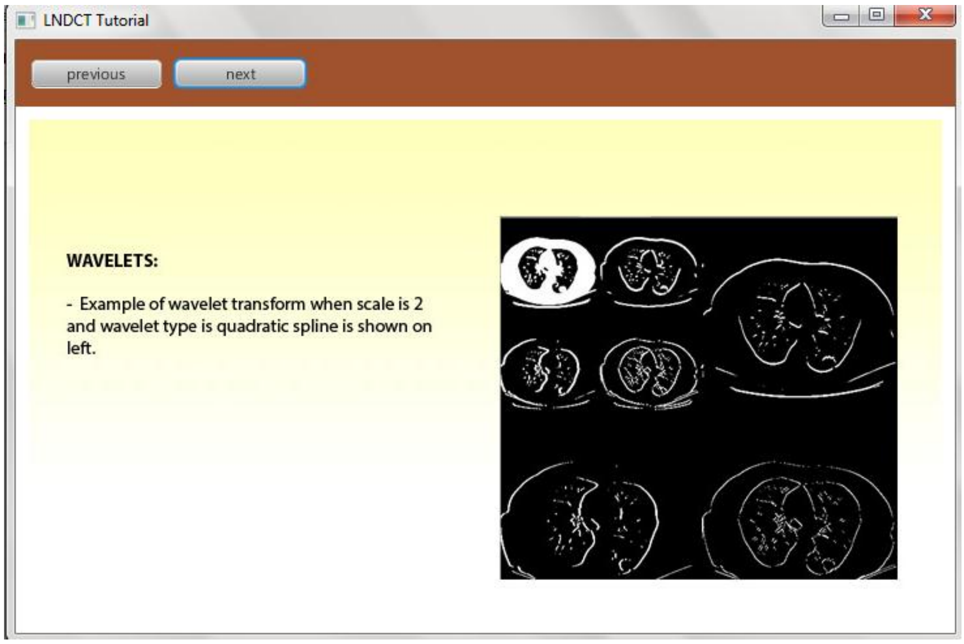
Fig.57 Page 8 of Tutorial, LNDCT



WAVELETS:

- Each wavelet type consists of filter values. LL is the approximation of the original image, while HL, LH, and HH are the detailed coefficients. For scale 1, the image is divided into 4, using subsampling. When scale is two, or higher, the approximation coefficient is divided into 4 and so on. Some of the wavelet families are the following:

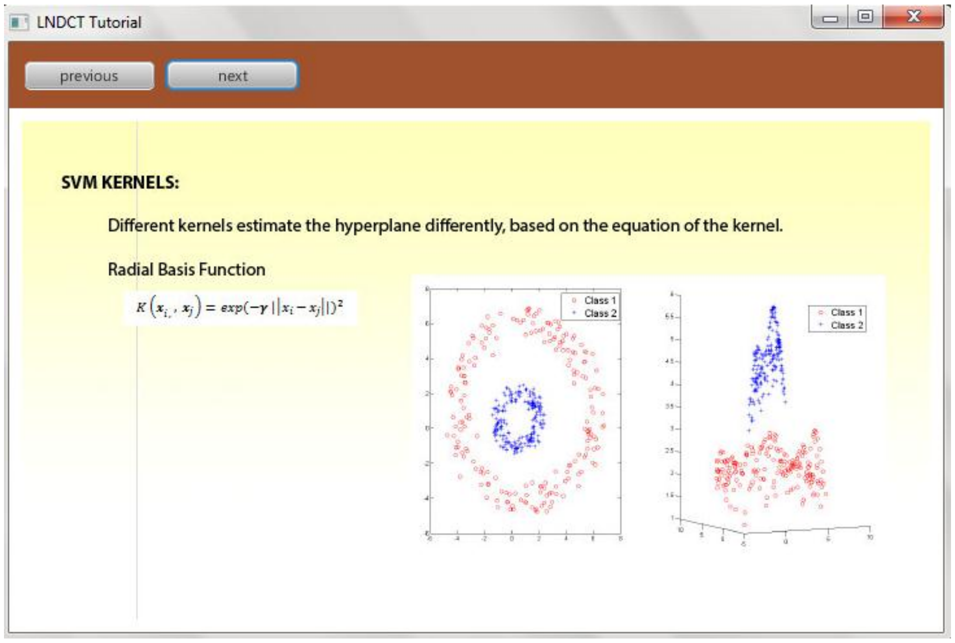
Fig.58 Page 9 of Tutorial, LNDCT



WAVELETS:

- Example of wavelet transform when scale is 2 and wavelet type is quadratic spline is shown on left.

Fig.59 Page 10 of Tutorial, LNDCT



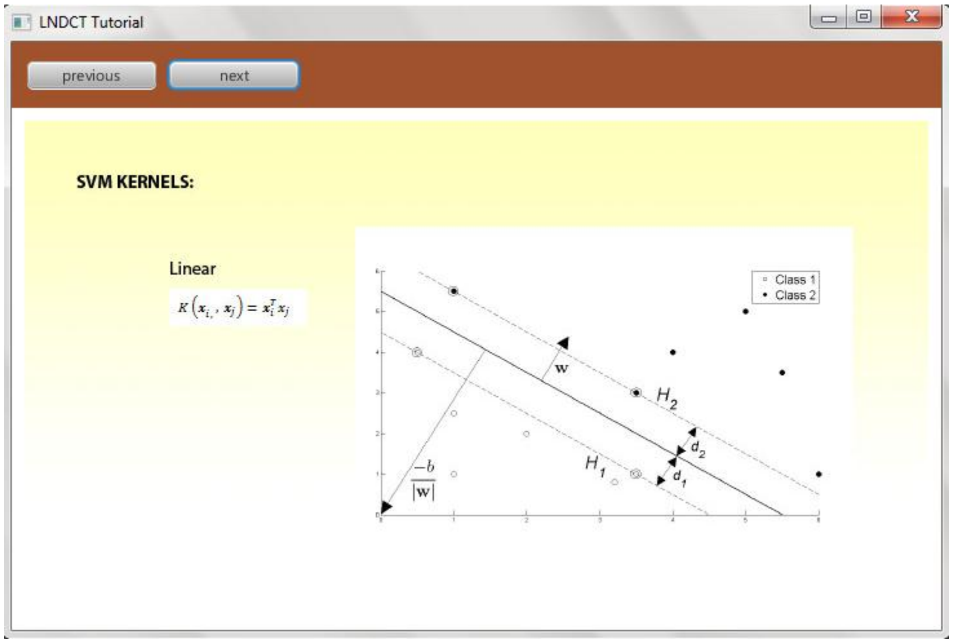
SVM KERNELS:

Different kernels estimate the hyperplane differently, based on the equation of the kernel.

Radial Basis Function

$$K\left(x_{i,}, x_j\right) = exp(-\gamma\,||x_i - x_j||)^2$$

Fig.60 Page 11 of Tutorial, LNDCT


LNDCT Tutorial

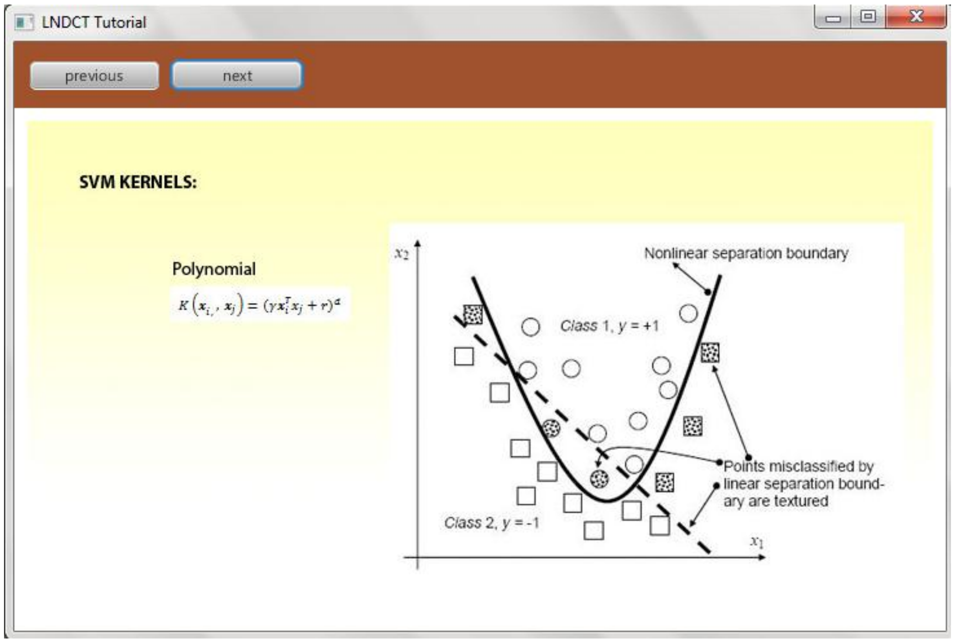
previous next

SVM KERNELS:

Linear

$K(x_i, x_j) = x_i^T x_j$

Fig.61 Page 12 of Tutorial, LNDCT


LNDCT Tutorial

previous next

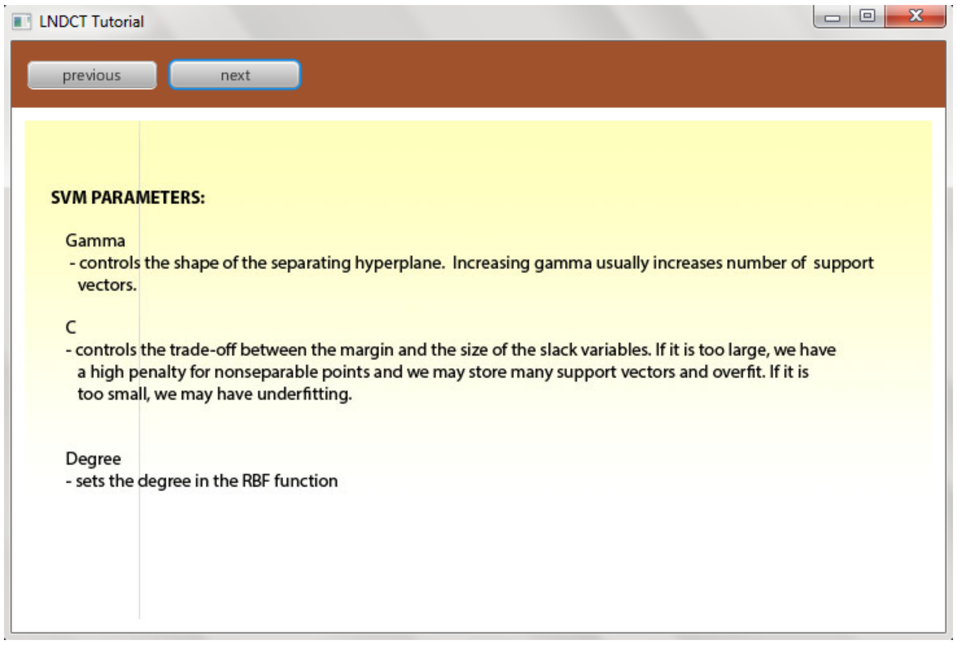
SVM KERNELS:

Polynomial

$K(x_i, x_j) = (\gamma x_i^T x_j + r)^d$

x_2
Nonlinear separation boundary
Class 1, y = +1
Points misclassified by linear separation boundary are textured
Class 2, y = -1
x_1

Fig.62 Page 13 of Tutorial, LNDCT



SVM PARAMETERS:

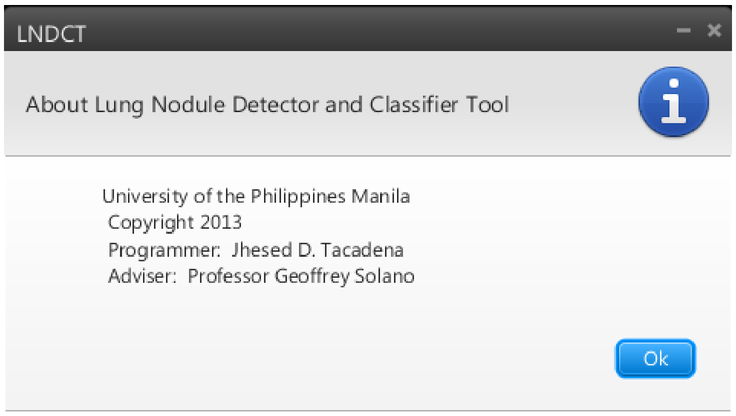
Gamma
- controls the shape of the separating hyperplane. Increasing gamma usually increases number of support vectors.

C
- controls the trade-off between the margin and the size of the slack variables. If it is too large, we have a high penalty for nonseparable points and we may store many support vectors and overfit. If it is too small, we may have underfitting.

Degree
- sets the degree in the RBF function

Fig.63 Page 14 of Tutorial, LNDCT

Information about the software can be seen by accessing file -> about.



About Lung Nodule Detector and Classifier Tool

University of the Philippines Manila
Copyright 2013
Programmer: Jhesed D. Tacadena
Adviser: Professor Geoffrey Solano

Fig.64 About dialog box, LNDCT

VI. DISCUSSION

The Lung Nodule Detector and Classifier Tool can be used to aid physicians in recognizing the potential nodules. Its user interface is friendly, with default values already given. The user can start using LNDCT by just loading the folder which contains the dicom files and the xml validation file (if available). When the user click the “start” button, LNDCT will automatically segment all potential nodules, and try to classify these using the SVM model file. LNDCT’s SVM training and testing phases have similar procedures, except for the final result generated.

LNDCT’s algorithm for lung nodule detection and classification involves the following steps: pre-processing using binarization and diffusion, multi-scale wavelet transform and edge detection, morphological operations, and support vector machines. Segmented nodules are colored differently to distinguish them from one another. It can be seen under “color-mapped image” tab.

Multi-scale edge detection is done by using wavelets. The wavelet families available in LNDCT are spline and Daubechies. The edge of the image will be obtained by removing the approximation part of the wavelet, and applying inverse wavelet transform to the detail coefficients. Edges obtained from the image will help in segmenting nodules attached to edges.

Morphological operations include erosion, dilation, and region filling. Erosion is performed to eliminate very small particles in the image. Region filling is done to remove the lung mask, while considering the nodules attached to it by using wavelets. Dilation is performed to smoothen the nodules.

Segmented candidate nodules will be removed if the width-height ratio is greater than 1.5.

The “result tab” will contain the information regarding each nodule. Its series instance UID, study instance UID, image SOP UID, patient ID, coordinates, center coordinate, and the computed values of mean, standard deviation, area, height, width and its ratio. Each nodule is given a different color and shown in “color-mapped image” tab to distinguish them from one another. The detected nodule will be cross validated to the XML file to know if it is cancerous or not. LNDCT will save the features of the nodule in a “data” file. If the operation chosen was training, LibSVM will generate a “model” file based on the segmented nodules and their features. If the operation chosen is “classifying”, LNDCT will predict the malignancy of the nodule based on the model file loaded. Model file can be loaded manually. It can also be placed in the same folder and LNDCT will automatically use it as a model if no model file was defined. The probability of malignancy will also be displayed on the result tab. The prediction will also be validated if correct or wrong based on the loaded known data. LNDCT’s accuracy will be computed as correctly classified nodules divided by total number of nodules.

The “Summary” tab will contain the summary of all the information computed. These include the number of predicted cancerous and non-cancerous nodules detected and the diagnosis for the patient (whether he/she has cancer). If an XML validation file is available, the accuracy of the system will also be computed.

Three testing sets were used to test the accuracy of detection and classification using LNDCT. All CT scan images were taken from LIDC. Set A contains lung images where most of the sizes of the cancerous nodules are greater than 200 pixels. Set B contains a combination of different possible nodule sizes. Sets A and B are composed of disjoint CT scan images. These use an SVM model file obtained from training 600 randomly chosen nodules extracted using LNDCT. Set C contains random training and testing data which came from pre-extracted features computed using LNDCT.

CT scan images used in set A came from 17 patients. Each image contains at least one cancerous nodule. Three of the four undetected nodules are detected on another image slice from the same patient. The summary is shown below:

Number of patients: **17**
Number of images: **45**
Total number of detected nodules: **143**
Number of undetected nodules: **4** (non-solid, irregular shape)
Number of non-cancerous nodules: **95**
Number of cancerous nodules: **48**
Over-all accuracy: $\mathbf{93.87}$ % $(\mathbf{138/147})$

Cancerous nodule size in pixels	**Number of cancerous nodules**	**Correctly Classified**	**Misclassified**	**Accuracy**
size < 100	1	0	1	0%
$100 \leq$ size < 200	1	0	1	0%
size > 200	46	46	0	100%

Table 1. Testing Set A results, LNDCT

CT scan images used in set B also came from 17 patients. Each image contains at least one cancerous nodule. The summary is shown below:

Number of patients: **17**
Number of images: **36**
Total number of detected nodules: **71**
Number of undetected nodules: **4** (non-solid, irregular shape)
Number of non-cancerous nodules: **35**
Number of cancerous nodules: **36**
Over-all accuracy: $\mathbf{81.33}$ % $(\mathbf{61/75})$

Cancerous nodule size in pixels	**Number of cancerous nodules**	**Correctly Classified**	**Misclassified**	**Accuracy**
size < 100	6	0	6	0%
$100 \leq$ size < 200	9	6	3	67%
size > 200	21	21	0	100%

Table 2. Testing Set B results, LNDCT

Set C is composed of data from pre-extracted candidate nodules using LNDCT. A total of 872 nodules are in the data, where 178 are cancerous and 694 are not. A java program was created to randomly split these data three different times to 70% training and 30% testing. The results are shown below:

Random Number	**Result**	**Accuracy**
1	213/245	86.93877551020408%
2	217/245	88.57142857142857%
3	211/245	86.12244897959184%

Table 3. Testing C results, LNDCT

Using LNDCT has these advantages. Computer aided nodule detection and classification does not produce subjective results. Computers do not feel tiredness in doing computations. LNDCT can detect multiple nodules from multiple images. Using wavelets for edge detection can successfully determine the lung mask and the nodules attached to it. Using morphological operations with wavelets can immediately eliminate non-nodules, smoothen the image, and segment nodules. Using SVM with good model results to accurate classification.

Some cancerous nodules have similar features to non-cancerous ones. This results to some classification error to SVM.

VII. CONCLUSION

Lung Nodule Detector and Classifier Tool uses wavelets, morphological operations, and support vector machine in detecting and classifying malignant and benign lung nodules.

LNDCT can be used to aid physicians in determining and classifying potential nodules. It can help them in analyzing multiple CT scan images, and provide useful information that will help them in giving the final decision.

When the cancerous nodule size is greater than 200 pixels, the accuracy of LNDCT is reported to be 100%. When the data trained and tested is completely random, LNDCT's accuracy ranges from 86 to 89%. It is higher than the reported sensitivity of manual detection which has been reported to be 70%-75%. Hence, it can be used as a decision support software for detecting lung nodules.

VIII. Recommendation

Though LNDCT became successful in detecting and classifying nodules, it can still be improved. First, by training using more data sets (good data sets) on SVM which will result to better classification. Second, by adding additional classifier variables. LNDCT uses two variables based on [8]: mean and standard deviation. Adding another classification variables (with significant difference across malignant and non-malignant nodules), might boost the classification results of SVM. Third, people may also try adding some rule based elimination techniques to immediately eliminate the non-nodules, aside from width-height ratio.

X. Bibliography

[1] Cancer statistics: Worldwide. *wcrf.org.* Retrieved October 4, 2012, from http://www.wcrf.org/cancer_statistics/world_cancer_statistics.php

[2] Cancer. *who.int.* Retrieved October 4, 2012, from http://www.who.int/mediacentre/factsheets/fs297/en/

[3] Rapista, A. (2010, December 13). Lung cancer global mortality rate exceeds all type of cancers combined. *mb.com.ph.* Retrieved October 4, 2012, from http://www.mb.com.ph/node/292550/function.unserialize

[4] Lung cancer still the deadliest type of cancer. *philstar.com.* Retrieved October 4, 2012, from http://www.philstar.com/Article.aspx?articleId=634673&publication..

[5] Brandman, S., & Ko, J. (2011, May). Pulmonary Nodule Detection, Characterization, and Management With Mulitdetector Computed Tomography. *Journal of Thoracic Imaging Vol, 26(2)*

[6] Graps, A. (1995). An Introduction to Wavelets. *IEEE Computational Science and Engineering, 2*(2).

[7] Jaffar, A., Ishtiaq, M., Hussain, A., & Mirza, A. (2011). Wavelet-Based Color Image Segmentation using Self-Organizing Map Neural Network. *International Conference on Computer Engineering and Applications IPCSIT, 2,* 3.

[8] Aarthy, K. P., & Ragupathy, U. S. (2012, July 3). Detection of Lung Nodule Using Multiscale Wavelets and Support Vector Machine. *International Journal of Soft Computing and Engineering Vol, 2*(3).

[9] Zhao, B., Gamsu, G., Ginsberg, M., Jiang, L., & Schwartz, L (2003). Automatic detection of small lung nodules on CT utilizing a local density maximum algorithm. *Journal of Applied Clnical Medical Physics, Vol4(3).*

[10] Korfiatis, P., Skiadopoulos, S., Sakellaropoulos, P., Kalogeropoulou, C., & Costaridou, L. (2007). Combining 2D Wavelet Edge Highlighting and 3D Thresholding for Lung Segmentation in Thin-slice CT. *British Journal of Radiology*.

[11] Ye, X., Lin, X., Dehmeshki, J., Slabaugh, G., & Beddoe, G. (2008). Shape Based Computer-Aided Detection of Lung Nodules in Thoracic CT Images. *IEEE*.

[12] Sousa, J., Silva A., Paiva, A., & Nunes, R (2009). Methodology for automatic detection of lung nodules in computerized tomography images. *Computer Methods and Programs in Biomedicine.*

[13] Khaz, F., & Kavita V. (2010). Pulmonary Lung Segmentation in Computed Tomography using Fuzzy Logic. *European Journal of Scientific Research.*

[14] Anitha, S., & Sridhar, S. (2010). Segmentation of Lung Lobes and Nodules in CT Image. *Signal & Image Processing: An International Journal, Vol. 1(1).*

[15] Karthikeyan, C., Ramadoss, B. (August 2011). Segmentation Algorithm for CT Images Using Morphological Operation and Artificial Neural Networks. *International Journal of Computer Theory and Engineering, Vol.3(4).*

[16] Gomathi, M., & Thangaraj P. (2011). A Computer Aided Diagnosis System for Lung Cancer Detection using Machine Learning Technique. *European Journal of Scientific Research.*

[17] Tidke, S., Vrishali, A., & Chakkarwar (2012). Classification of Lung Tumor Using SVM. *International Journal of Computational Engineering Research.*

[18] Oh, K. s., Kaneko, K., & Makinouchi, A. (2000). Image Classification and Retrieval based on Wavelet-SOM. *IEEE.*

[19] Zhi, Y., & Ming, G. (2005). A SOM-Wavelet Networks for Face Identification. *IEEE.*

[20] Shi, S., Zhang, L., & Wei, L. (2012). Condition Monitoring and Fault Diagnosis of Rolling Element Bearings Based on Wavelet Energy Entropy and SOM. *IEEE.*

[21] Kang, P., & Birthwhistle, D. (1998). Analysis of Vibration Signals for Condition Monitoring of Power Switching Equipment using Wavelet Transform. *IEEE.*

[22] Tao, X., Wang, Z., Ma, J., & Fan, H. (2012). Study on Fault Detection using Wavelet Packet and SOM Neural Network. *IEEE.*

[23] Yang , C., Yuan, Y., & Si, J. (2010). High Performance Spike Detection and Sorting using Neural Waveform Phase Information and SOM Clustering. *IEEE.*

[24] Seling, A., Turunen, J., & Tanttu, J. (2007). Wavelets in Recognition of Bird Sounds. *EURASIP Journal on Advances in Signal Processing*.

[25] Hao, Y., Campana, B., & Keogh, E. (2012). Monitoring and Mining Insect Sounds in Visual Space. *SDM.*

[26] Nourani, V., Parhizkar, M., Baghanam, T., & Sharghi, E. (2010). Wavelet-based Feature Extraction of Rainfall-runoff Process via Self-Organizing Maps. *Latest Advances in Information Science and Applications*.

[27] Janahira, T., & Win, K. (2011). SOM Based Segmentation Method to Identify Water Region in LANDSAT Images. *IJECCT, 2*.

[28] Semleer, L., Dettori, L., & Furst, J. (n.d.). Wavelet-based Texture Classification of Tissues in Computed Tomography.

[29] Padma, A., & Sukanesh, R. (2011). A Wavelet Based Automatic Segmentation of Brain Tumor in CT Images Using Optimal Statistical Texture Features. *International Journal of Image Processing, 5*(5).

[30] Tian, D., & Linan, F. (2010). MR Brain Image Segmentation Based on Wavelet Transform and SOM Neural Network. *IEEE*.

[31] Kaneko, M., Gotho, T., Iseri, F., Takeshita, K., Ohki, H., & Sueda, N. (2011). QRS Complex Analysis Using Wavelet Transform and Two Layered Self-Organizing Map. *Computing in Cardiology*.

[32] Lee, S., Abibullaev, B., Kang, W.-S., Yunhee, S., & An, J. (2010). Analysis of Attention Deficit Hyperactivity Disorder in EEG using Wavelet Transform and Self Organizing Maps. *ICROS*.

[33] Tawade, L., & Warpe, H. (2011, March). Detection of Epilepsy Disorder using Discrete Wavelet Transforms using MATLAB. *International Journal of Advanced Science and Technology, 28*.

[34] Vasantha, M., Bharathi, S., & Dhamodharan, R. (2010). Medical Image Feature, Extraction, Selection, And Classification. *International Journal on Engineering Science and Technology, 2*(6).

[35] Kharrat, A., Gasmi, K., Messoud, M., Benamrane , N., & Abid, M. (2011). Medical Image Classification Using an Optimal Feature Extraction Algorithm and a Supervised Classifier Technique. *International Journal of Software Science and Computational Intelligence*.

[36] Baranidharan, T., & Ghosh, D. (2012). A Two Dimensional Image Classification Neural Network for Medical Images. *European Journal of Scientific Research, 74*.

[37] Tadejko, P., & Rakowski, W. (2007). Hybrid wavelet-mathematical morphology feature extraction for heartbeat classification. *Eurocon27*.

[38] Dond, X., & Sun, G. (n.d.). Discrete Wavelet Transform Based Feature Extraction forTissue Classification Using Gene Expression Data.

[39] Malek, J., Sebri, A., Mabrouk, S., Torki, K., & Tourki, R. (2008). Automated Breast Cancer Diagnosis Based on GVF-Snake Segmentation, Wavelet Features Extraction and Fuzzy Classification. *Springer Science*.

[40] Samb, M., Camara, F., Ndiaye, S., Slimani, Y., & Esseghir, M. (2012, June). A Novel RFE-SVM-based Feature Selection Approach for Classification. *International Journal of Advance Science and Technology, 43*.

[41] Hejazi, M., & Singh, Y. (2012). Credit Data Fraud Detection using Kernel Methods with Support Vector Machine. *Journal of Advanced Computer Science and Technology Research*, 35-49.

[42] Okwuashi, O., McConchie, J., Nwilo, P., Isong, M., Eyoh, A., Nwanekezie, O., et al. (2012, May). Predicting Future Land Use Change Using Support Vector Machine. *Journal of Sustainable Development, 5*.

[43] Venkatesan, P., & Devi, G. (2012, January). Proximal Support Vector Machine for Disease Classification. *International Journal of Science and Tehnology, 2*.

[44] Mandle, A., Jain, P., & Shrivastava, K. (2012, February). Protein Structure Recognition Using Support Vector Machine. *International Journal on Soft Computing, 3*.

[45] Pan, Y., Shen, P., & Shen, L. (2012, April). Speech Emotion Recognition Using Support Vector Machine. *International Journal of Smart Home, 6*.

[46] decision support system. *webopedia.com.* Retrieved October 12, 2012 from HYPERLINK "http://www.webopedia.com/TERM/D/decision_support_system.html"

[47] Noble, W. (2006). What is a Support Vector Machine Primer. *Nature Biotechnology, 24*.

[48] Chang, C., & Lin, C. (2012). A Library for Support Vector Machines.

[49] Fletcher, T. (2009). Support Vector Machines Explained. www.cs.ucl.ac.uk/staff/T.Fletcher/.

[50] Rhody, H., & Carlson, C. (2005). Basic Morphological Image Processing.

[51] *yourdictionary.com.* Retrieved October 12, 2012, from http://www.yourdictionary.com/segmentation

[52] Armato , S., McLennan, G., Bidaut , L., McNitt-Gray, M., Meyer, C., Reeves, A., et al. (2011, February). The Lung Image Database Consortium (LIDC) and Image Database

and Resource Initiative (IDRI): A Completed Reference Database of Lung Nodules on CT Scans. *Medical Physics, 38*.

[53] CT Scan. *medicinenet.com.*Retrieved October 12, 2012, from http://www.medicinenet.com/cat_scan/meridian-id_city.htm

X. Appendix

```
<?xml version="1.0" encoding="UTF-8"?>

<?import java.lang.*?>
<?import java.net.*?>
<?import java.util.*?>
<?import javafx.collections.*?>
<?import javafx.geometry.*?>
<?import javafx.scene.*?>
<?import javafx.scene.control.*?>
<?import javafx.scene.image.*?>
<?import javafx.scene.layout.*?>
<?import javafx.scene.paint.*?>
<?import javafx.scene.shape.*?>
<?import javafx.scene.text.*?>

<AnchorPane id="AnchorPane" maxHeight="-Infinity"
maxWidth="-Infinity" minHeight="-Infinity" minWidth="-Infinity"
prefHeight="520.0" prefWidth="1000.0" styleClass="theme"
xmlns:fx="http://javafx.com/fxml"
fx:controller="lungnoduledc.MainController">
  <children>
    <MenuBar maxHeight="-Infinity" minHeight="-Infinity"
prefHeight="26.0" prefWidth="900.0"
AnchorPane.bottomAnchor="494.0"
AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0"
AnchorPane.topAnchor="0.0">
      <menus>
        <Menu mnemonicParsing="false" text="File">
          <items>
            <MenuItem mnemonicParsing="false"
onAction="#tutorial" text="Tutorial" />
            <MenuItem mnemonicParsing="false" onAction="#about"
text="About" />
          </items>
        </Menu>
      </menus>
    </MenuBar>
    <SplitPane dividerPositions="0.23947895791583165"
focusTraversable="true" prefHeight="495.0" prefWidth="900.0"
AnchorPane.bottomAnchor="1.0" AnchorPane.leftAnchor="-1.0"
AnchorPane.rightAnchor="1.0" AnchorPane.topAnchor="24.0">
      <items>
        <AnchorPane maxWidth="-1.0" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="373.0" prefWidth="230.0">
          <children>
            <TabPane maxHeight="-Infinity" maxWidth="-Infinity"
minHeight="-Infinity" minWidth="-Infinity" prefHeight="373.0"
prefWidth="139.0" tabClosingPolicy="UNAVAILABLE"
tabMinWidth="0.0" AnchorPane.bottomAnchor="0.0"
AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0"
AnchorPane.topAnchor="0.0">
              <stylesheets>
                <URL
value="file:/E:/academics/4th_yr_2nd_sem/eclipse_javafx_worksp
ace/LungNoduleDCT/src/lungnoduledc/design.css" />
              </stylesheets>
              <tabs>
                <Tab fx:id="tab_input" text="Input">
                  <content>
                    <ScrollPane id="scroll_files" fx:id="scroll_input"
prefHeight="462.0" prefWidth="170.0" styleClass="content">
                      <content>
                        <AnchorPane id="AnchorPane"
prefHeight="548.0" prefWidth="206.0">
                          <children>
                            <HBox id="HBox" alignment="TOP_LEFT"
layoutX="28.0" layoutY="43.0" spacing="5.0">
                              <children>
                                <Label text="CT Scan Images:"
textFill="#333100" HBox.hgrow="ALWAYS" />
                                <Button fx:id="button_import"
mnemonicParsing="false" onAction="#importCTimages"
styleClass="buttons" text="Import">
                                  <stylesheets>
                                    <URL
value="file:/E:/academics/4th_yr_2nd_sem/eclipse_javafx_worksp
ace/LungNoduleDCT/src/lungnoduledc/design.css" />
                                  </stylesheets>
                                  <tooltip>
                                    <Tooltip text="If directory contains an xml
file,&#10;it will be used as the default &#10;annotation for the
dicom images" wrapText="false" />
                                  </tooltip>
                                </Button>
                              </children>
                            </HBox>
                            <Label fx:id="message_import" layoutX="39.0"
layoutY="66.0" prefWidth="161.0" styleClass="message_success"
text="" textFill="#310099">
                              <stylesheets>
                                <URL
value="file:/E:/academics/4th_yr_2nd_sem/eclipse_javafx_worksp
ace/LungNoduleDCT/src/lungnoduledc/design.css" />
                              </stylesheets>
                            </Label>
                            <Group id="Group" layoutX="19.0"
layoutY="115.0" scaleX="1.0" scaleY="1.0">
                              <children>
                                <Rectangle arcHeight="30.0"
arcWidth="30.0" height="78.0" layoutX="0.0" layoutY="-8.0"
opacity="1.0" stroke="WHITE" strokeLineCap="ROUND"
strokeLineJoin="ROUND" strokeType="INSIDE" width="187.0">
                                  <fill>
                                    <Color blue="0.800" green="1.000"
red="0.960" fx:id="x1" />
                                  </fill>
                                </Rectangle>
                                <Label layoutX="12.0" layoutY="4.0"
text="Type:" />
                                <ChoiceBox fx:id="choicebox_wavelet"
layoutX="51.0" layoutY="4.0" prefWidth="122.0">
                                  <items>
                                    <FXCollections
fx:factory="observableArrayList">
                                      <String fx:value="Item 1" />
                                      <String fx:value="Item 2" />
                                      <String fx:value="Item 3" />
                                    </FXCollections>
                                  </items>
                                  <stylesheets>
                                    <URL
value="file:/E:/academics/4th_yr_2nd_sem/eclipse_javafx_worksp
ace/LungNoduleDCT/src/lungnoduledc/design.css" />
                                  </stylesheets>
                                </ChoiceBox>
                                <Label layoutX="12.0" layoutY="33.0"
text="Scale:" />
                                <ChoiceBox fx:id="choicebox_scale"
layoutX="61.0" layoutY="32.0">
                                  <items>
                                    <FXCollections
fx:factory="observableArrayList">
                                      <String fx:value="Item 1" />
                                      <String fx:value="Item 2" />
                                      <String fx:value="Item 3" />
                              </FXCollections>
                            </items>
                          </ChoiceBox>
                        </children>
                      </Group>
                      <Label layoutX="76.0" layoutY="92.0"
text="Wavelets">
                        <font>
                          <Font name="Verdana Bold" size="12.0"
fx:id="x2" />
                        </font>
                        <textFill>
                          <Color blue="0.000" green="0.192"
red="0.400" fx:id="x3" />
                        </textFill>
                      </Label>
                      <Group id="Group" layoutX="19.0"
layoutY="263.0" scaleX="1.0" scaleY="1.0" />
                      <Rectangle arcHeight="30.0" arcWidth="30.0"
fill="#fffdcc" height="310.0" layoutX="18.0" layoutY="215.0"
stroke="WHITE" strokeType="INSIDE" width="188.0" />
                      <Group id="Group" layoutX="25.0"
layoutY="210.0" scaleX="1.0" scaleY="1.0">
                        <children>
                          <Label font="$x2" layoutX="48.0" layoutY="-
9.0" text="SVM" textFill="$x3" />
                          <Label layoutX="6.0" layoutY="17.0"
text="Kernel:" />
                          <ChoiceBox fx:id="choicebox_kernel"
layoutX="55.0" layoutY="16.0" prefWidth="111.0">
                            <items>
                              <FXCollections
fx:factory="observableArrayList">
                                <String fx:value="Item 1" />
                                <String fx:value="Item 2" />
                                <String fx:value="Item 3" />
                              </FXCollections>
                            </items>
                          </ChoiceBox>
                        </children>
                      </Group>
                      <Group id="Group" layoutX="21.0"
layoutY="550.0">
                        <children>
                          <Button layoutX="124.0" layoutY="7.0"
mnemonicParsing="false" onAction="#startProcess"
prefHeight="25.9998779296875" styleClass="buttons"
text="Start">
                            <stylesheets>
                              <URL
value="file:/E:/academics/4th_yr_2nd_sem/eclipse_javafx_worksp
ace/LungNoduleDCT/src/lungnoduledc/design.css" />
                            </stylesheets>
                          </Button>
                          <RadioButton fx:id="radio_button_train"
layoutX="-1.0" layoutY="0.0" mnemonicParsing="false"
text="Train" />
                          <RadioButton fx:id="radio_button_classify"
layoutX="0.0" layoutY="18.0" mnemonicParsing="false"
selected="true" text="Classify" />
                        </children>
                      </Group>
                      <Group id="Group" layoutX="16.0"
layoutY="200.0">
                        <children>
                          <Label layoutX="93.0" layoutY="0.0"
text="(for training)" />
                          <Separator layoutX="10.0" layoutY="112.0"
prefWidth="170.0" />
                          <Label layoutX="14.0" layoutY="115.0"
text="*For data which are already extracted:">
                            <font>
                              <Font name="System Italic" size="10.0"
fx:id="x9" />
                            </font>
                            <textFill>
                              <Color blue="0.400" green="0.063"
red="0.000" fx:id="x10" />
                            </textFill>
                          </Label>
                          <Label layoutX="16.0" layoutY="135.0"
text="Data:" />
                          <TextField fx:id="svm_import_data"
layoutX="50.0" layoutY="132.0" onAction="#svmImportData"
prefWidth="133.0" promptText="click then enter" text="">
                            <tooltip>
                              <Tooltip text="not required" />
                            </tooltip>
                          </TextField>
                          <Label font="$x9" layoutX="14.0"
layoutY="188.0" text="*For model which is already available"
textFill="$x10" />
                          <Button id="train" layoutX="127.0"
layoutY="157.0" mnemonicParsing="false"
onAction="#svmImportTrain" styleClass="buttons" text="train" />
                          <Label layoutX="16.0" layoutY="210.0"
text="Model:" />
                          <Separator layoutX="0.0" layoutY="243.0"
minHeight="9.1552734375E-5" prefHeight="9.1552734375E-5"
prefWidth="200.0" />
                          <Label font="$x9" layoutX="19.0"
layoutY="244.0" text="*For testing precomputed features"
textFill="#002966" />
                          <TextField id="svm_imort_data"
fx:id="svm_import_model" layoutX="54.0" layoutY="209.0"
onAction="#svmImportModel" prefWidth="126.0"
promptText="click then enter" text="">
                            <tooltip>
                              <Tooltip text="not required" />
                            </tooltip>
                          </TextField>
                          <Label layoutX="16.0" layoutY="57.0"
text="Gamma:" />
                          <Label layoutX="17.0" layoutY="77.0"
text="Degree:" />
                          <Label layoutX="115.0" layoutY="58.0"
minWidth="9.1552734375E-5" prefWidth="22.0" text="C:" />
                          <TextField fx:id="svm_gamma"
layoutX="65.0" layoutY="55.0" prefWidth="47.0" text="1.0" />
                          <TextField fx:id="svm_degree"
layoutX="65.0" layoutY="77.0" prefWidth="47.0" text="3" />
                          <TextField fx:id="svm_c" layoutX="127.0"
layoutY="54.0" prefWidth="45.0" text="1.0" />
                          <Label layoutX="14.0" layoutY="264.0"
text="Data:" />
                          <TextField fx:id="svm_test" layoutX="49.0"
layoutY="264.0" onAction="#svmTestFeatures"
prefWidth="125.0" promptText="click then enter" />
                          <Button id="svm_test_button"
layoutX="132.0" layoutY="288.0" mnemonicParsing="false"
onAction="#startTesting" styleClass="buttons" text="test" />
                        </children>
                      </Group>
                    </children>
                    <stylesheets>
                      <URL
value="file:/E:/academics/4th_yr_2nd_sem/eclipse_javafx_worksp
ace/LungNoduleDCT/src/lungnoduledc/design.css" />
                    </stylesheets>
```

```
            </AnchorPane>
           </content>
           <stylesheets>
            <URL
value="file:/E:/academics/4th_yr_2nd_sem/eclipse_javafx_worksp
ace/LungNoduleDCT/src/lungnoduledc/design.css" />
           </stylesheets>
          </ScrollPane>
         </content>
        </Tab>
        <Tab text="Files">
         <content>
          <AnchorPane fx:id="tab_files" prefHeight="200.0"
prefWidth="200.0" styleClass="content" />
         </content>
        </Tab>
        <Tab text="Export">
         <content>
          <AnchorPane id="Content" minHeight="0.0"
minWidth="0.0" prefHeight="180.0" prefWidth="200.0"
styleClass="content">
           <children>
            <Label font="$x2" layoutX="24.0" layoutY="14.0"
text="Additional comments">
             <textFill>
              <Color blue="0.000" green="0.286" red="0.400"
fx:id="x11" />
             </textFill>
            </Label>
            <TextArea fx:id="popup_comments"
blendMode="HARD_LIGHT" layoutX="14.0" layoutY="38.0"
prefHeight="287.0" prefWidth="200.0" promptText="optional"
wrapText="true" />
            <Button id="exportStart" layoutX="165.0"
layoutY="349.0" mnemonicParsing="false"
onAction="#exportStart" styleClass="buttons" text="Export" />
           </children>
          </AnchorPane>
         </content>
        </Tab>
       </tabs>
      </TabPane>
     </children>
     <stylesheets>
      <URL
value="file:/E:/academics/4th_yr_2nd_sem/eclipse_javafx_worksp
ace/LungNoduleDCT/src/lungnoduledc/design.css" />
     </stylesheets>
    </AnchorPane>
    <AnchorPane blendMode="SRC_OVER" cache="false"
disable="false" minHeight="0.0" minWidth="0.0"
mouseTransparent="false" prefHeight="493.0" prefWidth="720.0"
snapToPixel="true">
     <children>
      <TabPane prefHeight="493.0" prefWidth="774.0"
tabClosingPolicy="UNAVAILABLE"
AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
       <tabs>
        <Tab text="Original Image">
         <content>
          <SplitPane dividerPositions="0.6870026525198939"
focusTraversable="true" prefHeight="160.0" prefWidth="200.0">
           <items>
            <AnchorPane id="tab_original_image"
maxWidth="-Infinity" minHeight="0.0" minWidth="-Infinity"
prefHeight="460.0" prefWidth="515.0">
             <children>
              <ScrollPane prefHeight="460.0"
prefWidth="500.0" styleClass="content"
AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
               <content>
                <AnchorPane id="Content"
fx:id="tab_original_image" minHeight="-Infinity" minWidth="-
Infinity" prefHeight="512.0" prefWidth="512.0">
                 <children>
                  <TextField layoutX="-100.0"
layoutY="420.0" prefWidth="40.0" />
                 </children>
                </AnchorPane>
               </content>
              </ScrollPane>
             </children>
            </AnchorPane>
            <AnchorPane fx:id="details_dicom"
minHeight="0.0" minWidth="0.0" prefHeight="460.0"
prefWidth="245.0">
             <children>
              <SplitPane
dividerPositions="0.9934497816593887" focusTraversable="true"
orientation="VERTICAL" prefHeight="460.0" prefWidth="234.0"
AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
               <items>
                <AnchorPane minHeight="0.0"
minWidth="0.0" prefHeight="100.0" prefWidth="160.0">
                 <children>
                  <ScrollPane prefHeight="226.0"
prefWidth="232.0" styleClass="content"
AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
                   <content>
                    <AnchorPane id="Content" maxWidth="-
Infinity" minHeight="0.0" minWidth="0.0" prefHeight="278.0"
prefWidth="430.0">
                     <children>
                      <Label font="$x2" layoutX="14.0"
layoutY="25.0" maxWidth="1.7976931348623157E308"
text="Image Information:">
                       <textFill>
                        <Color blue="0.000" green="0.096"
red="0.400" fx:id="x4" />
                       </textFill>
                      </Label>
                      <Label layoutX="16.0"
layoutY="150.0" text="SOP Instance UID:" textFill="$x4">
                       <font>
                        <Font name="Verdana" size="12.0"
fx:id="x5" />
                       </font>
                      </Label>
                      <Label id="info_refSOP_classUID"
fx:id="info_refSOP_instanceUID" layoutX="34.0"
layoutY="165.0" maxWidth="1.7976931348623157E308"
text="no image loaded">
                       <font>
                        <Font name="System Italic"
size="12.0" fx:id="x6" />
                       </font>
                      </Label>
                      <Label font="$x5" layoutX="15.0"
layoutY="82.0" text="Study Instance UID:">
                       <textFill>
                        <Color blue="0.000" green="0.094"
red="0.400" fx:id="x7" />
                       </textFill>
                      </Label>
                      <Label
fx:id="info_study_instanceUID" font="$x6" layoutX="33.0"
```

```
layoutY="97.0" maxWidth="1.7976931348623157E308" text="no
image loaded" />
                              <Label font="$x5" layoutX="15.0"
layoutY="116.0" text="Series Instance UID:" textFill="$x7" />
                              <Label
fx:id="info_series_instanceUID" font="$x6" layoutX="32.0"
layoutY="131.0" maxWidth="1.7976931348623157E308"
prefWidth="-1.0" text="no image loaded" />
                              <Label font="$x5" layoutX="16.0"
layoutY="184.0" text="SOP Class UID:" textFill="$x7" />
                              <Label fx:id="info_studyID"
font="$x6" layoutX="34.0" layoutY="199.0"
maxWidth="1.7976931348623157E308" text="no image loaded"
/>
                              <Label font="$x5" layoutX="14.0"
layoutY="49.0" text="Patient ID:" textFill="$x7" />
                              <Label fx:id="info_patientID"
font="$x6" layoutX="34.0" layoutY="64.0"
maxHeight="1.7976931348623157E308" text="no image loaded"
/>
                              <Label font="$x5" layoutX="16.0"
layoutY="220.0" text="Pixels (in mm):" textFill="$x7" />
                              <Label id="info_patientID"
fx:id="info_dimension" font="$x6" layoutX="36.0"
layoutY="235.0" maxHeight="1.7976931348623157E308"
text="no image loaded" />
                              <HBox fx:id="progress_box"
layoutX="0.0" layoutY="416.0" prefHeight="26.0"
prefWidth="223.0" />
                             </children>
                            </AnchorPane>
                           </content>
                          </ScrollPane>
                         </children>
                        </AnchorPane>
                        <AnchorPane minHeight="0.0"
minWidth="0.0" prefHeight="100.0" prefWidth="160.0" />
                       </items>
                      </SplitPane>
                     </children>
                    </AnchorPane>
                   </items>
                  </SplitPane>
                 </content>
                </Tab>
                <Tab text="Segmented Image">
                 <content>
                  <SplitPane dividerPositions="0.696236559139785"
focusTraversable="true" prefHeight="160.0" prefWidth="200.0">
                   <items>
                    <AnchorPane
maxHeight="1.7976931348623157E308"
maxWidth="1.7976931348623157E308" minHeight="-Infinity"
minWidth="-Infinity" prefHeight="460.0" prefWidth="515.0">
                     <children>
                      <ScrollPane prefHeight="460.0"
prefWidth="500.0" AnchorPane.bottomAnchor="0.0"
AnchorPane.leftAnchor="0.0" AnchorPane.rightAnchor="0.0"
AnchorPane.topAnchor="0.0">
                       <content>
                        <AnchorPane id="Content"
fx:id="tab_image_morphed" minHeight="-Infinity" minWidth="-
Infinity" prefHeight="512.0" prefWidth="512.0"
styleClass="content" />
                       </content>
                      </ScrollPane>
                     </children>
                    </AnchorPane>
                    <AnchorPane minHeight="0.0" minWidth="-
Infinity" prefHeight="460.0" prefWidth="223.0"
styleClass="content">
                     <children>
                      <Label font="$x2" layoutX="14.0"
layoutY="21.0" text="Processes:" textFill="$x4" />
                      <Label font="$x5" layoutX="13.0"
layoutY="177.0" text="4. Dilation:" textFill="$x7" />
                      <Label fx:id="info_dilation" font="$x6"
layoutX="28.0" layoutY="194.0" text="No image loaded" />
                      <Label font="$x5" layoutX="13.0"
layoutY="216.0" text="5. Width-height ratio elimination"
textFill="$x7" />
                      <Label font="$x5" layoutX="12.0"
layoutY="119.0" text="3. Wavelet lung mask" textFill="$x7" />
                      <Label fx:id="info_region_filling" font="$x6"
layoutX="28.0" layoutY="155.0" text="No image loaded" />
                      <Label font="$x5" layoutX="12.0"
layoutY="85.0" text="2. Erosion" textFill="$x7" />
                      <Label fx:id="info_erosion" font="$x6"
layoutX="28.0" layoutY="98.0" text="No image loaded" />
                      <Label font="$x5" layoutX="13.0"
layoutY="51.0" text="1. Diffusion:" textFill="$x7" />
                      <Label id="info_dilation"
fx:id="info_anti_geometric_diffusion" font="$x6" layoutX="28.0"
layoutY="64.0" text="No image loaded" />
                      <Label fx:id="info_width_height_ratio"
font="$x6" layoutX="28.0" layoutY="234.0" text="Not yet done. "
/>
                      <Label font="$x5" layoutX="28.0"
layoutY="137.0" text="segmentation and region filling"
textFill="$x7" />
                     </children>
                    </AnchorPane>
                   </items>
                  </SplitPane>
                 </content>
                </Tab>
                <Tab text="Color Mapped Image">
                 <content>
                  <AnchorPane id="Content" minHeight="0.0"
minWidth="0.0" prefHeight="180.0" prefWidth="200.0"
styleClass="content">
                   <children>
                    <SplitPane
dividerPositions="0.6936339522546419" focusTraversable="true"
prefHeight="462.0" prefWidth="757.0" styleClass="content"
AnchorPane.bottomAnchor="0.0" AnchorPane.leftAnchor="0.0"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
                     <items>
                      <AnchorPane minHeight="-Infinity"
minWidth="-Infinity" prefHeight="478.0" prefWidth="520.0">
                       <children>
                        <ScrollPane id="tab_image_color_mapped"
prefHeight="471.0" prefWidth="520.0" styleClass="content"
AnchorPane.bottomAnchor="7.0" AnchorPane.leftAnchor="0.0"
AnchorPane.rightAnchor="0.0" AnchorPane.topAnchor="0.0">
                         <content>
                          <AnchorPane id="Content"
fx:id="tab_image_segmented" minHeight="0.0" minWidth="0.0"
prefHeight="493.0" prefWidth="520.0" />
                         </content>
                        </ScrollPane>
                       </children>
                      </AnchorPane>
                      <AnchorPane minHeight="0.0" minWidth="0.0"
prefHeight="160.0" prefWidth="100.0">
                       <children>
                        <Label font="$x2" layoutX="14.0"
layoutY="14.0" text="Details" textFill="$x11" />
```

```
                              <Label layoutX="13.0" layoutY="34.0"
text="Cancerous:" textFill="$x11">
                                <font>
                                  <Font name="System Bold" size="12.0"
fx:id="x8" />
                                </font>
                              </Label>
                              <TextArea fx:id="text_cancerous_colors"
blendMode="HARD_LIGHT" layoutX="12.0" layoutY="60.0"
prefWidth="200.0" text="None" wrapText="true" />
                              <Label font="$x8" layoutX="12.0"
layoutY="238.0" text="Non-cancerous:" textFill="$x11" />
                              <TextArea fx:id="text_noncancerous_colors"
blendMode="HARD_LIGHT" layoutX="12.0" layoutY="264.0"
prefWidth="200.0" text="None" wrapText="true" />
                            </children>
                          </AnchorPane>
                        </items>
                      </SplitPane>
                    </children>
                  </AnchorPane>
                </content>
              </Tab>
              <Tab text="Results">
                <content>
                  <AnchorPane id="Content" minHeight="0.0"
minWidth="0.0" prefHeight="180.0" prefWidth="200.0"
styleClass=", content">
                    <children>
                      <ScrollPane id="ScrollPane" layoutX="14.0"
layoutY="1.0" prefViewportHeight="462.0"
prefViewportWidth="731.0" styleClass="content">
                        <content>
                          <TableView fx:id="results_table"
prefHeight="449.0" prefWidth="2000.0" styleClass="content">
                            <columns>
                              <TableColumn prefWidth="75.0" text="File
Name" fx:id="table_file_name" />
                              <TableColumn prefWidth="75.0"
text="Patient ID" fx:id="table_patient_id" />
                              <TableColumn maxWidth="5000.0"
minWidth="10.0" prefWidth="157.0" text="Series Instance UID"
fx:id="table_series_instance_uid" />
                              <TableColumn maxWidth="5000.0"
minWidth="10.0" prefWidth="169.0" text="Study Instance UID"
fx:id="table_study_instance_uid" />
                              <TableColumn maxWidth="5000.0"
minWidth="10.0" prefWidth="266.0" text="SOP Instance UID"
fx:id="table_sop_instance_uid" />
                              <TableColumn prefWidth="75.0"
text="Color" fx:id="table_color" />
                              <TableColumn maxWidth="5000.0"
minWidth="10.0" prefWidth="129.0" text="Pixel Coordinates"
fx:id="table_pixel_coordinates" />
                              <TableColumn prefWidth="75.0"
text="Mean" fx:id="table_mean" />
                              <TableColumn prefWidth="75.0" text="Sd"
fx:id="table_sd" />
                              <TableColumn prefWidth="75.0" text="Area"
fx:id="table_area" />
                              <TableColumn prefWidth="75.0"
text="Center" fx:id="table_center" />
                              <TableColumn prefWidth="75.0"
text="Height" fx:id="table_height" />
                              <TableColumn prefWidth="75.0"
text="Width" fx:id="table_width" />
                              <TableColumn prefWidth="150.0"
text="Width Height Ratio" fx:id="table_ratio" />
                              <TableColumn prefWidth="75.0"
text="Known Data">
                                <columns>
                                  <TableColumn minWidth="90.0"
prefWidth="90.0" text="Cancerous" fx:id="table_cancerous" />
                                </columns>
                              </TableColumn>
                              <TableColumn minWidth="350.0"
prefWidth="350.0" text="Predicted Data">
                                <columns>
                                  <TableColumn prefWidth="75.0"
text="Cancerous" fx:id="table_predicted_cancerous" />
                                  <TableColumn minWidth="190.0"
prefWidth="230.0" text="Proby of malignancy"
fx:id="table_probability_of_malignancy">
                                    <columns>
                                      <TableColumn prefWidth="60.0"
text="yes" fx:id="proby_5" />
                                      <TableColumn prefWidth="60.0"
text="no" fx:id="proby_0" />
                                    </columns>
                                  </TableColumn>
                                </columns>
                              </TableColumn>
                              <TableColumn minWidth="90.0"
prefWidth="90.0" text="Remarks" fx:id="table_prediction" />
                            </columns>
                          </TableView>
                        </content>
                      </ScrollPane>
                    </children>
                  </AnchorPane>
                </content>
              </Tab>
              <Tab text="Summary">
                <content>
                  <AnchorPane id="Content" minHeight="0.0"
minWidth="0.0" prefHeight="180.0" prefWidth="200.0"
styleClass="content">
                    <children>
                      <Label font="$x2" layoutX="14.0" layoutY="20.0"
text="Details" textFill="$x3" />
                      <Label layoutX="14.0" layoutY="41.0" text="Total
number of potential nodules:" textFill="$x11" />
                      <Label layoutX="14.0" layoutY="84.0"
text="Cancerous:" textFill="$x11" />
                      <Label layoutX="14.0" layoutY="106.0"
text="Accuracy of classification:" textFill="$x11" />
                      <Label fx:id="summary_accuracy_cancerous"
font="$x6" layoutX="152.0" layoutY="106.0" text="not yet
computed" />
                      <Label fx:id="summary_nodules" font="$x6"
layoutX="206.0" layoutY="41.0" text="not yet computed" />
                      <Label fx:id="summary_cancerous" font="$x6"
layoutX="86.0" layoutY="84.0" text="not yet computed" />
                      <Label layoutX="15.0" layoutY="62.0" text="Non-
cancerous:" textFill="$x11" />
                      <Label id="summary_nodules"
fx:id="summary_non_cancerous" font="$x6" layoutX="107.0"
layoutY="62.0" text="not yet computed" />
                      <TextArea fx:id="summary_textbox"
blendMode="HARD_LIGHT" layoutX="15.0" layoutY="143.0"
prefWidth="330.0" promptText="other comments" text="other
details:" wrapText="true" />
                    </children>
                  </AnchorPane>
                </content>
              </Tab>
            </tabs>
          </TabPane>
        </children>
        <padding>
```

```
          <Insets />
        </padding>
      </AnchorPane>
    </items>
    <stylesheets>
      <URL
value="file:/E:/academics/4th_yr_2nd_sem/eclipse_javafx_worksp
ace/LungNoduleDCT/src/lungnoduledc/design.css" />
    </stylesheets>
  </SplitPane>
 </children>
 <stylesheets>
   <URL value="@design.css" />
 </stylesheets>
</AnchorPane>


.theme{
      master-color: #ff8c00;
}
.menu_bar{
      -fx-background-insets: 0, 1;
      -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 3,
0.0 , 0 , 1 );
}
.buttons{
      -fx-background-color:
       linear-gradient(
          derive(master-color, 120%),
          derive(master-color, 90%)
       ),
       radial-gradient(
          center 50% -40%,
          radius 180%,
          derive(master-color, 95%) 55%,
          derive(master-color, 75%) 55%
       );
   -fx-background-insets: 0, 1;
   -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 3, 0.0 ,
0 , 1 );
      -fx-text-fill: derive(master-color, -60%);
      -fx-background-radius: 30px;
}
.buttons:hover{
      -fx-background-color:
       linear-gradient(
          derive(#805817, 110%),
          derive(#805817, 110%)
       ),
       radial-gradient(
          center -40% 50%,
          radius 170%,
          derive(#805817, 110%) 55%,
          derive(#805817, 110%) 55%
       );
   -fx-background-insets: 0, 1;
   -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 3, 0.0 ,
0 , 1 );
      -fx-text-fill: derive(#805817, -60%);
      -fx-background-radius: 30px;
}
.content{
      -fx-background-color:
       linear-gradient(
          derive(#827839, 60%),
          derive(#827839, 100%),
          derive(#827839, 60%)
       );
}
.text_style{
      -fx-effect: dropshadow( three-pass-box , rgba(0,0,0,0.6) , 3,
0.0 , 0 , 1 );
}
.message_success{
      -fx-font-style: italic;
      -fx-font-size: 10px;
}
.choice-box{
      -fx-background-color:
       linear-gradient(
          derive(#827839, 70%),
          derive(#827839, 120%),
          derive(#827839, 100%)
       );
      -fx-border-color: #C9BE62;
      -fx-border-radius: 5px;
}
.split-pane-divider {
      -fx-border-color: #827839;
      -fx-background-color: transparent;
}
.mytextarea {
      -fx-text-fill: black;
      -fx-background-color: #a0522d;
}
#RED{
      -fx-background-color: #ff0000;
}
#GREEN{
      -fx-background-color: #00ff00;
}
#BLUE{
      -fx-background-color: #0000ff;
}
#YELLOW{
      -fx-background-color: #ffff00;
}
#ORANGE{
      -fx-background-color: #ffa500;
}
#VIOLET{
      -fx-background-color: #ee82ee;
}
#LIGHT_PINK{
      -fx-background-color: #ffb6c1;
}
#TURQUIOSE{
      -fx-background-color: #00f5ff;
}
#INDIAN_RED{
      -fx-background-color: #b0171f;
}
#KHAKI{
      -fx-background-color: #fff68f;
}
#CYAN{
      -fx-background-color: #00ffff;
}
#SGI_CHARTREUSE{
      -fx-background-color: #71c671;
}
#BISQUE{
      -fx-background-color: #ff1cc4;
}
#SEA_GREEN{
      -fx-background-color: #2e8b57;
}
#VIOLET_RED{
      -fx-background-color: #d02090;
}
```

```
#GOLDEN_ROD{
	-fx-background-color: #daa520;
}
#SPRING_GREEN{
	-fx-background-color: #00ff7f;
}
#MINT{
	-fx-background-color: #bdfcc9;
}
#OLIVE_DRAB{
	-fx-background-color: #6b8e23;
}
#TAN{
	-fx-background-color: #d2b48c;
}
#ROSY_BROWN{
	-fx-background-color: #bc8f8f;
}
#FUSCHIA{
	-fx-background-color: #ff00ff;
}
#DEEP_PINK{
	-fx-background-color: #ff1493;
}
#SLATE_BLUE{
	-fx-background-color: #836fff;
}
#GREEN_YELLOW{
	-fx-background-color: #adff2f;
}
#SGI_TEAL{
	-fx-background-color: #388e8e;
}
#CRIMSON{
	-fx-background-color: #dc143c;
}
#LAVENDER_BLUSH{
	-fx-background-color: #eee0e5;
}
#RASP_BERRY{
	-fx-background-color: #872657;
}
#PURPLE{
	-fx-background-color: #9b30ff;
}
#ROYAL_BLUE{
	-fx-background-color: #4169e1;
}
#PALE_GREEN{
	-fx-background-color: #98fb98;
}
#LEMON_CHIFFON{
	-fx-background-color: #fffacd;
}
#GOLD{
	-fx-background-color: #ffd700;
}
#PAPAYA_WHIP{
	-fx-background-color: #ffefd5;
}
#BURLY_WOOD{
	-fx-background-color: #deb887;
}
#CARROT{
	-fx-background-color: #ed9121;
}
#TOMATO{
	-fx-background-color: #ff6347;
}
#MISTY_ROSE{
	-fx-background-color: #ffe4e1;
}
#MAROON{
	-fx-background-color: #800000;
}
#LIGHT_GRAY{
	-fx-background-color: #bababa;
}
#BROWN{
	-fx-background-color: #a52a2a;
}
#LIGHT_CORAL{
	-fx-background-color: #f08080;
}
#SIENNA{
	-fx-background-color: #a0522d;
}
#PEACH_PUFF{
	-fx-background-color: #ffdab9;
}
#PERU{
	-fx-background-color: #cd853f;
}
#DARK_KHAKI{
	-fx-background-color: #bdb76b;
}
#POWDER_BLUE{
	-fx-background-color: #b0e0e6;
}
#THISTLE{
	-fx-background-color: #d8bfd8;
}
#PLUM{
	-fx-background-color: #dda0dd;
}
#ORCHID{
	-fx-background-color: #da70d6;
}
#ORCHID2{
	-fx-background-color: #ee7ae9;
}
#ORCHID3{
	-fx-background-color: #cd69c9;
}
#ORCHID4{
	-fx-background-color: #8b4789;
}
#CADMIUM_ORANGE{
	-fx-background-color: #ff6103;
}
#BURNT_SIENNA{
	-fx-background-color: #8a360f;
}
```

```
/**
 * @author jhesed tacadena
 */

package lungnoduledc;

import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.scene.layout.AnchorPane;

import javafx.stage.Stage;
```

```
public class LungNoduleDC extends Application {

    @Override
    public void start(final Stage primaryStage) {
        try {
            AnchorPane page = (AnchorPane)
FXMLLoader.load(LungNoduleDC.class
                    .getResource("MainScreenView.fxml"));
            Scene scene = new Scene(page);
            primaryStage.setScene(scene);
            primaryStage.setTitle("Lung Nodule Detector and
Classifier Tool");
            primaryStage.show();
        } catch (Exception ex) {

    Logger.getLogger(LungNoduleDC.class.getName()).log(Leve
l.SEVERE,
                    null, ex);
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}


/**
 * @author jhesed tacadena
 */

package lungnoduledc;

import javafx.application.Preloader;
import javafx.scene.Scene;
import javafx.scene.control.ProgressBar;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class LNDCTPreloader extends Preloader {

    ProgressBar bar;
    Stage stage;

    private Scene createPreloaderScene() {
        bar = new ProgressBar();
        BorderPane p = new BorderPane();
        p.setCenter(bar);
        return new Scene(p, 300, 150);
    }

    @Override
    public void start(Stage stage) throws Exception {
        this.stage = stage;
        stage.setScene(createPreloaderScene());
        stage.show();
    }

    @Override
    public void
handleStateChangeNotification(StateChangeNotification scn) {
        if (scn.getType() ==
StateChangeNotification.Type.BEFORE_START) {
            stage.hide();
        }
    }

    @Override
    public void handleProgressNotification(ProgressNotification
pn) {
        bar.setProgress(pn.getProgress());
    }
}


/**
 * @author jhesed tacadena
 */

package lungnoduledc;

import featureextraction.Nodule;
import fileprocessing.DicomImage;
import fileprocessing.KnownData;
import fileprocessing.XmlParser;

import java.awt.Desktop;
import java.awt.image.BufferedImage;
import java.io.BufferedOutputStream;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.OutputStream;
import java.net.MalformedURLException;
import java.net.URL;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.ResourceBundle;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import javax.imageio.ImageIO;
import javax.xml.parsers.ParserConfigurationException;

import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.pdmodel.PDPage;
import org.apache.pdfbox.pdmodel.edit.PDPageContentStream;
import org.apache.pdfbox.pdmodel.font.PDSimpleFont;
import org.apache.pdfbox.pdmodel.font.PDType1Font;
import org.apache.pdfbox.pdmodel.graphics.xobject.PDJpeg;
import
org.apache.pdfbox.pdmodel.graphics.xobject.PDXObjectImage;
import org.xml.sax.SAXException;

import com.sun.image.codec.jpeg.JPEGCodec;
import com.sun.image.codec.jpeg.JPEGImageEncoder;

import diffusion.Diffusion;

import libsvm.svm_predict;
import libsvm.svm_train;
import morph.GrayMorphology;
import morph.OtsuBinarize;

import javafx.application.Platform;
import javafx.beans.value.ChangeListener;
import javafx.beans.value.ObservableValue;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.embed.swing.SwingFXUtils;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.geometry.Insets;
import javafx.scene.Node;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.ChoiceBox;
import javafx.scene.control.Dialogs;
import javafx.scene.control.Label;
import javafx.scene.control.MenuItem;
import javafx.scene.control.RadioButton;
import javafx.scene.control.SelectionMode;
import javafx.scene.control.Separator;
import javafx.scene.control.TableCell;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableRow;
import javafx.scene.control.TableView;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.control.ToggleGroup;
import javafx.scene.control.TreeItem;
import javafx.scene.control.TreeView;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.VBox;
import javafx.stage.DirectoryChooser;
import javafx.stage.FileChooser;
import javafx.stage.Modality;
import javafx.stage.Stage;
import javafx.util.Callback;
import segmentation.NodulesExtraction;
import util.ImageUtil;
import wavelet.Wavelet;

public class MainController implements Initializable {
	private final int QUADRATIC_SPLINE = 0;
	private final int DAUBECHIES = 2;
	private final int DAUBECHIES5 = 3;
	private final int SCALE2 = 0;
	private final int SCALE1 = 2;
	private final int SCALE3 = 3;
	private float mm_dimension;

	private File directory;
	private File svm_data_to_import = null;
	private File svm_model_to_import = null;
	private File svm_test_to_import = null;

	private ImageView iv_image_original = new ImageView();
	private ImageView iv_image_morphed = new ImageView();
	private ImageView iv_image_segmented = new ImageView();
	private ArrayList<int[][]> pixel_binarized = new
ArrayList<int[][]>();
	private ArrayList<BufferedImage> image_morphed = new
ArrayList<BufferedImage>();
	private ArrayList<BufferedImage> image_segmented = new
ArrayList<BufferedImage>();
	private ArrayList<int[]> image_pixels = new
ArrayList<int[]>();
	private ArrayList<DicomImage> dicom_image = new
ArrayList<DicomImage>();
	private ArrayList<Nodule> nodules = new
ArrayList<Nodule>();

	@FXML
	private MenuItem menu_export;
	@FXML
	private Label message_import;
	@FXML
	private AnchorPane tab_files;
	@FXML
	private AnchorPane tab_original_image;
	@FXML
	private AnchorPane tab_image_wavelet;
	@FXML
	private AnchorPane tab_image_morphed;
	@FXML
	private AnchorPane tab_image_segmented;
	@FXML
	private AnchorPane tab_image_color_mapped;
	@SuppressWarnings("rawtypes")
	@FXML
	private ChoiceBox choicebox_wavelet;
	@SuppressWarnings("rawtypes")
	@FXML
	private ChoiceBox choicebox_scale;
	@SuppressWarnings("rawtypes")
	@FXML
	private ChoiceBox choicebox_kernel;
	@FXML
	private TextField svm_import_data;
	@FXML
	private TextField svm_import_model;
	@FXML
	private TextField svm_test;
	@FXML
	private TextField svm_gamma;
	@FXML
	private TextField svm_degree;
	@FXML
	private TextField svm_c;
	@FXML
	private RadioButton radio_button_train;
	@FXML
	private RadioButton radio_button_classify;
	@FXML
	private Label info_wavelet;
	@FXML
	private Label info_scale;
	@FXML
	private Label info_refSOP_instanceUID;
	@FXML
	private Label info_series_instanceUID;
	@FXML
	private Label info_study_instanceUID;
	@FXML
	private Label info_studyID;
	@FXML
	private Label info_patientID;
	@FXML
	private Label info_dimension;
	@FXML
	private Label info_anti_geometric_diffusion;
	@FXML
	private Label info_dilation;
	@FXML
	private Label info_region_filling;
	@FXML
	private Label info_erosion;
	@FXML
	private Label info_width_height_ratio;
	@FXML
	private Label summary_accuracy_cancerous;
	@FXML
	private Label summary_nodules;
	@FXML
	private Label summary_cancerous;
	@FXML
	private Label summary_non_cancerous;
	@FXML
	private TextArea summary_textbox;
	@FXML
	private TextArea text_cancerous_colors;
	@FXML
	private TextArea text_noncancerous_colors;
	@FXML
	private TableView<Nodule> results_table;
	@FXML
	private TableColumn<Nodule, String> table_file_name;
	@FXML
	private TableColumn<Nodule, String> table_patient_id;
	@FXML
	private TableColumn<Nodule, String>
table_series_instance_uid;
	@FXML
	private TableColumn<Nodule, String>
table_study_instance_uid;
	@FXML
	private TableColumn<Nodule, String>
table_sop_instance_uid;
	@FXML
	private TableColumn<Nodule, String> table_nodule_id;
	@FXML
	private TableColumn<Nodule, String> table_color;
	@FXML
	private TableColumn<Nodule, String>
table_pixel_coordinates;
	@FXML
	private TableColumn<Nodule, Double> table_mean;
	@FXML
	private TableColumn<Nodule, Double> table_sd;
	@FXML
	private TableColumn<Nodule, String> table_area;
	@FXML
	private TableColumn<Nodule, String> table_center;
	@FXML
	private TableColumn<Nodule, String> table_height;
	@FXML
	private TableColumn<Nodule, String> table_width;
	@FXML
	private TableColumn<Nodule, String> table_ratio;
	@FXML
	private TableColumn<Nodule, String> table_cancerous;
	@FXML
	private TableColumn<Nodule, String> table_malignancy;
	@FXML
	private TableColumn<Nodule, String>
table_predicted_cancerous;
	@FXML
	private TableColumn<Nodule, String>
table_predicted_malignancy;
	@FXML
	private TableColumn<Nodule, Double> proby_0;

	@FXML
	private TableColumn<Nodule, Double> proby_5;

	@FXML
	private TableColumn<Nodule, String> table_prediction;
	@FXML
	private TextArea popup_comments;
	private final ObservableList<Nodule> data = FXCollections
			.observableArrayList();

	@SuppressWarnings("unused")
	private KnownData known_data = new KnownData();
	private ArrayList<ArrayList<KnownData>> known_data_list
= new ArrayList<ArrayList<KnownData>>();
	private final Node root_icon = new ImageView(new Image(
			ResourceLoader.load("folder2.png")));
	private final Image ct_pic = new
Image(ResourceLoader.load("pic.png"));

	private boolean are_images_processed = false;
	private String svm_operation = "_testing";

	private boolean is_imported = false;
	private boolean is_xml_in_folder = false;

	@SuppressWarnings("unused")
	private int known_malignant_count = 0;
	private int detected_malignant_count = 0;
	private ArrayList<BufferedImage> tutorial_images = new
ArrayList<BufferedImage>();
	private int tutorial_page = 0;
	private ImageView page = new ImageView();
	private boolean is_already_loaded = false;

	private void detect() {

		ExecutorService executor =
Executors.newFixedThreadPool(2);
		try {
			BufferedWriter out = new BufferedWriter(new
FileWriter(directory
					+ "\\data" + svm_operation + ".data", true));
			for (int i = 0; i < dicom_image.size(); i++) {
				Callable<int[]> diffuse = new
Diffusion(Integer.parseInt("5"),
						40, Float.parseFloat("0.20"),
image_pixels.get(i),
						dicom_image.get(i).getWidth(),
dicom_image.get(i)
								.getHeight());
				Future<int[]> future_diffused =
executor.submit(diffuse);

				Callable<GrayMorphology> gm = new
GrayMorphology(dicom_image
						.get(i).getWidth(),
dicom_image.get(i).getHeight(),
						mm_dimension * Float.parseFloat("3"),
0, -1,
						future_diffused.get());

				OtsuBinarize otsu = new OtsuBinarize();
				BufferedImage edge =
ImageUtil.arrayToBufferedImage(otsu

	.binarize(ImageUtil.getPixels(dicom_image.get(i)
								.getOriginalImage(),
dicom_image.get(i)
								.getWidth(),
dicom_image.get(i).getHeight()),

	dicom_image.get(i).getHeight(), dicom_image
										.get(i).getWidth(), -1),
dicom_image
						.get(i).getWidth(),
dicom_image.get(i).getHeight());
				Future<GrayMorphology> future_gm =
executor.submit(gm);

	image_morphed.add(ImageUtil.getImageFromArray(future_g
m.get()
```

```
                        .getPixels(),
dicom_image.get(i).getWidth(),
                        dicom_image.get(i).getHeight()));
                Callable<Wavelet> wavelet = new Wavelet(
                        getScale(choicebox_scale),
choicebox_wavelet

    .getSelectionModel().getSelectedIndex(), edge,
                        pixel_binarized.get(i));
                Future<Wavelet> future_wavelet =
executor.submit(wavelet);

                Callable<NodulesExtraction> segment = new
NodulesExtraction(
                        dicom_image.get(i),
future_wavelet.get()
                                .getWaveletNodule(),
future_diffused.get(),
                        dicom_image.get(i).getWidth(),
dicom_image.get(i)
                                .getHeight(), known_data_list,
directory, -1,
                        svm_operation);
                Future<NodulesExtraction> future_extraction =
executor
                        .submit(segment);

                image_morphed.set(i, future_extraction.get()
                        .getRuleBasedMorphImage());
                image_segmented

    .add(future_extraction.get().getSegmentedImage());

                for (int j = 0; j <
future_extraction.get().getNodules().size(); j++) {

    nodules.add(future_extraction.get().getNodules().get(j));
                    try {
                        if
(!future_extraction.get().getNodules().get(j)

    .nativeGetMalignancy().equals("0")) {
                            out.write("1"); // malignant
                            detected_malignant_count++;
                        } else {
                            out.write("0"); // benign
                        }
                        out.write(" 1:"
                                +
future_extraction.get().getNodules().get(j)
                                        .nativeGetMean());
                        out.write(" 2:"
                                +
future_extraction.get().getNodules().get(j)
                                        .nativeGetSd() +
"\n");
                    } catch (Exception e) {
                        System.out.println("Error writing
file.");
                    }
                }

                dicom_image.get(i).setCancerousColors(

    future_extraction.get().getColorsOfCancerousNodules());
                dicom_image.get(i).setNonCancerousColors(
                        future_extraction.get()

    .getColorsOfNonCancerousNodules());
            }
            out.close();
            executor.shutdown();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (ExecutionException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @SuppressWarnings("unchecked")
    @Override
    public void initialize(URL url, ResourceBundle rb) {

    choicebox_wavelet.setItems(FXCollections.observableArrayL
ist("Spline",
                new Separator(), "Daubechies",
"Daubechies5"));
        choicebox_wavelet.getSelectionModel().select(0);

    choicebox_scale.setItems(FXCollections.observableArrayList(
2,
                new Separator(), 1));
        choicebox_scale.getSelectionModel().select(0);

    choicebox_kernel.setItems(FXCollections.observableArrayLis
t("RBF",
                new Separator(), "Linear", "Polynomial"));
        choicebox_kernel.getSelectionModel().select(0);

        final ToggleGroup radio_group = new ToggleGroup();
        radio_button_train.setToggleGroup(radio_group);
        radio_button_classify.setToggleGroup(radio_group);
        radio_button_classify.setSelected(true);

        table_file_name
                .setCellValueFactory(new
PropertyValueFactory<Nodule, String>(
                        "file_name"));
        table_patient_id
                .setCellValueFactory(new
PropertyValueFactory<Nodule, String>(
                        "patient_id"));
        table_series_instance_uid
                .setCellValueFactory(new
PropertyValueFactory<Nodule, String>(
                        "series_instance_uid"));
        table_study_instance_uid
                .setCellValueFactory(new
PropertyValueFactory<Nodule, String>(
                        "study_instance_uid"));
        table_sop_instance_uid
                .setCellValueFactory(new
PropertyValueFactory<Nodule, String>(
                        "imageSOP_UID"));
        table_color
                .setCellValueFactory(new
PropertyValueFactory<Nodule, String>(
                        "color"));

        table_pixel_coordinates
                .setCellValueFactory(new
PropertyValueFactory<Nodule, String>(
                        "pixel_coordinates"));
        table_mean
                .setCellValueFactory(new
PropertyValueFactory<Nodule, Double>(
                        "mean"));
```

```
		table_sd.setCellValueFactory(new
PropertyValueFactory<Nodule, Double>(
				"sd"));
		table_area
				.setCellValueFactory(new
PropertyValueFactory<Nodule, String>(
						"table_area"));
		table_center
				.setCellValueFactory(new
PropertyValueFactory<Nodule, String>(
						"table_center"));
		table_height
				.setCellValueFactory(new
PropertyValueFactory<Nodule, String>(
						"table_height"));
		table_width
				.setCellValueFactory(new
PropertyValueFactory<Nodule, String>(
						"table_width"));
		table_ratio
				.setCellValueFactory(new
PropertyValueFactory<Nodule, String>(
						"table_ratio"));
		table_cancerous
				.setCellValueFactory(new
PropertyValueFactory<Nodule, String>(
						"is_cancerous"));
		table_prediction
				.setCellValueFactory(new
PropertyValueFactory<Nodule, String>(
						"patient_prediction"));
		table_predicted_cancerous
				.setCellValueFactory(new
PropertyValueFactory<Nodule, String>(
						"prediction_is_cancerous"));
		table_prediction
				.setCellValueFactory(new
PropertyValueFactory<Nodule, String>(
						"prediction"));

		proby_0.setCellValueFactory(new
PropertyValueFactory<Nodule, Double>(
				"proby_0"));
		proby_5.setCellValueFactory(new
PropertyValueFactory<Nodule, Double>(
				"proby_5"));

		results_table.setEditable(true);
		results_table.setItems(data);
		try {

	tutorial_images.add(ImageIO.read(ResourceLoader.load("pag
e1.jpg")));

	tutorial_images.add(ImageIO.read(ResourceLoader.load("pag
e2.jpg")));

	tutorial_images.add(ImageIO.read(ResourceLoader.load("pag
e3.jpg")));

	tutorial_images.add(ImageIO.read(ResourceLoader.load("pag
e4.jpg")));

	tutorial_images.add(ImageIO.read(ResourceLoader.load("pag
e5.jpg")));

	tutorial_images.add(ImageIO.read(ResourceLoader.load("pag
e6.jpg")));

	tutorial_images.add(ImageIO.read(ResourceLoader.load("pag
e7.jpg")));

	tutorial_images.add(ImageIO.read(ResourceLoader.load("_1.j
pg")));

	tutorial_images.add(ImageIO.read(ResourceLoader.load("_2.j
pg")));

	tutorial_images.add(ImageIO.read(ResourceLoader.load("_3.j
pg")));

	tutorial_images.add(ImageIO.read(ResourceLoader.load("_4.j
pg")));

	tutorial_images.add(ImageIO.read(ResourceLoader.load("_5.j
pg")));

	tutorial_images.add(ImageIO.read(ResourceLoader.load("_6.j
pg")));

	tutorial_images.add(ImageIO.read(ResourceLoader.load("_7.j
pg")));

		} catch (Exception e) {
			System.out.println("not found");
		}
	}

	@SuppressWarnings("unused")
	@FXML
	protected void importCTimages(ActionEvent event)
			throws MalformedURLException {

		if (is_already_loaded) {
			final Stage myDialog = new Stage();
			Dialogs.showWarningDialog(myDialog,
					"All previous session will be erased.",
"Warning", "LNDCT");
			this.image_morphed = new
ArrayList<BufferedImage>();
			this.image_pixels = new ArrayList<int[]>();
			this.image_segmented = new
ArrayList<BufferedImage>();
			is_imported = false;
			is_xml_in_folder = false;
			known_malignant_count = 0;
			detected_malignant_count = 0;
			this.dicom_image = new ArrayList<DicomImage>();
			this.pixel_binarized = new ArrayList<int[][]>();
			this.nodules = new ArrayList<Nodule>();
		}

		DirectoryChooser fileChooser = new DirectoryChooser();
		FileChooser.ExtensionFilter extFilter = new
FileChooser.ExtensionFilter(
				"DICOM files (*.dcm)", "*.dcm");
		directory = fileChooser.showDialog(null);
		listFilesForFolder(directory);
		showFileTree(dicom_image, directory.toString());
		loadDicomImage();
		message_import.setText("See imported images in file
tab");
		is_imported = true;

		if (!is_xml_in_folder) {
			final Stage dialog_warning = new Stage();
			Dialogs.showWarningDialog(dialog_warning,
                    "There is no annotation xml file in the
chosen folder.",
                    "Warning", "LNDCT ");
        }
    }

    @FXML
    private void svmImportData(ActionEvent event) throws
MalformedURLException {
        FileChooser fileChooser = new FileChooser();
        FileChooser.ExtensionFilter extFilter = new
FileChooser.ExtensionFilter(
                "SVM data file", "*.data");
        fileChooser.getExtensionFilters().add(extFilter);
        svm_data_to_import =
fileChooser.showOpenDialog(null);

    svm_import_data.setText(svm_data_to_import.toString());
    }

    @FXML
    private void svmImportModel(ActionEvent event) throws
MalformedURLException {
        FileChooser fileChooser = new FileChooser();
        FileChooser.ExtensionFilter extFilter = new
FileChooser.ExtensionFilter(
                "SVM model file", "*.model");
        fileChooser.getExtensionFilters().add(extFilter);
        svm_model_to_import =
fileChooser.showOpenDialog(null);

    svm_import_model.setText(svm_model_to_import.toString())
;
    }

    @FXML
    private void svmTestFeatures(ActionEvent event)
            throws MalformedURLException {
        FileChooser fileChooser = new FileChooser();
        FileChooser.ExtensionFilter extFilter = new
FileChooser.ExtensionFilter(
                "SVM data file", "*.data");
        fileChooser.getExtensionFilters().add(extFilter);
        svm_test_to_import =
fileChooser.showOpenDialog(null);
        svm_test.setText(svm_test_to_import.toString());
    }

    @FXML
    private void svmImportTrain(ActionEvent event) {
        svm_train train;
        final Stage myDialog = new Stage();

        try {
            if
(choicebox_kernel.getSelectionModel().getSelectedIndex() == 0) {
                train = new
svm_train(svm_data_to_import.toString(), 2,

    Double.parseDouble(svm_gamma.getText()),
                        Integer.parseInt(svm_degree.getText()),
                        Double.parseDouble(svm_c.getText()));
// RBF
            } else if
(choicebox_kernel.getSelectionModel().getSelectedIndex() == 2) {
                train = new
svm_train(svm_data_to_import.toString(), 0,

    Double.parseDouble(svm_gamma.getText()),
                        Integer.parseInt(svm_degree.getText()),
                        Double.parseDouble(svm_c.getText()));
// linear
            } else {
                train = new
svm_train(svm_data_to_import.toString(), 1,

    Double.parseDouble(svm_gamma.getText()),
                        Integer.parseInt(svm_degree.getText()),
                        Double.parseDouble(svm_c.getText()));
// poly
            }
            train.run();
            Dialogs.showInformationDialog(myDialog, "SVM
Training completed.",
                    "Information", "LNDCT");
        } catch (Exception e) {

            Dialogs.showErrorDialog(myDialog, "No data was
loaded.", "Error",
                    "LNDCT");
        }
    }

    @FXML
    public void startTesting(ActionEvent e) {
        final Stage myDialog = new Stage();
        try {
            svm_predict predict = new svm_predict(
                    svm_test_to_import.toString(),
                    svm_model_to_import.toString());
            predict.run2();
            Dialogs.showInformationDialog(
                    myDialog,
                    "SVM Testing completed.\nPlease see the
prediction "
                            + "file in the same folder of the
data file loaded.\nAccuracy: "
                            + predict.getSeparateAccuracy(),
"Information",
                    "LNDCT");
        } catch (Exception ex) {
            Dialogs.showErrorDialog(
                    myDialog,
                    "Please check if a model and a prediction
file was loaded.",
                    "Error", "LNDCT");
        }
    }

    public void listFilesForFolder(final File folder) {
        for (final File fileEntry : folder.listFiles()) {
            if (fileEntry.isDirectory()) {
                listFilesForFolder(fileEntry);
            } else {
                if
(fileExtensionParser(fileEntry.getName()).equals("dcm")) {
                    dicom_image.add(new DicomImage(new
File(folder + "\\"
                            + fileEntry.getName()),
fileEntry.getName()));
                } else if
(fileExtensionParser(fileEntry.getName()).equals(
                        "model")) {
                    this.svm_model_to_import =
fileEntry.getAbsoluteFile();
                } else if
(fileExtensionParser(fileEntry.getName()).equals(
                        "xml")) {
                    try {
```

```
						XmlParser sp = new XmlParser((new
File(folder + "\\"
								+
fileEntry.getName())).toString());
						sp.startParsing(sp);

	known_data_list.add(sp.getKnownDataList());
						is_xml_in_folder = true;
					} catch (IOException e) {
						e.printStackTrace();
					} catch (SAXException e) {
						e.printStackTrace();
					} catch (ParserConfigurationException e) {
						e.printStackTrace();
					}
				}
			}
		}
	}

	@SuppressWarnings({ "rawtypes", "unchecked" })
	public void showFileTree(ArrayList<DicomImage>
dcm_files,
			final String parent_folder) {
		TreeItem<String> root_item = new
TreeItem<String>(parent_folder,
				root_icon);
		root_item.setExpanded(true);

		for (int i = 0; i < dcm_files.size(); i++) {
			TreeItem item = new
TreeItem(imageNameParser(dcm_files.get(i)
					.getDcmFile().toString()), new
ImageView(ct_pic));
			root_item.getChildren().add(item);
		}

		final TreeView<String> file_tree = new
TreeView<String>(root_item);

	file_tree.getSelectionModel().setSelectionMode(SelectionMod
e.SINGLE);
		file_tree.getSelectionModel().selectedItemProperty()
				.addListener(new ChangeListener() {

					@Override
					public void changed(ObservableValue
observable,
							Object oldValue, Object newValue)
{

						TreeItem<String> treeItem =
(TreeItem) newValue;

						int index_of_selected = 0;
						for (int i = 0; i < dicom_image.size();
i++) {
							if (dicom_image
									.get(i)
									.getDcmFile()
									.toString()
									.equals(parent_folder +
"\\"
											+
treeItem.getValue().toString())) {
								index_of_selected = i;
								break;
							}
						}
						setDicomImage(index_of_selected);
						if (are_images_processed) {

	setMorphImage(index_of_selected);

	setSegmentedImage(index_of_selected);
						}
					}
				});

		Platform.runLater(new Runnable() {
			@Override
			public void run() {
				tab_files.getChildren().add(file_tree);
			}
		});
	}

	private void setDicomImage(int index) {

	iv_image_original.setImage(SwingFXUtils.toFXImage(dicom
_image
				.get(index).getOriginalImage(), null));
		setOriginalImageInfo(dicom_image.get(index));
		text_cancerous_colors.setText(dicom_image.get(index)
				.getCancerousColors());

	text_noncancerous_colors.setText(dicom_image.get(index)
				.getNonCancerousColors());
	}

	private void setMorphImage(int index) {
		iv_image_morphed.setImage(SwingFXUtils.toFXImage(
				image_morphed.get(index), null));
	}

	private void setSegmentedImage(int index) {

	iv_image_segmented.setImage(SwingFXUtils.toFXImage(
				image_segmented.get(index), null));
	}

	private void setOriginalImageInfo(DicomImage
dicom_image) {

	info_refSOP_instanceUID.setText(dicom_image.getRefSOPC
lassUID());

	info_series_instanceUID.setText(dicom_image.getSeriesInsta
nceUID());

	info_study_instanceUID.setText(dicom_image.getStudyInstan
ceUID());
		info_studyID.setText(dicom_image.getStudyID());
		info_patientID.setText(dicom_image.getPatientID());
		mm_dimension =
Float.parseFloat(dicom_image.getDimension());
		info_dimension.setText(dicom_image.getDimension());
	}

	@SuppressWarnings({ "unused", "rawtypes" })
	private void setInfoWaveletMessage(ChoiceBox wavelet) {
		if (wavelet.getSelectionModel().getSelectedIndex() ==
QUADRATIC_SPLINE) {
			info_wavelet.setText("Spline");
		} else if (wavelet.getSelectionModel().getSelectedIndex()
== DAUBECHIES) {
			info_wavelet.setText("Daubechies");
		} else if (wavelet.getSelectionModel().getSelectedIndex()
== DAUBECHIES5) {
			info_wavelet.setText("Daubechies5");
```

```
            }
        }

        @FXML
        protected void startProcess() {

            if (is_imported) {
                if (radio_button_train.isSelected()) {
                    svm_operation = "_training";
                }
                detect();

                are_images_processed = true;

                if (radio_button_train.isSelected()) {

                    svm_train training_data;
                    if
(choicebox_kernel.getSelectionModel().getSelectedIndex() == 0) {
                        training_data = new svm_train(directory
                                + "\\data_training.data", 2,

        Double.parseDouble(svm_gamma.getText()),

        Integer.parseInt(svm_degree.getText()),

        Double.parseDouble(svm_c.getText())); // RBF
                    } else if (choicebox_kernel.getSelectionModel()
                            .getSelectedIndex() == 2) {
                        training_data = new svm_train(directory
                                + "\\data_training.data", 0,

        Double.parseDouble(svm_gamma.getText()),

        Integer.parseInt(svm_degree.getText()),

        Double.parseDouble(svm_c.getText())); // linear
                    } else {
                        training_data = new svm_train(directory
                                + "\\data_training.data", 1,

        Double.parseDouble(svm_gamma.getText()),

        Integer.parseInt(svm_degree.getText()),

        Double.parseDouble(svm_c.getText())); // poly
                    }

                    try {
                        training_data.run();
                        summary_nodules.setText(nodules.size() +
"");
                        summary_non_cancerous
                                .setText((nodules.size() -
detected_malignant_count)
                                        + "");

        summary_cancerous.setText(detected_malignant_count + "");

        summary_accuracy_cancerous.setText("NA");
                        for (int l = 0; l < nodules.size(); l++) {
                            nodules.get(l).setPrediction("NA");

        nodules.get(l).setPredictionMalignancy("NA");

        nodules.get(l).setIsPredictionCancerous("NA");
                            nodules.get(l).setProby0(-0.0);
                            nodules.get(l).setProby5(-0.0);

                            if (!is_xml_in_folder) {

        nodules.get(l).setIsCancerous("NA");
                            }
                            summary_textbox.setText("other
details: \n None.");
                            data.add(nodules.get(l));
                        }
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                } else if (radio_button_classify.isSelected()) {
                    svm_predict predict_data;
                    if (svm_model_to_import != null) {
                        predict_data = new svm_predict(directory
                                + "\\data_testing.data",
                                svm_model_to_import.toString(),
nodules);
                    } else {
                        predict_data = new svm_predict(directory
                                + "\\data_testing.data", directory
                                + "\\data_training.data.model",
nodules);
                    }
                    try {
                        predict_data.run();
                        nodules = predict_data.getNodules();

                        String summary = "other details\n";
                        boolean is_infected = false;
                        String patient =
nodules.get(0).nativeGetPatientId();

                        summary_nodules.setText(nodules.size() +
"");
                        summary_non_cancerous
                                .setText((nodules.size() -
detected_malignant_count)
                                        + "");

        summary_cancerous.setText(detected_malignant_count + "");

        summary_accuracy_cancerous.setText(predict_data
                                .getCancerousClassification());
                        for (int l = 0; l < nodules.size(); l++) {
                            if (nodules
                                    .get(l)
                                    .nativeGetCancerous()
                                    .equals(nodules.get(l)

        .nativeGetPredictedisCancerous())) {

        nodules.get(l).setPrediction("correct");
                            } else {

        nodules.get(l).setPrediction("wrong");
                            }
                            if (!is_xml_in_folder) {

        nodules.get(l).setIsCancerous("NA");
                                nodules.get(l).setPrediction("NA");
                            }
                            if
(nodules.get(l).nativeGetPredictedisCancerous()
                                    .equals("yes")) {
                                is_infected = true;
                            }
                            if (!nodules.get(l).nativeGetPatientId()
                                    .equals(patient)) {
                                if (is_infected) {
                                        summary += patient + " has
cancer\n";
                                    } else {
                                        summary += patient + "doesn't
have cancer\n";
                                    }
                                    patient =
nodules.get(l).nativeGetPatientId();
                                }

                                data.add(nodules.get(l));
                            }
                            if (is_infected) {
                                summary += patient + " has cancer\n";
                            } else {
                                summary += patient + " doesn't have
cancer\n";
                            }
                            summary_textbox.setText(summary);

    text_cancerous_colors.setText(dicom_image.get(0)
                                    .getCancerousColors());

    text_noncancerous_colors.setText(dicom_image.get(0)
                                    .getNonCancerousColors());
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                    }

                    Platform.runLater(new Runnable() {
                        @Override
                        public void run() {

    iv_image_morphed.setImage(SwingFXUtils.toFXImage(
                                    image_morphed.get(0), null));

                            if
(tab_image_morphed.getChildren().isEmpty()) {

    tab_image_morphed.getChildren().add(iv_image_morphed);
                            } else {
                                tab_image_morphed.getChildren()
                                        .remove(iv_image_morphed);

    tab_image_morphed.getChildren().add(iv_image_morphed);
                            }


    iv_image_segmented.setImage(SwingFXUtils.toFXImage(
                                    image_segmented.get(0), null));

                            if
(tab_image_segmented.getChildren().isEmpty()) {

    tab_image_segmented.getChildren().add(
                                        iv_image_segmented);
                            } else {

    tab_image_segmented.getChildren().remove(
                                        iv_image_segmented);

    tab_image_segmented.getChildren().add(
                                        iv_image_segmented);
                            }

                            info_erosion.setText("Done");
                            info_dilation.setText("Done");
                            info_width_height_ratio.setText("Done");

    info_anti_geometric_diffusion.setText("Done");
                            info_region_filling.setText("Done");
                        }
                    });
                    table_color
                            .setCellFactory(new
Callback<TableColumn<Nodule, String>, TableCell<Nodule,
String>>() {
                                public TableCell<Nodule, String> call(
                                        TableColumn<Nodule, String>
param) {
                                    return new TableCell<Nodule,
String>() {

                                        @Override
                                        public void updateItem(String
item,
                                                boolean empty) {
                                            super.updateItem(item,
empty);
                                            if (!isEmpty()) {

    @SuppressWarnings("rawtypes")
                                                TableRow row =
this.getTableRow();


    System.out.println("item: " + item);
                                                switch (item) {
                                                case "RED":

    row.setId("RED");
                                                    break;
                                                case "GREEN":

    row.setId("GREEN");
                                                    break;
                                                case "BLUE":

    row.setId("BLUE");
                                                    break;
                                                case "YELLOW":

    row.setId("YELLOW");
                                                    break;
                                                case "ORANGE":

    row.setId("ORANGE");
                                                    break;
                                                case "VIOLET":

    row.setId("VIOLET");
                                                    break;
                                                case "LIGHT PINK":

    row.setId("LIGHT_PINK");
                                                    break;
                                                case "TURQUIOSE":

    row.setId("TURQUIOSE");
                                                    break;
                                                case "INDIAN RED":

    row.setId("INDIAN_RED");
                                                    break;
                                                case "KHAKI":

    row.setId("KHAKI");
                                                    break;
```

```
case "CYAN":
    row.setId("CYAN");
    break;
case "SGI
CHARTREUSE":
    row.setId("SGI_CHARTREUSE");
    break;
case "BISQUE":
    row.setId("BISQUE");
    break;
case "SEA GREEN":
    row.setId("SEA_GREEN");
    break;
case "VIOLET RED":
    row.setId("VIOLET_RED");
    break;
case "GOLDEN
ROD":
    row.setId("GOLDEN_ROD");
    break;
case "SPRING
GREEN":
    row.setId("SPRING_GREEN");
    break;
case "MINT":
    row.setId("MINT");
    break;
case "OLIVE
DRAB":
    row.setId("OLIVE_DRAB");
    break;
case "TAN":
    row.setId("TAN");
    break;
case "ROSY
BROWN":
    row.setId("ROSY_BROWN");
    break;
case "FUSCHIA":
    row.setId("FUSCHIA");
    break;
case "DEEP PINK":
    row.setId("DEEP_PINK");
    break;
case "SLATE
BLUE":
    row.setId("SLATE_BLUE");
    break;
case "GREEN
YELLOW":
    row.setId("GREEN_YELLOQ");
    break;
case "SGI TEAL":
    row.setId("SGI_TEAL");
    break;
case "CRIMSON":
    row.setId("CRIMSON");
    break;
case "LAVENDER
BLUSH":
    row.setId("LAVENDER_BLUSH");
    break;
case "RASP
BERRY":
    row.setId("RASP_BERRY");
    break;
case "PURPLE":
    row.setId("PURPLE");
    break;
case "ROYAL
BLUE":
    row.setId("ROYAL_BLUE");
    break;
case "PALE
GREEN":
    row.setId("PALE_GREEN");
    break;
case "LEMON
CHIFFON":
    row.setId("LEMON_CHIFFON");
    break;
case "GOLD":
    row.setId("GOLD");
    break;
case "PAPAYA
WHIP":
    row.setId("PAPAYA_WHIP");
    break;
case "BURLY
WOOD":
    row.setId("BURLY_WOOD");
    break;
case "CARROT":
    row.setId("CARROT");
    break;
case "TOMATO":
    row.setId("TOMATO");
    break;
case "MISTY ROSE":
    row.setId("MISTY_ROSE");
    break;
case "MAROON":
    row.setId("MAROON");
    break;
case "LIGHT
GRAY":
    row.setId("LIGHT_GRAY");
    break;
case "BROWN":
	row.setId("BROWN");
								break;
							case "LIGHT
CORAL":

	row.setId("LIGHT_CORAL");
								break;
							case "SIENNA":

	row.setId("SIENNA");
								break;
							case "PEACH PUFF":

	row.setId("PEACH_PUFF");
								break;
							case "PERU":

	row.setId("PERU");
								break;
							case "DARK
KHAKI":

	row.setId("DARK_KHAKI");
								break;
							case "POWDER
BLUE":

	row.setId("POWDER_BLUE");
								break;
							case "THISTLE":

	row.setId("THISTLE");
								break;
							case "PLUM":

	row.setId("PLUM");
								break;
							case "ORCHID":

	row.setId("ORCHID");
								break;
							case "ORCHID2":

	row.setId("ORCHID2");
								break;
							case "ORCHID3":

	row.setId("ORCHID3");
								break;
							case "ORCHID4":

	row.setId("ORCHID4");
								break;
							case "CADMIUM
ORANGE":

	row.setId("CADMIUM_ORANGE");
								break;
							case "BURNT
SIENNA":

	row.setId("BURNT_SIENNA");
								break;
							}
							setText(item);
						}
					}
				};
			}
```

```
					});
			} else {
				final Stage dialog_error = new Stage();
				Dialogs.showErrorDialog(dialog_error,
						"You have not yet imported CT scan
images.", "Error",
						"LNDCT ");

			}
		}

	private void loadDicomImage() {

		OtsuBinarize otsu = new OtsuBinarize();

		for (int i = 0; i < dicom_image.size(); i++) {
			int pixels[][] = otsu.binarize(ImageUtil.getPixels(
					dicom_image.get(i).getOriginalImage(),
dicom_image.get(i)
							.getWidth(),
dicom_image.get(i).getHeight()),
					dicom_image.get(i).getHeight(),
dicom_image.get(i)
							.getWidth(), -1);
			pixel_binarized.add(pixels);
			image_pixels.add(ImageUtil.twoDtoOneD(pixels,
dicom_image.get(i)
					.getHeight(),
dicom_image.get(i).getWidth()));
		}

		is_already_loaded = true;
		Platform.runLater(new Runnable() {
			@Override
			public void run() {

	iv_image_original.setImage(SwingFXUtils.toFXImage(dicom
_image
						.get(0).getOriginalImage(), null));
				if (tab_original_image.getChildren().isEmpty())
{

	tab_original_image.getChildren().add(iv_image_original);
				} else {

	tab_original_image.getChildren().remove(iv_image_original);

	tab_original_image.getChildren().add(iv_image_original);
				}
				setOriginalImageInfo(dicom_image.get(0));
			}
		});
	}

	@FXML
	private void exportStart() {
		int text_counter = -1;
		String doctors_comment = popup_comments.getText();
		ArrayList<String> text = new ArrayList<String>();

		try {
			if (!doctors_comment.equals("")) {
				text.add("Doctor's comments: ");
				text.add(doctors_comment);
				text.add("");
			}

			String new_sop_id =
nodules.get(0).nativeGetImageSOP_UID();
```

```
                int dicom_counter = 0;
                for (int i = 0; i < nodules.size(); i++) {

                    if
(!new_sop_id.equals(nodules.get(i).nativeGetImageSOP_UID())) {
                        text.add("|");
                    }

                    text.add("Patient id: " +
nodules.get(i).nativeGetPatientId());
                    text.add("Series Instance UID: "
                            +
nodules.get(i).nativeGetSeriesInstanceUid());
                    text.add("Study Instance UID: "
                            +
nodules.get(i).nativeGetStudyInstanceUid());
                    text.add("Image SOP UID: "
                            +
nodules.get(i).nativeGetImageSOP_UID());
                    text.add("");

                    if (dicom_counter == 0) {
                        text.add("Colors of cancerous nodules:");
                        text.add("   "
                                + dicom_image.get(dicom_counter)
                                        .getCancerousColors());
                        text.add("");
                        text.add("Colors of non-cancerous
nodules:");
                        text.add("   "
                                + dicom_image.get(dicom_counter)

    .getNonCancerousColors());
                        text.add("");
                        dicom_counter++;
                    }

                    if
(!new_sop_id.equals(nodules.get(i).nativeGetImageSOP_UID())) {
                        text.add("Colors of cancerous nodules:");
                        text.add("   "
                                + dicom_image.get(dicom_counter)
                                        .getCancerousColors());
                        text.add("");
                        text.add("Colors of non-cancerous
nodules:");
                        text.add("   "
                                + dicom_image.get(dicom_counter)

    .getNonCancerousColors());
                        text.add("");
                        new_sop_id =
nodules.get(i).nativeGetImageSOP_UID();
                        dicom_counter++;
                    }

                    text.add("Nodule #" + (i + 1));
                    text.add("Color: " +
nodules.get(i).nativeGetColor());
                    text.add("");

                    text.add("Mean: " +
nodules.get(i).nativeGetMean());
                    text.add("Standard deviation: " +
nodules.get(i).nativeGetSd());

                    text.add("");
                    text.add("Coordinates of nodule: ");
                    for (int j = 0; j <
nodules.get(i).getXYCoord().size(); j += 10) {
                        String temp =
nodules.get(i).getXYCoord().get(j) + ", ";
                        if (j + 1 <
nodules.get(i).getXYCoord().size()) {
                            temp +=
nodules.get(i).getXYCoord().get(j + 1) + ", ";
                        }
                        if (j + 2 <
nodules.get(i).getXYCoord().size()) {
                            temp +=
nodules.get(i).getXYCoord().get(j + 2) + ", ";
                        }
                        if (j + 3 <
nodules.get(i).getXYCoord().size()) {
                            temp +=
nodules.get(i).getXYCoord().get(j + 3) + ", ";
                        }
                        if (j + 4 <
nodules.get(i).getXYCoord().size()) {
                            temp +=
nodules.get(i).getXYCoord().get(j + 4) + ", ";
                        }
                        if (j + 5 <
nodules.get(i).getXYCoord().size()) {
                            temp +=
nodules.get(i).getXYCoord().get(j + 5) + ", ";
                        }
                        if (j + 6 <
nodules.get(i).getXYCoord().size()) {
                            temp +=
nodules.get(i).getXYCoord().get(j + 6) + ", ";
                        }
                        if (j + 7 <
nodules.get(i).getXYCoord().size()) {
                            temp +=
nodules.get(i).getXYCoord().get(j + 7) + ", ";
                        }
                        if (j + 8 <
nodules.get(i).getXYCoord().size()) {
                            temp +=
nodules.get(i).getXYCoord().get(j + 8) + ", ";
                        }
                        if (j + 9 <
nodules.get(i).getXYCoord().size()) {
                            temp +=
nodules.get(i).getXYCoord().get(j + 9) + ", ";
                        }
                        text.add(temp);
                    }

                    text.add("");
                    text.add("");
                    text.add("Predicted Data");
                    text.add("   Cancerous: "
                            +
nodules.get(i).nativeGetPredictedisCancerous());
                    text.add("       Probabilities of malignancy:");
                    text.add("          yes: "
                            +
nodules.get(i).proby_5Property().get());
                    text.add("          no: "
                            +
nodules.get(i).proby_0Property().get());
                    text.add(" ");
                    text.add("-----------------------------------------------
----");
                }

                File temp_image = new File(directory +
"\\temp.jpg");
```

```
                OutputStream out = null;
                PDDocument doc = null;
                PDPage page = null;
                PDXObjectImage ximage = null;
                doc = new PDDocument();

                int fontSize = 12;
                PDSimpleFont font = PDType1Font.HELVETICA;
                int margin = 40;
                float height =
font.getFontDescriptor().getFontBoundingBox()
                        .getHeight() / 1000;

                height = height * fontSize * 1.05f;
                page = new PDPage();
                PDPageContentStream contentStream = null;
                float y = -1;

                for (int i = 0; i < dicom_image.size(); i++) {
                    try {
                        out = new BufferedOutputStream(new
FileOutputStream(
                                temp_image));
                        JPEGImageEncoder enc =
JPEGCodec.createJPEGEncoder(out);

    enc.encode(dicom_image.get(i).getOriginalImage());
                        out.close();
                    } catch (Exception e) {

                    }
                    try {
                        page = new PDPage();
                        doc.addPage(page);
                        BufferedImage img =
ImageIO.read(temp_image);
                        ximage = new PDJpeg(doc, img, 1.0f);
                        PDPageContentStream content = new
PDPageContentStream(doc,
                                page);
                        content.drawImage(ximage, 50, 145);
                        content.close();
                        out = new BufferedOutputStream(new
FileOutputStream(
                                temp_image));
                        JPEGImageEncoder enc =
JPEGCodec.createJPEGEncoder(out);
                        enc.encode(image_segmented.get(i));
                        out.close();

                        page = new PDPage();
                        doc.addPage(page);
                        img = ImageIO.read(temp_image);
                        ximage = new PDJpeg(doc, img, 1.0f);
                        content = new PDPageContentStream(doc,
page);
                        content.drawImage(ximage, 50, 145);
                        content.close();

                        text: for (int p = text_counter + 1; p <
text.size(); ++p) {
                            if (y < margin) {
                                page = new PDPage();
                                doc.addPage(page);
                                if (contentStream != null) {
                                    contentStream.endText();
                                    contentStream.close();
                                }
                                contentStream = new
PDPageContentStream(doc, page);
                                    contentStream.setFont(font,
fontSize);
                                    contentStream.beginText();
                                    y =
page.getMediaBox().getHeight() - margin
                                            + height;

    contentStream.moveTextPositionByAmount(margin, y);
                                }
                                if (contentStream != null) {

    contentStream.moveTextPositionByAmount(0, -height);
                                    y -= height;
                                    if (!text.get(p).equals("|")) {
                                        contentStream

    .drawString(text.get(p).toString());
                                    }
                                }
                                if (text.get(p).equals("|")) {
                                    text_counter = p;
                                    break text;
                                }
                            }

                            if (contentStream != null) {
                                contentStream.endText();
                                contentStream.close();
                            }

                            y = -1;
                            contentStream = null;

                        } catch (IOException ie) {
                            ie.printStackTrace();
                        } catch (Exception e) {
                            e.printStackTrace();
                        }
                    }
                    DateFormat dateFormat = new
SimpleDateFormat("yyyy.MM.dd HH.mm.ss");
                    Date date = new Date();

                    String generated_pdf_file = directory + "\\LNDCT
report "
                            + dateFormat.format(date) + ".pdf";
                    doc.save(generated_pdf_file);

                    File pdfFile = new File(generated_pdf_file);

                    if (pdfFile.exists()) {
                        final Stage myDialog = new Stage();
                        if (Desktop.isDesktopSupported()) {
                            Desktop.getDesktop().open(pdfFile);
                        } else {
                            Dialogs.showErrorDialog(myDialog, "No
data was loaded.",
                                    "Error", "LNDCT");
                        }
                    } else {
                    }

                } catch (Exception e) {
                    System.out.println(e);
                }
            }

    @FXML
    private void about() {
        final Stage dialog_about = new Stage();
        Dialogs.showInformationDialog(
                dialog_about,
                "University of the Philippines Manila \n "
                        + "Copyright 2013 \n Programmer:
Jhesed D. Tacadena \n Adviser:  Professor Geoffrey Solano",
                "About Lung Nodule Detector and Classifier
Tool", "LNDCT ");
    }

    @FXML
    private void tutorial() {
        final Stage dialog_tutorial = new Stage();

    dialog_tutorial.initModality(Modality.WINDOW_MODAL);
        dialog_tutorial.setTitle("LNDCT Tutorial");

        // tutorial
        page.setImage(SwingFXUtils.toFXImage(
                tutorial_images.get(tutorial_page), null));

        HBox hbox = new HBox();
        hbox.setPadding(new Insets(15, 12, 15, 12));
        hbox.setSpacing(10);
        hbox.setStyle("-fx-background-color: #a0522d;");

        Button buttonCurrent = new Button("previous");
        buttonCurrent.setPrefSize(100, 20);

        Button buttonProjected = new Button("next");
        buttonProjected.setPrefSize(100, 20);

        buttonCurrent.setOnAction(new
EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) {
                if (tutorial_page > -1) {
                    tutorial_page--;
                    page.setImage(SwingFXUtils.toFXImage(
                            tutorial_images.get(tutorial_page),
null));
                }
            }
        });

        buttonProjected.setOnAction(new
EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) {
                if (tutorial_page < tutorial_images.size()) {
                    tutorial_page++;
                    page.setImage(SwingFXUtils.toFXImage(
                            tutorial_images.get(tutorial_page),
null));
                }
            }
        });

        hbox.getChildren().addAll(buttonCurrent,
buttonProjected);

        VBox vbox = new VBox();
        vbox.setPadding(new Insets(10));
        vbox.setSpacing(8);
        vbox.getChildren().add(page);

        VBox general_vbox = new VBox();
        general_vbox.getChildren().addAll(hbox, vbox);

        AnchorPane tutorial = new AnchorPane();
        tutorial.getChildren().addAll(general_vbox);
        Scene myDialogScene = new Scene(tutorial);

        dialog_tutorial.setScene(myDialogScene);
        dialog_tutorial.show();
    }

    @SuppressWarnings({ "rawtypes" })
    private int getScale(ChoiceBox scale) {
        int choice =
scale.getSelectionModel().getSelectedIndex();
        if (choice == SCALE2) {
            return 2;
        } else if (choice == SCALE1) {
            return 1;
        } else if (choice == SCALE3) {
            return 3;
        }
        return 4;
    }

    protected String fileExtensionParser(String file_name) {
        String extension = "";
        int i = file_name.lastIndexOf('.');
        if (i > 0) {
            extension = file_name.substring(i + 1);
        }
        return extension;
    }

    protected String imageNameParser(String file_name) {
        String extension = "";
        int i = file_name.lastIndexOf('\\');
        if (i > 0) {
            extension = file_name.substring(i + 1);
        }
        return extension;
    }
}


/**
 * @author jhesed tacadena
 */

package lungnoduledc;

import java.io.InputStream;

final public class ResourceLoader {
    public static InputStream load(String path) {
        InputStream input =
ResourceLoader.class.getResourceAsStream(path);
        if (input == null) {
            input =
ResourceLoader.class.getResourceAsStream("/" + path);
        }
        return input;
    }
}


/**
 * @author jhesed tacadena
 */

package featureextraction;

import java.util.ArrayList;

import javafx.beans.property.SimpleDoubleProperty;
import javafx.beans.property.SimpleStringProperty;

public class Nodule {

	private SimpleStringProperty file_name;
	private SimpleStringProperty patient_id;
	private SimpleStringProperty study_instance_uid;
	private SimpleStringProperty series_instance_uid;
	private SimpleStringProperty imageSOP_UID;
	private SimpleStringProperty nodule_id;
	private SimpleStringProperty color;
	private SimpleStringProperty pixel_coordinates;
	private SimpleDoubleProperty mean;
	private SimpleDoubleProperty sd;
	private SimpleStringProperty is_cancerous;
	private SimpleStringProperty malignancy;
	private SimpleStringProperty prediction;
	private String sop_primary_key;
	private String coord;
	private int perimeter;
	private int area;
	private double circularity;
	private ArrayList<String> xy_coord;
	private ArrayList<Integer> x_coord;
	private ArrayList<Integer> y_coord;
	private Integer x_min;
	private Integer x_max;
	private Integer y_min;
	private Integer y_max;
	private Integer x_center;
	private Integer y_center;
	private Integer width;
	private Integer height;
	private double aspect_ratio;
	private double roundness;
	private SimpleStringProperty prediction_is_cancerous;
	private SimpleStringProperty prediction_malignancy;
	private SimpleDoubleProperty proby_0;
	private SimpleDoubleProperty proby_1;
	private SimpleDoubleProperty proby_2;
	private SimpleDoubleProperty proby_3;
	private SimpleDoubleProperty proby_4;
	private SimpleDoubleProperty proby_5;
	private int int_color;

	public Nodule(String file_name, String sop_primary_key, 
String patient_id,
			String study_instance_uid, String 
series_instance_uid,
			String imageSOP_UID, String color, int int_color) {

		this.file_name = new SimpleStringProperty(file_name);
		this.sop_primary_key = sop_primary_key;
		this.patient_id = new SimpleStringProperty(patient_id);
		this.study_instance_uid = new 
SimpleStringProperty(study_instance_uid);
		this.series_instance_uid = new 
SimpleStringProperty(series_instance_uid);
		this.imageSOP_UID = new 
SimpleStringProperty(imageSOP_UID);
		this.color = new SimpleStringProperty(color);
		this.is_cancerous = new SimpleStringProperty("no");
		this.proby_0 = new SimpleDoubleProperty(0.0);
		this.proby_1 = new SimpleDoubleProperty(0.0);
		this.proby_2 = new SimpleDoubleProperty(0.0);
		this.proby_3 = new SimpleDoubleProperty(0.0);
		this.proby_4 = new SimpleDoubleProperty(0.0);
		this.proby_5 = new SimpleDoubleProperty(0.0);
		this.mean = new SimpleDoubleProperty(0.0);
		this.sd = new SimpleDoubleProperty(0.0);
		this.coord = "";
		this.xy_coord = new ArrayList<String>();
		this.malignancy = new SimpleStringProperty("0");
		this.prediction_is_cancerous = new 
SimpleStringProperty("no");
		this.int_color = int_color;

		x_coord = new ArrayList<Integer>();
		y_coord = new ArrayList<Integer>();
		x_min = 0;
		x_max = 0;
		y_min = 0;
		y_max = 0;
		x_center = 0;
		y_center = 0;
		width = 0;
		height = 0;
		aspect_ratio = 0.0;
	}

	public void addXYCoord(String xy) {
		this.xy_coord.add(xy);
	}

	public void addXCoord(Integer x) {
		this.x_coord.add(x);
	}

	public void addYCoord(Integer y) {
		this.y_coord.add(y);
	}

	public ArrayList<Integer> getXCoord() {
		return this.x_coord;
	}

	public ArrayList<Integer> getYCoord() {
		return this.y_coord;
	}

	public void setXMin(Integer x_min) {
		this.x_min = x_min;
	}

	public void setXMax(Integer x_max) {
		this.x_max = x_max;
	}

	public void setYMin(Integer y_min) {
		this.y_min = y_min;
	}

	public void setYMax(Integer y_max) {
		this.y_max = y_max;
	}

	public void setXCenter(Integer x_center) {
		this.x_center = x_center;
	}

	public void setYCenter(Integer y_center) {
		this.y_center = y_center;
	}

	public void setHeight(Integer height) {
		this.height = height;
	}

	public void setWidth(Integer width) {
		this.width = width;
	}

	public void setAspectRatio(double ratio) {
		this.aspect_ratio = ratio;
	}

	public Integer getXMin() {
		return this.x_min;
	}

	public Integer getXMax() {
		return this.x_max;
	}

	public Integer getYMin() {
		return this.y_min;
	}

	public Integer getYMax() {
		return this.y_max;
	}

	public Integer getXCenter() {
		return this.x_center;
	}

	public Integer getYCenter() {
		return this.y_center;
	}

	public Integer getHeight() {
		return this.height;
	}

	public Integer getWidth() {
		return this.width;
	}

	public int getIntColor() {
		return this.int_color;
	}

	public double getAspectRatio() {
		return this.aspect_ratio;
	}

	public String getXYCoord(int index) {
		return this.xy_coord.get(index);
	}

	public ArrayList<String> getXYCoord() {
		return this.xy_coord;
	}

	public int countCoord() {
		return this.xy_coord.size();
	}

	public void createCoordString() {
		for (int i = 0; i < this.xy_coord.size(); i++) {
			this.coord += "{" + xy_coord.get(i) + "},  ";
		}
		this.pixel_coordinates = new
SimpleStringProperty(this.coord);
	}

	public SimpleStringProperty patient_idProperty() {
		return this.patient_id;
	}

	public SimpleStringProperty study_instance_uidProperty() {
		return this.study_instance_uid;
	}

	public SimpleStringProperty series_instance_uidProperty() {
		return this.series_instance_uid;
	}

	public SimpleStringProperty imageSOP_UIDProperty() {
		return this.imageSOP_UID;
	}

	public SimpleStringProperty nodule_idProperty() {
		return this.nodule_id;
	}

	public SimpleStringProperty colorProperty() {
		return this.color;
	}

	public SimpleStringProperty malignancyProperty() {
		return this.malignancy;
	}

	public SimpleStringProperty pixel_coordinatesProperty() {
		return this.pixel_coordinates;
	}

	public SimpleDoubleProperty meanProperty() {
		return this.mean;
	}

	public SimpleDoubleProperty sdProperty() {
		return this.sd;
	}

	public SimpleStringProperty is_cancerousProperty() {
		return this.is_cancerous;
	}

	public SimpleStringProperty file_nameProperty() {
		return this.file_name;
	}

	public SimpleStringProperty predictionProperty() {
		return this.prediction;
	}

	public SimpleDoubleProperty proby_0Property() {
		return this.proby_0;
	}

	public SimpleDoubleProperty proby_1Property() {
		return this.proby_1;
	}

	public SimpleDoubleProperty proby_2Property() {
		return this.proby_2;
	}

	public SimpleDoubleProperty proby_3Property() {
		return this.proby_3;
	}

	public SimpleDoubleProperty proby_4Property() {
		return this.proby_4;
	}

	public SimpleDoubleProperty proby_5Property() {
		return this.proby_5;
	}

	public void setPrediction(String prediction) {
		this.prediction = new SimpleStringProperty(prediction);
	}

	public SimpleStringProperty table_areaProperty() {
		return new SimpleStringProperty(this.area + " px");
	}

	public SimpleStringProperty table_centerProperty() {
		return new SimpleStringProperty("{" + this.x_center + ",
"
				+ this.y_center + "}");
	}

	public SimpleStringProperty table_heightProperty() {
		return new SimpleStringProperty(this.height + " px");
	}

	public SimpleStringProperty table_widthProperty() {
		return new SimpleStringProperty(this.width + " px");
	}

	public SimpleStringProperty table_ratioProperty() {
		return new SimpleStringProperty(this.aspect_ratio + "");
	}

	public void setImageSOP_UID(String imageSOP_UID) {
		this.imageSOP_UID = new
SimpleStringProperty(imageSOP_UID);
	}

	public void setMean(double mean) {
		this.mean = new SimpleDoubleProperty(mean);
	}

	public void setSd(double sd) {
		this.sd = new SimpleDoubleProperty(sd);
	}

	public void setIsCancerous(String is_cancerous) {
		this.is_cancerous = new
SimpleStringProperty(is_cancerous);
	}

	public void setMalignancy(String malignancy) {
		this.malignancy = new
SimpleStringProperty(malignancy);
	}

	public String nativeGetPatientId() {
		return this.patient_id.get();
	}

	public String nativeGetStudyInstanceUid() {
		return this.study_instance_uid.get();
	}

	public String nativeGetSeriesInstanceUid() {
		return this.series_instance_uid.get();
	}

	public String nativeGetImageSOP_UID() {
		return this.imageSOP_UID.get();
	}

	public String nativeGetNoduleId() {
		return this.nodule_id.get();
	}

	public String nativeGetColor() {
		return this.color.get();
	}

	public double nativeGetMean() {
		return this.mean.get();
	}

	public double nativeGetSd() {
		return this.sd.get();
	}

	public int nativeGetArea() {
		return this.area;
	}

	public double nativeGetCircularity() {
		return this.circularity;
	}

	public int nativeGetPerimeter() {
		return this.perimeter;
	}

	public String nativeGetMalignancy() {
		return this.malignancy.get();
	}

	public String nativeGetCancerous() {
		return this.is_cancerous.get();
	}

	public String getPrimarySopUid() {
		return this.sop_primary_key;
	}

	public double getRoundness() {
		return this.roundness;
	}

	public void setRoundness(double roundness) {
		this.roundness = roundness;
	}

	public void setIsPredictionCancerous(String is_cancerous) {
		this.prediction_is_cancerous = new
SimpleStringProperty(is_cancerous);
	}

	public void setPredictionMalignancy(String malignancy) {
		this.prediction_malignancy = new
SimpleStringProperty(malignancy);
	}

	public SimpleStringProperty
prediction_is_cancerousProperty() {
		return this.prediction_is_cancerous;
	}

	public SimpleStringProperty prediction_malignancyProperty()
{
		return this.prediction_malignancy;
	}

	public String nativeGetPredictedisCancerous() {
		return this.prediction_is_cancerous.get();
	}

	public String nativeGetPredictedMalignancy() {
```

```
            return this.prediction_malignancy.get();
      }

      public void setPerimeter(int perimeter) {
            this.perimeter = perimeter;
      }

      public void setArea(int area) {
            this.area = area;
      }

      public void setCircularity(double circularity) {
            this.circularity = circularity;
      }

      public void setProby0(Double proby_0) {
            this.proby_0 = new SimpleDoubleProperty(proby_0);
      }

      public void setProby1(Double proby_1) {
            this.proby_1 = new SimpleDoubleProperty(proby_1);
      }

      public void setProby2(Double proby_2) {
            this.proby_2 = new SimpleDoubleProperty(proby_2);
      }

      public void setProby3(Double proby_3) {
            this.proby_3 = new SimpleDoubleProperty(proby_3);
      }

      public void setProby4(Double proby_4) {
            this.proby_4 = new SimpleDoubleProperty(proby_4);
      }

      public void setProby5(Double proby_5) {
            this.proby_5 = new SimpleDoubleProperty(proby_5);
      }

      public String getPixelCoordinates() {
            return this.pixel_coordinates.get();
      }
}


/**
 * @author jhesed tacadena
 */

package fileprocessing;

import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.IOException;
import java.util.Iterator;
import java.util.logging.Level;
import java.util.logging.Logger;

import javax.imageio.ImageIO;
import javax.imageio.ImageReader;
import javax.imageio.stream.ImageInputStream;

import lungnoduledc.MainController;

import org.dcm4che.data.Dataset;
import org.dcm4che.dict.Tags;
import org.dcm4che.imageio.plugins.DcmMetadata;

public class DicomImage {
      private String file_name;
      private BufferedImage dcm_image;
      private Dataset dataset;
      private int width;
      private int height;
      private String refSOP_classUID;
      private String ref_series_instanceUID;
      private String study_instanceUID;
      private String patientID;
      private String studyID;
      private String dimension;
      private File file;
      private String cancerous_colors;
      private String noncancerous_colors;

      public DicomImage(File file, String file_name) {
            this.file_name = file_name;
            this.refSOP_classUID = "not available";
            this.ref_series_instanceUID = "not available";
            this.study_instanceUID = "not available";
            this.patientID = "not available";
            this.studyID = "not available";
            this.dimension = "not available";
            this.file = file;
            getDicomImage();
            this.width = this.dcm_image.getWidth();
            this.height = this.dcm_image.getHeight();
      }

      @SuppressWarnings("rawtypes")
      public void getDicomImage() {
            ImageInputStream iis = null;
            ImageReader reader = null;
            try {
                  iis = ImageIO.createImageInputStream(file);
                  Iterator iter = 
ImageIO.getImageReadersByFormatName("DICOM");
                  reader = (ImageReader) iter.next();
                  reader.setInput(iis, false);
                  dcm_image = reader.read(0);

                  this.dataset = ((DcmMetadata) 
reader.getStreamMetadata())
                              .getDataset();
                  setDicomMetadata();
            } catch (IOException ex) {
                  System.out.println("Error in reading image.");

      Logger.getLogger(MainController.class.getName()).log(Level
.SEVERE,
                              null, ex);
            }
      }

      public static BufferedImage createImageFromBytes(byte[] 
bytes)
                  throws IOException {
            ByteArrayInputStream bis = new 
ByteArrayInputStream(bytes);
            Iterator<?> readers = 
ImageIO.getImageReadersByFormatName("DICOM");
            ImageReader reader = (ImageReader) readers.next();
            Object source = bis;
            ImageInputStream iis = 
ImageIO.createImageInputStream(source);
            reader.setInput(iis, true);
            return reader.read(0);
      }
	public static BufferedImage
getBufferedImageFromBytes(byte[] bytes) {

		try {
			return (ImageIO.read(new
ByteArrayInputStream(bytes)));
		} catch (IOException e) {
			e.printStackTrace();
		}
		return null;
	}

	private void setDicomMetadata() {
		this.width = dcm_image.getWidth();
		this.height = dcm_image.getHeight();
		if (dataset.getString(Tags.SOPInstanceUID) != null) {
			this.refSOP_classUID =
dataset.getString(Tags.SOPInstanceUID);
		}
		if (dataset.getString(Tags.StudyInstanceUID) != null) {
			this.study_instanceUID =
dataset.getString(Tags.StudyInstanceUID);
		}
		if (dataset.getString(Tags.SeriesInstanceUID) != null) {
			this.ref_series_instanceUID = dataset
					.getString(Tags.SeriesInstanceUID);
		}
		if (dataset.getString(Tags.PatientID) != null) {
			this.patientID = dataset.getString(Tags.PatientID);
		}
		if (dataset.getString(Tags.SOPClassUID) != null) {
			this.studyID = dataset.getString(Tags.SOPClassUID);
		}
		if (dataset.getString(Tags.PixelSpacing) != null) {
			this.dimension =
dataset.getString(Tags.PixelSpacing);
		}
	}

	public BufferedImage getOriginalImage() {
		return this.dcm_image;
	}

	public File getDcmFile() {
		return this.file;
	}

	public int getWidth() {
		return this.width;
	}

	public int getHeight() {
		return this.height;
	}

	public String getRefSOPClassUID() {
		return this.refSOP_classUID;
	}

	public String getSeriesInstanceUID() {
		return this.ref_series_instanceUID;
	}

	public String getStudyInstanceUID() {
		return this.study_instanceUID;
	}

	public String getPatientID() {
		return this.patientID;
	}

	public String getStudyID() {
		return this.studyID;
	}

	public String getDimension() {
		return this.dimension;
	}

	public String getFileName() {
		return this.file_name;
	}

	public String getCancerousColors() {
		return this.cancerous_colors;
	}

	public String getNonCancerousColors() {
		return this.noncancerous_colors;
	}

	public void setCancerousColors(String cancerous) {
		this.cancerous_colors = cancerous;
	}

	public void setNonCancerousColors(String noncancerous) {
		this.noncancerous_colors = noncancerous;
	}

}


/**
 * uses SAP parser
 * @author jhesed tacadena
 */

package fileprocessing;

import java.io.IOException;
import java.util.ArrayList;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

public class XmlParser extends DefaultHandler {

	private KnownData known_data;
	private ArrayList<KnownData> known_data_list = new
ArrayList<KnownData>();
	private String xml_file;
	private String temp;
	private String x;
	private String y;

	private int[] malignancy_count = new int[4];
	private ArrayList<String> xy;

	public void startParsing(XmlParser handler) throws
IOException,
			SAXException, ParserConfigurationException {

		this.x = "";
		this.y = "";
		this.xy = new ArrayList<String>();
		SAXParserFactory spfac =
SAXParserFactory.newInstance();
		SAXParser sp = spfac.newSAXParser();
		sp.parse(this.xml_file, handler);
	}

	public XmlParser(String xml_file) {
		this.xml_file = xml_file;
	}

	@Override
	public void characters(char[] buffer, int start, int length) {
		temp = new String(buffer, start, length);
	}

	@Override
	public void startElement(String uri, String localName, String
qName,
			Attributes attributes) throws SAXException {
		temp = "";
		if (qName.equalsIgnoreCase("unblindedReadNodule")) {
			known_data = new KnownData();
			known_data.setType(attributes.getValue("type"));
		}
	}

	@Override
	public void endElement(String uri, String localName, String
qName)
			throws SAXException {

		if (qName.equalsIgnoreCase("unblindedReadNodule")) {
			known_data.addXYCoordinates(xy);
			known_data_list.add(known_data);
		} else if (qName.equalsIgnoreCase("noduleID")) {
			known_data.setNoduleId(temp);
		} else if (qName.equalsIgnoreCase("malignancy")) {
			known_data.setMalignancy(temp);
		} else if (qName.equalsIgnoreCase("imageZPosition")) {
			known_data.addImageZPosition(temp);
		} else if (qName.equalsIgnoreCase("imageSOP_UID")) {
			known_data.addImageSopUid(temp);
			known_data.addXYCoordinates(xy);
			x = "";
			y = "";
		} else if (qName.equalsIgnoreCase("xCoord")) {
			x = temp;
		} else if (qName.equalsIgnoreCase("yCoord")) {
			y = temp;
			xy.add(x + "," + y);
		}
	}

	public int getNumberOfKnownMalignantNodules() {
		int max = 0;
		for (int i = 0; i < 4; i++) {
			if (malignancy_count[i] > max) {
				max = malignancy_count[i];
			}
		}
		return max;
	}

	public KnownData getIndividualKnownData() {
		return this.known_data;
	}

	public ArrayList<KnownData> getKnownDataList() {
		return this.known_data_list;
	}
}


/**
 * @author jhesed tacadena
 */

package fileprocessing;

import java.util.ArrayList;

public class KnownData {
	@SuppressWarnings("unused")
	private String type;
	private String patient_id;
	private String series_instance_uid;
	private String study_instance_uid;
	private ArrayList<String> image_z_position;
	private ArrayList<String> image_sop_uid;
	private String nodule_id;
	private String malignancy;
	private ArrayList<ArrayList<String>> xy_coordinates;

	public KnownData() {
		this.type = null;
		this.patient_id = null;
		this.series_instance_uid = null;
		this.study_instance_uid = null;
		this.image_z_position = null;
		this.image_sop_uid = null;
		this.image_z_position = new ArrayList<String>();
		this.image_sop_uid = new ArrayList<String>();
		this.nodule_id = null;
		this.xy_coordinates = new
ArrayList<ArrayList<String>>();
		this.malignancy = "0";
	}

	public String getPatientId() {
		return this.patient_id;
	}

	public String getSeriesInstanceUid() {
		return this.series_instance_uid;
	}

	public String getStudyInstanceUid() {
		return this.study_instance_uid;
	}

	public ArrayList<String> getImageZPosition() {
		return this.image_z_position;
	}

	public void setPatientID(String patient_id) {
		this.patient_id = patient_id;
	}

	public void setSeriesInstanceUid(String series_instance_uid) {
		this.series_instance_uid = series_instance_uid;
	}

	public void setStudyInstanceUid(String study_instance_uid) {
		this.study_instance_uid = study_instance_uid;
	}

	public void addImageZPosition(String image_z_position) {
		this.image_z_position.add(image_z_position);
	}
	public void setImageZPosition(String image_z_position, int
index) {
		this.image_z_position.set(index, image_z_position);
	}

	public ArrayList<String> getImageSopUid() {
		return this.image_sop_uid;
	}

	public String getNoduleId() {
		return this.nodule_id;
	}

	public ArrayList<String> getXYCoordinates(int index) {
		return this.xy_coordinates.get(index);
	}

	public String getMalignancy() {
		return this.malignancy;
	}

	public String getImageZPosition(int index) {
		return this.image_z_position.get(index);
	}

	public String getImageSopUid(int index) {
		return this.image_sop_uid.get(index);
	}

	public String getXYCoordinates(int position, int index) {
		return this.xy_coordinates.get(position).get(index);
	}

	public void addImageSopUid(String image_sop_uid) {
		this.image_sop_uid.add(image_sop_uid);
	}

	public void addXYCoordinates(ArrayList<String>
xy_coordinate) {
		this.xy_coordinates.add(xy_coordinate);
	}

	public void setImageSopUid(String image_sop_uid, int index)
{
		this.image_sop_uid.set(index, image_sop_uid);
	}

	public void setNoduleId(String nodule_id) {
		this.nodule_id = nodule_id;
	}

	public void setXYCoordinates(int position, String
xy_coordinate, int index) {
		this.xy_coordinates.get(position).set(index,
xy_coordinate);
	}

	public void setMalignancy(String malignancy) {
		this.malignancy = malignancy;
	}

	public int countData() {
		return this.image_sop_uid.size();
	}

	public void setType(String type) {
		this.type = type;
	}

	public void combineXY(int position, String x, String y) {
		this.xy_coordinates.get(position).add(x + "," + y);
	}
}


/**
 * @author jhesed tacadena
 */

package util;

import java.awt.Image;
import java.awt.image.BufferedImage;
import java.awt.image.ColorModel;
import java.awt.image.DataBufferByte;
import java.awt.image.PixelGrabber;
import java.awt.image.WritableRaster;

public class ImageUtil {

	public static void imageTo255(int pixel) {
		pixel = ((int) (0.298912D * (0xFF & pixel >> 16) +
0.586611D
				* (0xFF & pixel >> 8) + 0.114478D * (0xFF &
pixel)));
	}

	public static BufferedImage
toExportableImage(BufferedImage image) {
		for (int i = 0; i < image.getHeight(); i++) {
			for (int j = 0; j < image.getWidth(); j++) {
				int pixel = image.getRGB(j, i);
				pixel = ((int) (0.298912D * (0xFF & pixel >>
16) + 0.586611D
						* (0xFF & pixel >> 8) + 0.114478D *
(0xFF & pixel)));
				image.setRGB(j, i, pixel);
			}
		}
		return image;
	}

	public static int[] twoDtoOneD(int[][] image, int height, int
width) {
		int[] oned = new int[height * width];
		for (int i = 0; i < height; i++) {
			int offset = i * height;
			for (int j = 0; j < width; j++) {
				oned[offset + j] = image[j][i];
			}
		}
		return oned;
	}

	public static int[][] clone(int[][] image) {
		int[][] copy = new int[image.length][];
		for (int i = 0; i < image.length; i++) {
			copy[i] = image[i].clone();
		}
		return copy;
	}

	public static BufferedImage
copyBufferedImage(BufferedImage bi) {
		ColorModel cm = bi.getColorModel();
		boolean isAlphaPremultiplied =
cm.isAlphaPremultiplied();
		WritableRaster raster = bi.copyData(null);
		BufferedImage image = new BufferedImage(cm, raster,
				isAlphaPremultiplied, null);
		return image;
	}

	public static BufferedImage getImageFromArray(int[] pixels,
int width,
			int height) {
		BufferedImage image = new BufferedImage(width,
height,
				BufferedImage.TYPE_INT_RGB);
		image.setRGB(0, 0, width, height, pixels, 0, width);
		return image;
	}

	public static BufferedImage getWhiteImageFromArray(int[]
pixels, int width,
			int height) {
		BufferedImage image = new BufferedImage(width,
height,
				BufferedImage.TYPE_INT_RGB);

		for (int i = 0; i < height; i++) {
			int offset = i * height;
			for (int j = 0; j < width; j++) {
				if (pixels[offset + j] != -16777216) {
					image.setRGB(j, i, -1);
				} else {
					image.setRGB(j, i, -16777216);
				}
			}
		}
		return image;
	}

	public static BufferedImage toFloatImage(double[] pixels, int
width,
			int height) {
		BufferedImage image = new BufferedImage(width,
height,
				BufferedImage.TYPE_INT_RGB);
		final WritableRaster wr = (WritableRaster)
image.getData();
		wr.setPixels(0, 0, width, height, pixels);
		return image;
	}

	public static int[][] getDicomPixels(BufferedImage image) {
		int[][] pixels = new
int[image.getWidth()][image.getHeight()];

		for (int i = 0; i < image.getWidth(); i++)
			for (int j = 0; j < image.getHeight(); j++)
				pixels[i][j] = image.getRGB(i, j);

		return pixels;
	}

	public static int[] getPixels(Image paramImage, int width, int
height) {
		int[] pixels = new int[width * height];

		PixelGrabber localPixelGrabber = new
PixelGrabber(paramImage, 0, 0,
				width, height, pixels, 0, width);

		try {
			localPixelGrabber.grabPixels(3000L);
		} catch (InterruptedException localInterruptedException)
{
		}
		for (int i = 0; i < width * height; i++) {
			pixels[i] = ((int) (0.298912D * (0xFF & pixels[i] >>
16)
					+ 0.586611D * (0xFF & pixels[i] >> 8) +
0.114478D * (0xFF & pixels[i])));
		}
		return pixels;
	}

	public static int colorToRGB(int alpha, int red, int green, int
blue) {
		int newPixel = 0;
		newPixel += alpha;
		newPixel = newPixel << 8;
		newPixel += red;
		newPixel = newPixel << 8;
		newPixel += green;
		newPixel = newPixel << 8;
		newPixel += blue;
		return newPixel;
	}

	public static BufferedImage arrayToBufferedImage(int[][]
image_matrix,
			int width, int height) {
		BufferedImage image = new BufferedImage(width,
height,
				BufferedImage.TYPE_BYTE_GRAY);

		for (int i = 0; i < image_matrix.length; i++) {
			for (int j = 0; j < image_matrix[i].length; j++) {
				image.setRGB(i, j, image_matrix[i][j]);
			}
		}
		return image;
	}

	public static BufferedImage arrayToColoredBufferedImage(
			int[][] image_matrix, int width, int height) {
		BufferedImage image = new BufferedImage(width,
height,
				BufferedImage.TYPE_INT_RGB);

		for (int i = 0; i < image_matrix.length; i++) {
			for (int j = 0; j < image_matrix[i].length; j++) {
				image.setRGB(i, j, image_matrix[i][j]);
			}
		}
		return image;
	}

	public static BufferedImage arrayToColoredBufferedImage2(
			int[] image_matrix, int width, int height) {
		BufferedImage image = new BufferedImage(width,
height,
				BufferedImage.TYPE_INT_RGB);

		for (int i = 0; i < height; i++) {
			int offset = i * height;
			for (int j = 0; j < width; j++) {
				image.setRGB(i, j, image_matrix[offset + j]);
			}
		}
		return image;
	}

	public static BufferedImage fromArrToImage(BufferedImage
original,
			int[] wltpix, int height, int width) {
		BufferedImage image_wavelet = original;
		int counter = 0;
		for (int i = 0; i < height; i++) {
			for (int j = 0; j < width; j++) {
				image_wavelet.setRGB(j, i, wltpix[counter]);
				counter++;
			}
		}
		return image_wavelet;
	}

	public static int[][]
convertTo2DWithoutUsingGetRGB(BufferedImage image) {

		final byte[] pixels = ((DataBufferByte) image.getRaster()
				.getDataBuffer()).getData();
		final int width = image.getWidth();
		final int height = image.getHeight();
		final boolean hasAlphaChannel = image.getAlphaRaster()
!= null;

		int[][] result = new int[height][width];
		if (hasAlphaChannel) {
			final int pixelLength = 4;
			for (int pixel = 0, row = 0, col = 0; pixel <
pixels.length; pixel += pixelLength) {
				int argb = 0;
				argb += ((pixels[pixel] & 0xff) << 24); // alpha
				argb += (pixels[pixel + 1] & 0xff); // blue
				argb += ((pixels[pixel + 2] & 0xff) << 8); //
green
				argb += ((pixels[pixel + 3] & 0xff) << 16); // red
				result[row][col] = argb;
				col++;
				if (col == width) {
					col = 0;
					row++;
				}
			}
		} else {
			final int pixelLength = 3;
			for (int pixel = 0, row = 0, col = 0; pixel <
pixels.length; pixel += pixelLength) {
				int argb = 0;
				argb += -16777216; // 255 alpha
				argb += (pixels[pixel] & 0xff); // blue
				argb += ((pixels[pixel + 1] & 0xff) << 8); //
green
				argb += ((pixels[pixel + 2] & 0xff) << 16); // red
				result[row][col] = argb;
				col++;
				if (col == width) {
					col = 0;
					row++;
				}
			}
		}
		return result;
	}
}


/**
 * @author jhesed tacadena
 */

package util;

import java.io.BufferedReader;

import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Random;

public class ValidationFileAutoArranger {

	private final static String url =
"C:\\Users\\jhesed\\Desktop\\DEFENSE\\data\\1st random\\";
	private static ArrayList<String> malignant = new
ArrayList<String>();
	private static ArrayList<String> non_malignant = new
ArrayList<String>();

	public static void separateNodules() {
		try {
			BufferedReader myReader = new
BufferedReader(new FileReader(url
					+ "DATA.data"));
			String nextLine = myReader.readLine();
			while (nextLine != null) {
				if (nextLine.substring(0, 2).equals("1 ")) {
					malignant.add(nextLine);
				} else {
					non_malignant.add(nextLine);
				}
				nextLine = myReader.readLine();
			}
			myReader.close();
			BufferedWriter outmalignant = new
BufferedWriter(new FileWriter(url
					+ "MALIGNANT.LNDCT"));

			for (int i = 0; i < malignant.size(); i++) {
				outmalignant.write(malignant.get(i) + "\n");
			}

			BufferedWriter outnonmalignant = new
BufferedWriter(new FileWriter(
					url + "NONMALIGNANT.LNDCT"));

			for (int i = 0; i < non_malignant.size(); i++) {
				outnonmalignant.write(non_malignant.get(i) +
"\n");
			}
			outmalignant.close();
			outnonmalignant.close();
		} catch (IOException e) {
			e.printStackTrace();
		}

	}

	public static void randomizeMalignantRealSampleSize() {
		try {
			BufferedReader myReader = new
BufferedReader(new FileReader(url
					+ "MALIGNANT.LNDCT"));

			int total_malignant = 0;
			String nextLine = myReader.readLine();
			while (nextLine != null) {
				malignant.add(nextLine);
				total_malignant++;
				nextLine = myReader.readLine();
			}
			myReader.close();
```

```
                ArrayList<Integer> training_malignant = new
ArrayList<Integer>();
                int stopper_training_malignant = (int) Math
                        .floor(total_malignant * .70);

                Random r = new Random();

                while (stopper_training_malignant > 0) {
                    int random = r.nextInt(total_malignant - 1) + 1;
                    if (!training_malignant.contains(random)) {
                        training_malignant.add(random);
                        stopper_training_malignant--;
                    }
                }

                BufferedWriter file_70_malignant = new
BufferedWriter(
                        new FileWriter(url + "TRAINING
malignant.LNDCT"));

                for (int i = 0; i < training_malignant.size(); i++) {
                    file_70_malignant

    .write(malignant.get(training_malignant.get(i)) + "\n");
                }
                file_70_malignant.close();

                BufferedWriter file_30_malignant = new
BufferedWriter(
                        new FileWriter(url + "TESTING
malignant.LNDCT"));

                for (int i = 0; i < malignant.size(); i++) {
                    if (!training_malignant.contains(i)) {
                        file_30_malignant.write(malignant.get(i) +
"\n");
                    }
                }
                file_30_malignant.close();
            } catch (Exception e) {

            }
    }

    public static void randomizeNonMalignantRealSampleSize() {
            try {
                BufferedReader myReader = new
BufferedReader(new FileReader(url
                        + "NONMALIGNANT.LNDCT"));

                int total_malignant = 0;
                String nextLine = myReader.readLine();
                while (nextLine != null) {
                    malignant.add(nextLine);
                    total_malignant++;
                    nextLine = myReader.readLine();
                }
                myReader.close();
                ArrayList<Integer> training_malignant = new
ArrayList<Integer>();
                int stopper_training_malignant = (int) Math
                        .floor(total_malignant * .70);

                Random r = new Random();

                while (stopper_training_malignant > 0) {
                    int random = r.nextInt(total_malignant - 1) + 1;
                    if (!training_malignant.contains(random)) {
                        training_malignant.add(random);
                        stopper_training_malignant--;
                    }
                }

                BufferedWriter file_70_malignant = new
BufferedWriter(
                        new FileWriter(url + "TRAINING non
malignant1.LNDCT"));

                for (int i = 0; i < training_malignant.size(); i++) {
                    file_70_malignant

    .write(malignant.get(training_malignant.get(i)) + "\n");
                }
                file_70_malignant.close();

                BufferedWriter file_30_malignant = new
BufferedWriter(
                        new FileWriter(url + "TESTING non
malignant1.LNDCT"));

                for (int i = 0; i < malignant.size(); i++) {
                    if (!training_malignant.contains(i)) {
                        file_30_malignant.write(malignant.get(i) +
"\n");
                    }
                }
                file_30_malignant.close();
            } catch (Exception e) {
            }
    }
}


/**
 * @author jhesed tacadena
 */

package wavelet;

import java.awt.image.BufferedImage;
import java.util.concurrent.Callable;
import morph.FloodFiller;
import morph.MorphoProcessor;
import morph.StructureElement;
import util.ImageUtil;

public class Wavelet implements Callable<Wavelet> {

    final static int WHITE = -1;
    final static int BLACK = -16777216;
    private BufferedImage wltimage;
    private int scale;
    private int wavelet_type;
    private WaveletEdge wtip;
    private int[] wt_smooth;
    private int[] wt_smooth2;
    private int[] wavelet_nodule;
    private int[][] wt_segmented;
    private int[][] binarized;
    final static int GREEN = -16711936;

    private int height;
    private int width;
    private BufferedImage original;

    public Wavelet(int scale, int WAVELET_TYPE,
BufferedImage original,
                int[][] binarized) {
            this.original = original;
            this.scale = scale;
            this.wavelet_type = WAVELET_TYPE;
            this.wltimage = null;
            this.height = original.getHeight();
            this.width = original.getWidth();
            this.binarized = binarized;
            wavelet_nodule = new int[width * height];
      }

      @Override
      public Wavelet call() throws Exception {
            this.wtip = new WaveletEdge(original, this.width,
this.height,
                        this.wavelet_type, this.scale);
            this.wltimage =
ImageUtil.getImageFromArray(wtip.wltpix, this.width,
                        this.height);
            wt_segmented = new int[width][height];

            wl_show();
            edge_show();

            if (this.wavelet_type == 0) {
                  trimEdges();
                  wtSegmentation();
                  extractEdgeNodules();
            }

            return this;
      }

      public BufferedImage getEdge() {
            if (wavelet_type == 0) {
                  return
ImageUtil.getImageFromArray(this.wt_smooth, width, height);
            }
            if (wavelet_type == 4) {
                  return
ImageUtil.getWhiteImageFromArray(wtip.edge_pix, width,
                              height);
            } else {
                  return ImageUtil.getImageFromArray(wtip.edge_pix,
width, height);
            }
      }

      public int[] getEdgeArray() {
            return wtip.edge_pix;
      }

      public BufferedImage getEdgeTemp() {
            return ImageUtil.getImageFromArray(wtip.edge_temp,
width, height);
      }

      public void wl_show() {
            this.wtip.WT_next();
            this.wltimage =
ImageUtil.getImageFromArray(wtip.wltpix, this.width,
                        this.height);
      }

      public void edge_show() {
            this.wtip.get_edge();
      }

      public BufferedImage getWltImage() {
            return this.wltimage;
      }

      private void trimEdges() {
            int[] in = { 0, 0 };
            MorphoProcessor m = new MorphoProcessor(new
StructureElement(0, 1, 3,
                        in), -1);
            this.wt_smooth = m.smoothWTErode(wtip.edge_pix,
width, height);

            m = new MorphoProcessor(new StructureElement(0, 1, 2,
in), -1);
            this.wt_smooth2 = m.smoothWTErode(wtip.edge_pix,
width, height);
      }

      private void wtSegmentation() {
            int offset;
            for (int i = 0; i < height; i++) {
                  offset = i * height;
                  for (int j = 0; j < width; j++) {
                        wt_segmented[j][i] = wt_smooth2[offset + j];
                  }
            }

      }

      private void extractEdgeNodules() {
            for (int i = 0; i < height; i++) {
                  int offset = i * height;
                  for (int j = 0; j < width; j++) {
                        if (binarized[j][i] == WHITE &&
wt_smooth2[offset + j] == WHITE) {
                              binarized[j][i] = BLACK;
                        }
                  }
            }
            int[] mask = findMask(binarized);
            FloodFiller f = new FloodFiller(binarized, width, height);
            f.fill(mask[0], mask[1], BLACK);

            for (int i = 0; i < height; i++) {
                  int offset = i * height;
                  for (int j = 0; j < width; j++) {
                        wavelet_nodule[offset + j] = binarized[j][i];
                  }
            }

            int[] in = { 0, 0 };
            MorphoProcessor m = new MorphoProcessor(new
StructureElement(0, 1, 4,
                        in), -1);
            wavelet_nodule = m.smoothWTErode(wavelet_nodule,
width, height);

            for (int i = 0; i < height; i++) {
                  int offset = i * height;
                  for (int j = 0; j < width; j++) {
                        if (wavelet_nodule[offset + j] == WHITE) {
                              f.fill(j, i, GREEN);
                        }
                  }
            }

            for (int i = 0; i < height; i++) {
                  int offset = i * height;
                  for (int j = 0; j < width; j++) {
                        if (binarized[j][i] == GREEN) {
                              wavelet_nodule[offset + j] = WHITE;
                        }
                  }
            }
```

```
		m = new MorphoProcessor(new StructureElement(0, 1, 6,
in), -1);
		wavelet_nodule = m.smoothWtDilate(wavelet_nodule,
width, height);
	}

	public int[] getWaveletNodule() {
		return this.wavelet_nodule;
	}

	public int[] findMask(int[][] image) {
		int[] mask = new int[2];

		find: for (int y = 0; y < height; y++) {
			for (int x = 0; x < width; x++) {
				mask: for (int i = y; i < (y + 30); i++) {
					if (image[x][i] != -1) {
						break mask;
					}
					if (i == (y + 29)) {
						mask[0] = x;
						mask[1] = y;
						break find;
					}
				}
			}
		}
		return mask;
	}

	public BufferedImage getEdgeNodules() {
		return
ImageUtil.arrayToColoredBufferedImage(binarized, width,
height);
	}

	public BufferedImage getWTSegmentedImage() {
		return
ImageUtil.arrayToColoredBufferedImage(wt_segmented, width,
				height);
	}
}


package wavelet;

import java.awt.Image;
import java.awt.image.PixelGrabber;

public class ImagePic {

	protected int[] pix = null;
	protected int height;
	protected int width;
	protected int pixelnum;

	public ImagePic(int width, int height) {
		this.width = width;
		this.height = height;
		this.pixelnum = (this.width * this.height);
	}

	public void getPixel(Image paramImage) {
		this.pix = new int[this.pixelnum];
		PixelGrabber localPixelGrabber = new
PixelGrabber(paramImage, 0, 0,
				this.width, this.height, this.pix, 0, this.width);

		try {
			localPixelGrabber.grabPixels(3000L);
		} catch (InterruptedException localInterruptedException)
{
		}

		for (int i = 0; i < this.pixelnum; i++) {
			this.pix[i] = ((int) (0.298912D * (0xFF & this.pix[i]
>> 16)
					+ 0.586611D * (0xFF & this.pix[i] >> 8) +
0.114478D * (0xFF & this.pix[i])));
		}
	}
}


package wavelet;

import java.awt.Image;

public class WTImagePic extends ImagePic {

	private final int QUADRATIC_SPLINE = 0;
	private final int DAUBECHIES = 2;
	private final int DAUBECHIES5 = 3;
	private final int HAAR = 4;
	public int[] wltpix = null;
	public int[] edge_temp = null;
	public int[] image_magnitude = null;
	public int[] image_angular = null;
	protected double[] wltcoef = null;
	public double[] wltcoeftmp = null;
	protected double max;
	protected double min;
	protected int wltstep;
	protected int N;
	protected double[] alpha;
	protected double[] beta;
	boolean shouldRescale;
	protected int scale;
	private int scale_counter;

	public WTImagePic(Image image, int width, int height, int
WAVELET_TYPE,
			int scale) {
		super(width, height);
		this.wltstep = 0;
		this.scale = scale;
		wltclear();
		rescalePlan(true);
		setupWavelet(WAVELET_TYPE);
	}

	public void setupWavelet(int WAVELET_TYPE) {
		if (WAVELET_TYPE == this.QUADRATIC_SPLINE) {
			setupQuadraticSpline();
		} else if (WAVELET_TYPE == this.HAAR) {
			setupHaar();
		} else if (WAVELET_TYPE == this.DAUBECHIES) {
			setupDB();
		} else if (WAVELET_TYPE == this.DAUBECHIES5) {
			setupDB5();
		}
	}

	public void rescalePlan(boolean paramBoolean) {
		this.shouldRescale = paramBoolean;
	}

	@Override
	public void getPixel(Image paramImage) {
		super.getPixel(paramImage);
```

```
        if (this.wltcoeftmp == null) {
            this.wltcoeftmp = new double[this.pixelnum];
        }
        if (this.wltpix == null) {
            this.wltpix = new int[this.pixelnum];
        }
        if (this.wltcoef == null) {
            this.wltcoef = new double[this.pixelnum];
        }
        if (this.image_magnitude == null) {
            this.image_magnitude = new int[this.pixelnum /
this.scale / 2];
        }
        if (this.image_angular == null) {
            this.image_angular = new int[this.pixelnum /
this.scale / 2];
        }
        if (this.edge_temp == null) {
            this.edge_temp = new int[this.pixelnum];
        }

        for (int i = 0; i < this.pixelnum; i++) {
            this.wltcoeftmp[i] = this.pix[i];
        }
    }

    public void setupHaar() {
        this.alpha = (this.beta = null);
        this.N = 2;
        this.alpha = new double[this.N];
        this.beta = new double[this.N];
        double tmp55_54 = (this.beta[0] =
0.707106782373095D);
        this.alpha[1] = tmp55_54;
        this.alpha[0] = tmp55_54;
        this.beta[1] = (-this.beta[0]);
    }

    public void setupQuadraticSpline() {
        this.N = 6;
        this.alpha = new double[this.N];
        this.beta = new double[this.N];
        alpha[0] = 0.0;
        alpha[1] = 0.3535533906;
        alpha[2] = 1.060660172;
        alpha[3] = 1.060660172;
        alpha[4] = 0.3535533906;
        alpha[5] = 0.0;

        beta[0] = -0.7071067812;
        beta[1] = -0.7071067812;
        beta[2] = -0.7071067812;
        beta[3] = 0.7071067812;
        beta[4] = 0.7071067812;
        beta[5] = 0.7071067812;
    }

    public void setupDB() {
        this.alpha = (this.beta = null);

        this.N = 6;
        this.alpha = new double[this.N];
        this.beta = new double[this.N];

        this.alpha[0] = 0.33267055295D;
        this.alpha[1] = 0.806891509311D;
        this.alpha[2] = 0.459877502118D;
        this.alpha[3] = -0.13501102001D;
        this.alpha[4] = -0.08544127388300001D;
        this.alpha[5] = 0.035226291886D;

        for (int i = 0; i < this.N; i++) {
            this.beta[i] = (Math.pow(-1, i) * this.alpha[(this.N - 1
- i)]);
        }
    }

    public void setupDB5() {
        this.alpha = (this.beta = null);

        this.N = 10;
        this.alpha = new double[this.N];
        this.beta = new double[this.N];

        this.alpha[0] = 0.160102397974D;
        this.alpha[1] = 0.603829269797D;
        this.alpha[2] = 0.724308528438D;
        this.alpha[3] = 0.138428145901D;
        this.alpha[4] = -0.242294887066D;
        this.alpha[5] = -0.032244869585D;
        this.alpha[6] = 0.07757149384000001D;
        this.alpha[7] = -0.006241490213D;
        this.alpha[8] = -0.012580751999D;
        this.alpha[9] = 0.003335725285D;

        for (int i = 0; i < this.N; i++) {
            this.beta[i] = (Math.pow(-1, i) * this.alpha[(this.N - 1
- i)]);
        }
    }

    public double pixval(int paramInt1, int paramInt2, int
paramInt3,
            int paramInt4) {
        if ((paramInt1 < paramInt3) && (paramInt2 <
paramInt4)) {
            return this.wltcoeftmp[(paramInt2 * this.width +
paramInt1)];
        }
        if ((paramInt1 >= paramInt3) && (paramInt2 <
paramInt4)) {
            return this.wltcoeftmp[(paramInt2 * this.width +
paramInt3 - 1)];
        }
        if ((paramInt1 < paramInt3) && (paramInt2 >=
paramInt4)) {
            return this.wltcoeftmp[((paramInt4 - 1) * this.width +
paramInt1)];
        }
        return this.wltcoeftmp[((paramInt4 - 1) * this.width +
paramInt3 - 1)];
    }

    public void wavelet(int width_over_wltstep, int
height_over_wltstep) {
        int image_size = width_over_wltstep / 2 *
(height_over_wltstep / 2);
        int[] x_position = new int[4];
        int[] y_position = new int[4];

        x_position[0] = 0;
        y_position[0] = 0;

        x_position[1] = (width_over_wltstep / 2);
        y_position[1] = 0;

        x_position[2] = 0;
        y_position[2] = (height_over_wltstep / 2);
        x_position[3] = (width_over_wltstep / 2);
        y_position[3] = (height_over_wltstep / 2);

        for (int quadrant = 0; quadrant < 4; quadrant++) {
            for (int k = 0; k < image_size; k++) {

                int m = k % (width_over_wltstep / 2);
                int n = k / (width_over_wltstep / 2);

                double d = 0.0D;

                double[] wavelet_array1;
                double[] wavelet_array2;

                if (quadrant == 0) {
                    wavelet_array1 = this.alpha;
                    wavelet_array2 = this.alpha;
                } else if (quadrant == 1) {
                    wavelet_array1 = this.alpha;
                    wavelet_array2 = this.beta;
                } else if (quadrant == 2) {
                    wavelet_array1 = this.beta;
                    wavelet_array2 = this.alpha;
                } else {
                    wavelet_array1 = this.beta;
                    wavelet_array2 = this.beta;
                }

                for (int i = 0; i < this.N; i++) {
                    for (int j = 0; j < this.N; j++) {
                        d += wavelet_array1[i]
                                * wavelet_array2[j]
                                * pixval(2 * m + i, 2 * n + j,
                                        width_over_wltstep,
height_over_wltstep);
                    }
                }

                this.wltcoef[((n + y_position[quadrant]) *
this.width + m + x_position[quadrant])] = d;

                if (quadrant != 0) {
                    if (d > this.max) {
                        this.max = d;
                    }
                    if (d < this.min) {
                        this.min = d;
                    }
                }
            }
        }

        if (width_over_wltstep % 2 == 1) {
            for (int j = 0; j < height_over_wltstep; j++) {
                this.wltcoef[(j * this.width + width_over_wltstep
- 1)] = 0.0D;
            }
        }

        if (height_over_wltstep % 2 == 1) {
            for (int j = 0; j < width_over_wltstep; j++) {
                this.wltcoef[((height_over_wltstep - 1) *
this.width + j)] = 0.0D;
            }
        }
    }

    public void WT_next() {
        for (this.scale_counter = 0; this.scale_counter < this.scale;
this.scale_counter++) {
            if (this.wltstep == 0) {
                this.wltstep = 1;
            }

            if ((this.width / (2 * this.wltstep) < 1)
                    || (this.height / (2 * this.wltstep) < 1)) {
                return;
            }

            wavelet(this.width / this.wltstep, this.height /
this.wltstep);

            for (int i = 1; i <= this.wltstep; i *= 2) {
                if ((!this.shouldRescale) || (i == this.wltstep)) {

                    int computed_image_size = this.width / i
                            * (this.height / i);
                    for (int pixel_counter = 0; pixel_counter <
computed_image_size; pixel_counter++) {

                        int m = pixel_counter % (this.width / i);
                        int n = pixel_counter / (this.width / i);

                        int area = which_area(m, n, this.width /
i, this.height
                                / i);

                        int i2 = (int) this.wltcoef[(this.width * n
+ m)];
                        int i3 = this.width * n + m;

                        if (area != 0) {
                            i2 /= 2 * i;

                            if (i2 > 255) {
                                this.wltpix[i3] = 255;
                            } else if (i2 < 0) {
                                this.wltpix[i3] = 0;
                            } else {
                                this.wltpix[i3] = i2;
                            }

                        } else {
                            i2 /= 2 * i;
                            if (i2 > 255) {
                                this.wltpix[i3] = 255;
                            } else if (i2 < 0) {
                                this.wltpix[i3] = 0;
                            } else {
                                this.wltpix[i3] = i2;
                            }
                        }

                        this.edge_temp[i3] = wltpix[i3];
                        this.wltpix[i3] = (0xFF000000 |
this.wltpix[i3] << 16
                                | this.wltpix[i3] << 8 |
this.wltpix[i3]);
                        this.wltcoeftmp[i3] = this.wltcoef[i3];
                    }
                }
            }
            this.wltstep *= 2;
        }
    }

    public int which_area(int paramInt1, int paramInt2, int
paramInt3,
            int paramInt4) {
```

```
            if ((paramInt1 < paramInt3 / 2) && (paramInt2 <
paramInt4 / 2)) {
                return 0;
            }
            if ((paramInt1 >= paramInt3 / 2) && (paramInt2 <
paramInt4 / 2)) {
                return 1;
            }
            if ((paramInt1 < paramInt3 / 2) && (paramInt2 >=
paramInt4 / 2)) {
                return 2;
            }
            return 3;
        }

        public void wltclear() {
            setPixNull(this.wltpix);
            setcoefzero(this.wltcoef);
            resetwltstep();
            resetmaxmin();
        }

        public void setPixNull(int[] paramArrayOfInt) {
            if (paramArrayOfInt == null) {
                paramArrayOfInt = new int[this.pixelnum];
            }

            for (int i = 0; i < this.pixelnum; i++) {
                paramArrayOfInt[i] = 255;
                paramArrayOfInt[i] = (0xFF000000 |
paramArrayOfInt[i] << 16
                        | paramArrayOfInt[i] << 8 |
paramArrayOfInt[i]);
            }
        }

        public void setcoefzero(double[] paramArrayOfDouble) {
            if (paramArrayOfDouble == null) {
                paramArrayOfDouble = new double[this.pixelnum];
            }

            for (int i = 0; i < this.pixelnum; i++) {
                paramArrayOfDouble[i] = 0.0D;
            }
        }

        public void resetwltstep() {
            this.wltstep = 0;
        }

        public void resetmaxmin() {
            this.max = -127.0D;
            this.min = 127.0D;
        }
}


package wavelet;

import java.awt.Image;

class WaveletEdge extends WTImagePic {

        protected double[] edgecoef;
        public int[] edge_pix;
        boolean getInverse = false;

        public WaveletEdge(Image image, int width, int height, int
WAVELET_TYPE,
                int scale) {
            super(image, width, height, WAVELET_TYPE, scale);
            getPixel(image);
        }

        @Override
        public void getPixel(Image paramImage) {
            super.getPixel(paramImage);

            if (this.edgecoef == null) {
                this.edgecoef = new double[this.pixelnum * 2];
            }

            if (this.edge_pix == null) {
                this.edge_pix = new int[this.pixelnum];
            }

            for (int i = 0; i < this.pixelnum; i++)
                this.edge_pix[i] = -1;
        }

        @Override
        public void wltclear() {
            super.wltclear();
            setPixNull(this.edge_pix);
            resetwltstep();
        }

        @Override
        public double pixval(int paramInt1, int paramInt2, int
paramInt3,
                int paramInt4) {
            int i = paramInt1;
            while (i >= paramInt3)
                i -= paramInt3;

            int j = paramInt2;
            while (j >= paramInt4)
                j -= paramInt4;

            return this.wltcoeftmp[(j * this.width + i)];
        }

        public double wltval(int paramInt1, int paramInt2, int
paramInt3,
                int paramInt4, int paramInt5, int paramInt6) {
            int i = paramInt1;
            while (i < 0)
                i += paramInt3;

            int j = paramInt2;
            while (j < 0)
                j += paramInt4;

            return this.wltcoef[((j + paramInt6) * this.width + i +
paramInt5)];
        }

        public double edgeval(int paramInt1, int paramInt2, int
paramInt3,
                int paramInt4) {
            int i = paramInt1;
            while (i < 0)
                i += paramInt3;

            int j = paramInt2;
            while (j < 0)
                j += paramInt4;

            return this.edgecoef[(j * 2 * this.width + i + paramInt3)];
        }
```

```
    @Override
    public void WT_next() {
        if (this.wltstep == 0) {
            this.wltstep = 1;
        }

        if ((this.width / (2 * this.wltstep) < 1)
                || (this.height / (2 * this.wltstep) < 1)) {
            return;
        }

        super.WT_next();
        this.wltstep /= 2;
        prepare_edge();
        this.wltstep *= 2;
    }

    @Override
    public void resetwltstep() {
        this.wltstep = 0;
    }

    public void prepare_edge() {
        int i = this.width / this.wltstep;
        int j = this.height / this.wltstep;

        int k = i / 2;
        int m = j / 2;
        double d = 0.0D;

        for (int n = 0; n < i; n++)
            for (int y = 0; y < j; y++) {
                d = 0.0D;

                for (int x = 0; x < this.N; x++) {
                    for (int i3 = 0; i3 < this.N; i3++) {
                        if ((isEven(n - x)) && (isEven(y - i3)))
{
                            d += this.alpha[x]
                                    * this.beta[i3]
                                    * wltval((n - x) / 2, (y - i3)
/ 2, k, m,
                                            k, 0)
                                    + this.beta[x]
                                    * this.alpha[i3]
                                    * wltval((n - x) / 2, (y - i3)
/ 2, k, m,
                                            0, m)
                                    + this.beta[x]
                                    * this.beta[i3]
                                    * wltval((n - x) / 2, (y - i3)
/ 2, k, m,
                                            k, m);
                        }

                    }

                }
                this.edgecoef[(y * 2 * this.width + (n + i))] = d;
            }
    }

    public boolean isEven(int paramInt) {
        if (paramInt % 2 == 0) {
            return true;
        }
        return false;
    }
```

```
    public void InversePlan(boolean paramBoolean) {
        this.getInverse = paramBoolean;
    }

    public void get_edge() {
        if (this.wltstep == 0)
            return;

        int i;
        int j;
        int k;
        int m;
        double d;

        for (int n = this.wltstep / 2; n > 1; n /= 2) {
            i = 2 * this.width / n;
            j = 2 * this.height / n;
            k = i / 2;
            m = j / 2;

            for (int y = 0; y < i; y++) {
                for (int x = 0; x < j; x++) {
                    d = 0.0D;

                    for (int i3 = 0; i3 < this.N; i3++) {
                        for (int i4 = 0; i4 < this.N; i4++) {
                            if ((isEven(y - i3)) && (isEven(x -
i4))) {
                                d += this.alpha[i3]
                                        * this.alpha[i4]
                                        * edgeval((y - i3) / 2,
(x - i4) / 2,
                                                k, m);
                            }
                        }
                    }
                    this.edgecoef[(x * 2 * this.width + (y + i))]
+= d;
                }
            }
        }

        i = this.width;
        j = this.height;

        int x;
        for (int n = 0; n < i; n++)
            for (int y = 0; y < j; y++) {
                x = (int) this.edgecoef[(y * 2 * this.width + (n +
i))];

                if (x > 255)
                    x = 255;
                else if (x < 0) {
                    x = 0;
                }

                this.edge_pix[(this.width * y + n)] = x;

                this.edge_pix[(this.width * y + n)] =
(0xFF000000
                        | this.edge_pix[(this.width * y + n)] <<
16
                        | this.edge_pix[(this.width * y + n)] <<
8 | this.edge_pix[(this.width
                        * y + n)]);
            }
    }
}
```

```
package diffusion;

import java.util.concurrent.Callable;

import morph.OtsuBinarize;

public class Diffusion implements Callable<int[]> {

    final static int HMIN = 0;
    final static int HMAX = 256;
    private int ITERATIONS;
    private int K;
    private float lambda;
    private boolean bigRegionFunction;
    private int width;
    private int height;
    private int[] pixels;

    public Diffusion(int iterations, int k, float lambda,
            int[] pixels, int width, int height) {
        this.pixels = pixels;
        this.width = width;
        this.height = height;
        this.ITERATIONS = iterations;
        this.K = k;
        this.lambda = lambda;
        this.bigRegionFunction = true;
    }

    @Override
    public int[] call() throws Exception {
        DiffusionStructure DiffusionStructure = new
DiffusionStructure(
                pixels, width, height);
        DiffusionPixel.K = K;
        DiffusionPixel.lambda = lambda;

        for (int i = 0; i < ITERATIONS; i++) {
            DiffusionStructure.iterate(bigRegionFunction);
        }

        int[] new_pixels = new int[width * height];

        for (int pixelIndex = 0; pixelIndex < DiffusionStructure
                .size(); pixelIndex++) {
            DiffusionPixel p = DiffusionStructure
                    .get(pixelIndex);
            new_pixels[p.getX() + p.getY() * width] =
p.getIntHeight();
            new_pixels[p.getX() + p.getY() * width] =
(0xFF000000
                    | new_pixels[p.getX() + p.getY() * width]
<< 16
                    | new_pixels[p.getX() + p.getY() * width]
<< 8 | new_pixels[p
                    .getX() + p.getY() * width]);

            new_pixels[p.getX() + p.getY() * width] = ((int)
(0.298912D
                    * (0xFF & new_pixels[p.getX() + p.getY() *
width] >> 16)
                    + 0.586611D
                    * (0xFF & new_pixels[p.getX() + p.getY() *
width] >> 8) + 0.114478D * (0xFF & new_pixels[p
                    .getX() + p.getY() * width])));
        }
        OtsuBinarize bin = new OtsuBinarize();
        int[][] b = bin.binarize(new_pixels, width, height, -1);
        for (int i = 0; i < height; i++) {
            int offset = i * height;
            for (int j = 0; j < width; j++) {
                new_pixels[offset + j] = b[j][i];
            }
        }
        return new_pixels;
    }
}


/**
 * Original author:
 * Copyright (c) 2003 by Christopher Mei
(christopher.mei@sophia.inria.fr)
 *
 * This plugin is free software; you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License version 2
 * as published by the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied
warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public
License
 * along with this plugin; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 *
 * Modified by: Jhesed D. Tacadena
 */

package diffusion;

import java.util.*;

import diffusion.DiffusionPixel;

public class DiffusionStructure {

    @SuppressWarnings("rawtypes")
    private Vector DiffusionStructure;

    @SuppressWarnings({ "rawtypes", "unchecked" })
    public DiffusionStructure(int[] pixels, int width, int height) {

        DiffusionStructure = new Vector(width * height);
        int offset, topOffset, bottomOffset, i;

        for (int y = 0; y < height; y++) {
            offset = y * width;
            for (int x = 0; x < width; x++) {
                i = offset + x;
                DiffusionStructure
                        .add(new DiffusionPixel(x, y,
pixels[i]));
            }
        }

        for (int y = 0; y < height; y++) {

            offset = y * width;
            topOffset = offset + width;
            bottomOffset = offset - width;
            for (int x = 0; x < width; x++) {
                DiffusionPixel currentPixel = (DiffusionPixel)
DiffusionStructure
                        .get(x + offset);

                if (x + 1 < width) {
                    currentPixel

    .addDirectNeighbour((DiffusionPixel) DiffusionStructure
                                    .get(x + 1 + offset));
                }

                if (x - 1 >= 0) {
                    currentPixel

    .addDirectNeighbour((DiffusionPixel) DiffusionStructure
                                    .get(x - 1 + offset));
                }

                if (y - 1 >= 0) {
                    currentPixel

    .addDirectNeighbour((DiffusionPixel) DiffusionStructure
                                    .get(x + bottomOffset));
                }

                if (y + 1 < height) {
                    currentPixel

    .addDirectNeighbour((DiffusionPixel) DiffusionStructure
                                    .get(x + topOffset));
                }
            }
        }
    }

    public int size() {
        return DiffusionStructure.size();
    }

    public final DiffusionPixel get(int i) {
        return (DiffusionPixel) DiffusionStructure.get(i);
    }

    public void iterate(boolean bigRegionFunction) {

        for (int i = 0; i < DiffusionStructure.size(); i++) {
            get(i).setGradient();
            get(i).setNewHeight(bigRegionFunction);
            get(i).switchHeightValues();
        }
    }
}


/**
 * Original author:
 * Copyright (c) 2003 by Christopher Mei
(christopher.mei@sophia.inria.fr)
 *
 * This plugin is free software; you can redistribute it and/or
modify
 * it under the terms of the GNU General Public License version 2
 * as published by the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied
warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public
License
 * along with this plugin; if not, write to the Free Software
 * Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 *
 * Modified by: Jhesed D. Tacadena
 */

package diffusion;

import java.util.*;

public class DiffusionPixel {

    private int x;
    private int y;
    private int currentHeight;
    private int newHeight;
    @SuppressWarnings("rawtypes")
    private Vector directNeighbours; // N, S, E, W neighbor
    public static double K;
    public static double lambda;

    @SuppressWarnings("rawtypes")
    public DiffusionPixel(int x, int y, int height) {
        this.x = x;
        this.y = y;
        this.currentHeight = height;
        directNeighbours = new Vector(4);
        newHeight = 0;
    }

    @SuppressWarnings("unchecked")
    public void addDirectNeighbour(DiffusionPixel neighbour) {
        directNeighbours.add(new Neighbour(neighbour, 0));
    }

    public void setNewHeight(boolean bigRegionFunction) {
        double sum = 0;
        for (int i = 0; i < directNeighbours.size(); i++) {
            Neighbour neighbour = (Neighbour)
directNeighbours.get(i);
            sum += DiffusionPixel.g(neighbour.gradient,
                    bigRegionFunction) * neighbour.gradient;
        }
        newHeight = (int) (currentHeight + lambda * sum);
    }

    public void switchHeightValues() {
        currentHeight = newHeight;
    }

    public void setGradient() {
        for (int i = 0; i < directNeighbours.size(); i++) {
            Neighbour neighbour = (Neighbour)
directNeighbours.get(i);
            neighbour.gradient =
neighbour.neighbour.getIntHeight()
                    - getIntHeight();
        }
    }

    public final static double g(int gradient, boolean
bigRegionFunction) {
        if (bigRegionFunction) {
            return 1 / (1 + Math.pow(gradient / DiffusionPixel.K,
2));
        } else {
            return Math.exp(-Math.pow((gradient) /
DiffusionPixel.K,
                    2));
        }
    }

    public final int getIntHeight() {
        return currentHeight & 0xff;
    }

    public final int getX() {
        return x;
    }

    public final int getY() {
        return y;
    }
}

class Neighbour {

    public DiffusionPixel neighbour;
    public int gradient;

    public Neighbour(DiffusionPixel neighbour, int gradient) {
        this.neighbour = neighbour;
        this.gradient = gradient;
    }
}



/**
 * @version  2.3.1 13 June 2006
 *           2.3   25 May 2006
 *           2.1;  09 Dec 2004
          2.0;  13 Nov 2004
 *
 * Original @author Dimiter Prodanov
 *          Catholic University of Louvaion; University of Leiden
 *
 *
 * @contents     This plugin performs the basic morphologic
operations on grayscale images
 *    erosion, dilation, opening and closing with several types of
structuring elements.
 *    It is build upon the StructureElement class
 *
 *    The develpoment of this alogorithm was inspired by the book
of Jean Serra
 *    "Image Analysis and Mathematical Morphology"
 *
 * @license     This library is free software; you can redistribute it
and/or
 *    modify it under the terms of the GNU Lesser General Public
 *    License as published by the Free Software Foundation; either
 *    version 2.1 of the License, or (at your option) any later
version.
 *
 *    This library is distributed in the hope that it will be useful,
 *    but WITHOUT ANY WARRANTY; without even the
implied warranty of
 *    MERCHANTABILITY or FITNESS FOR A PARTICULAR
PURPOSE.  See the GNU
 *     Lesser General Public License for more details.
 *
 *  Modified by: Jhesed Tacadena
 */

package morph;

import java.util.concurrent.Callable;

public class GrayMorphology implements Constants,
Callable<GrayMorphology> {

    private static final int[] offset = OFFSET0;
    public StructureElement se, minus_se, plus_se, down_se,
up_se;
    MorphoProcessor mp;
    int type;
    public final static int FDILATE = 1;
    public final static int FERODE = 2;
    public final static int FILL = 3;
    float radius;
    private int width;
    private int height;
    private int threshold;
    private int[] pixels;

    public GrayMorphology(int width, int height, float radius, int
type,
            int threshold, int[] pixels) {
        this.pixels = pixels;
        this.threshold = threshold;
        this.type = type;
        this.radius = radius;
        this.width = width;
        this.height = height;
    }

    @Override
    public GrayMorphology call() throws Exception {
        se = new StructureElement(type, 1, radius, offset);
        mp = new MorphoProcessor(se, threshold);
        doOptions(pixels, mp, FERODE);
        doOptions(pixels, mp, FDILATE);
        return this;
    }

    public int[] Abs(int[] arr) {
        int[] warr = new int[arr.length];
        for (int i = 0; i < arr.length; i++) {
            warr[i] = Math.abs(arr[i]);
        }
        return warr;
    }

    private void doOptions(int[] mask, MorphoProcessor mp, int
morphoptions) {
        switch (morphoptions) {

        case FDILATE: {
            mp.fastDilate2(mask, width, height);
            break;
        }
        case FERODE: {
            mp.fastErode1(mask, width, height);
            break;
        }
        }
    }

    public int[] getPixels() {
        return this.pixels;
    }
}
```

```
package morph;

public class MorphoProcessor implements Constants {
    private StructureElement se, minus_se, plus_se;
    private LocalHistogram bh, p_h, m_h;
    private int[][] pg, pg_plus, pg_minus;
    private final static int ORIG = 0, PLUS = 1, MINUS = -1;
    int w, h;
    int threshold;

    public MorphoProcessor(StructureElement se, int threshold) {
        this.threshold = threshold;
        this.se = se;
        w = se.getWidth();
        h = se.getHeight();
        minus_se = new
StructureElement(se.H(se.Delta(SGRAD), HMINUS), w);
        plus_se = new StructureElement(se.H(se.Delta(NGRAD),
HMINUS), w);
        bh = new LocalHistogram();
        p_h = new LocalHistogram();
        m_h = new LocalHistogram();
        pg = se.getVect();
        pg_plus = plus_se.getVect();
        pg_minus = minus_se.getVect();
    }

    public StructureElement getSE(int options) {
        switch (options) {
        case ORIG: {
            return se;
        }
        case PLUS: {
            return plus_se;
        }
        case MINUS:
            return minus_se;
        }
        return se;
    }

    public int[] findMask(int[][] image, int width, int height) {
        int[] mask = new int[2];
        find: for (int y = 100; y < height; y++) {
            for (int x = 100; x < width; x++) {
                mask: for (int i = y; i < (y + 30); i++) {
                    if (image[x][i] != -1) { // if not white
                        break mask;
                    }
                    if (i == (y + 29)) {
                        mask[0] = x;
                        mask[1] = y;
                        break find;
                    }
                }
            }
        }
        return mask;
    }

    public void fastDilate1(int[] pixels, int width, int height) {
        int max = 32767;
        int[] new_pixel = new int[width * height];
        int row = 0, z = 0;
        int index = 0;

        for (row = 1; row <= height; row++) {
            z = (row - 1) * width;
            bh.initialize(z, width, height, pixels, pg, 0);
            max = bh.getMaximum();
            new_pixel[z] = (max & 0xFF);
            for (int col = 1; col < width; col++) {
                index = z + col;
                try {
                    p_h.initialize(index, width, height, pixels,
pg_plus, 0);
                    m_h.initialize(index - 1, width, height,
pixels, pg_minus,
                            0);
                    bh.sub(m_h);
                    bh.add(p_h);
                    bh.doMaximum();
                    max = bh.getMaximum();
                    new_pixel[index] = (max & 0xFF);
                } catch (ArrayIndexOutOfBoundsException
aiob) {
                }
            }
        }
        System.arraycopy(new_pixel, 0, pixels, 0, pixels.length);
    }

    public void fastDilate2(int[] pixels, int width, int height) {
        regionFillBinary(pixels, width, height);
        int max = 32767;
        int[] new_pixel = new int[width * height];
        int row = 0, z = 0;
        int index = 0;

        for (row = 1; row <= height; row++) {
            z = (row - 1) * width;
            bh.initialize(z, width, height, pixels, pg, 0);
            max = bh.getMaximum();
            new_pixel[z] = (max & 0xFF);
            for (int col = 1; col < width; col++) {
                index = z + col;
                try {
                    p_h.initialize(index, width, height, pixels,
pg_plus, 0);
                    m_h.initialize(index - 1, width, height,
pixels, pg_minus,
                            0);
                    bh.sub(m_h);
                    bh.add(p_h);
                    bh.doMaximum();
                    max = bh.getMaximum();
                    new_pixel[index] = (max & 0xFF);
                    new_pixel[index] = (0xFF000000 |
new_pixel[index] << 16
                            | new_pixel[index] << 8 |
new_pixel[index]);
                } catch (ArrayIndexOutOfBoundsException
aiob) {
                }
            }
        }
        System.arraycopy(new_pixel, 0, pixels, 0, pixels.length);
    }

    public void regionFillBinary(int[] pixels, int width, int height)
{
        int[][] binarized = new int[width][height];

        for (int i = 0; i < height; i++) {
            int offset = i * height;
            for (int j = 0; j < width; j++) {
                binarized[j][i] = pixels[offset + j];
            }
        }
        int[] mask = findMask(binarized, width, height);
        FloodFiller filler = new FloodFiller(binarized, width,
height);
        filler.fill(mask[0], mask[1], -16777216);
        for (int i = 0; i < height; i++) {
            int offset = i * height;
            for (int j = 0; j < width; j++) {
                pixels[offset + j] = binarized[j][i];
            }
        }
    }

    public void regionFillGray(int[] binarized, int[] eroded, int
width,
            int height) {
        for (int i = 0; i < width; i++) {
            int offset = i * height;
            for (int j = 0; j < height; j++) {
                if (eroded[offset + j] != -16777216) {
                    if (binarized[offset + j] == -16777216) {
                        eroded[offset + j] = -16777216;
                    }
                }
            }
        }
    }

    public void fastErode1(int[] pixels, int width, int height) {

        int min = -32767;
        int[] new_pixel = new int[width * height];
        int row = 0, z = 0;
        int index = 0;

        for (row = 1; row <= height; row++) {
            z = (row - 1) * width;
            bh.initialize(z, width, height, pixels, pg, 1);
            min = bh.getMinimum();
            new_pixel[z] = (min & 0xFF);
            for (int col = 1; col < width; col++) {
                index = z + col;
                try {
                    p_h.initialize(index, width, height, pixels,
pg_plus, 1);
                    m_h.initialize(index - 1, width, height,
pixels, pg_minus,
                            1);
                    bh.sub(m_h);
                    bh.add(p_h);
                    bh.doMinimum();
                    min = bh.getMinimum();
                    new_pixel[index] = (min & 0xFF);
                    new_pixel[index] = (0xFF000000 |
new_pixel[index] << 16
                            | new_pixel[index] << 8 |
new_pixel[index]);
                } catch (ArrayIndexOutOfBoundsException
aiob) {
                }
            }
        }
        System.arraycopy(new_pixel, 0, pixels, 0, pixels.length);
    }

    public int[] smoothWTErode(int[] pixels, int width, int height)
{
        int min = -32767;
        int[] new_pixel = new int[width * height];
        int row = 0, z = 0;
        int index = 0;

        for (row = 1; row <= height; row++) {
            z = (row - 1) * width;
            bh.initialize(z, width, height, pixels, pg, 1);
            min = bh.getMinimum();
            new_pixel[z] = (min & 0xFF);
            for (int col = 1; col < width; col++) {
                index = z + col;
                try {
                    p_h.initialize(index, width, height, pixels,
pg_plus, 1);
                    m_h.initialize(index - 1, width, height,
pixels, pg_minus,
                            1);
                    bh.sub(m_h);
                    bh.add(p_h);

                    bh.doMinimum();

                    min = bh.getMinimum();
                    new_pixel[index] = (min & 0xFF);
                    new_pixel[index] = (0xFF000000 |
new_pixel[index] << 16
                            | new_pixel[index] << 8 |
new_pixel[index]);
                } catch (ArrayIndexOutOfBoundsException
aiob) {
                    System.out.println("error erosion");
                }
            }
        }
        return new_pixel;
    }

    public int[] smoothWtDilate(int[] pixels, int width, int height)
{
        int max = 32767;
        int[] new_pixel = new int[width * height];
        int row = 0, z = 0;
        int index = 0;

        for (row = 1; row <= height; row++) {
            z = (row - 1) * width;
            bh.initialize(z, width, height, pixels, pg, 0);
            max = bh.getMaximum();
            new_pixel[z] = (max & 0xFF);
            for (int col = 1; col < width; col++) {
                index = z + col;
                try {
                    p_h.initialize(index, width, height, pixels,
pg_plus, 0);
                    m_h.initialize(index - 1, width, height,
pixels, pg_minus,
                            0);
                    bh.sub(m_h);
                    bh.add(p_h);
                    bh.doMaximum();
                    max = bh.getMaximum();
                    new_pixel[index] = (max & 0xFF);
                    new_pixel[index] = (0xFF000000 |
new_pixel[index] << 16
                            | new_pixel[index] << 8 |
new_pixel[index]);
                } catch (ArrayIndexOutOfBoundsException
aiob) {
                }
            }
        }
        return new_pixel;
    }
}



package morph;

public interface Constants {

    public final static int CIRCLE = 0;
    public final static int[] OFFSET0 = { 0, 0 };
    public final static int[] NGRAD = { 0, 1 };
    public final static int[] SGRAD = { 0, -1 };
    public final static int[] WGRAD = { 1, 0 };
    public final static int[] EGRAD = { -1, 0 };
    public final static int[] NEGRAD = { -1, -1 };
    public final static int[] NWGRAD = { 1, -1 };
    public final static int[] SEGRAD = { -1, -1 };
    public final static int[] SWGRAD = { 1, 1 };
    public final static int MAXSIZE = 151;
    public final static int HPLUS = 1;
    public final static int HMINUS = -1;
    public final static int PERIM = -16;
    public final static int FULLAREA = -32;
    public static final double cor3 = 1.5 - Math.sqrt(3.0);
}


package morph;

import java.util.*;

public class StructureElement implements Constants {

    private int[] mask;
    private int width = 1;
    private int height = 1;
    private double radius = 0;
    private int[][] vect;
    private int[] offset = OFFSET0;
    private int shift = 1;
    public int type = -1;
    public boolean offsetmodified = false;
    final static String EOL =
System.getProperty("line.separator");

    public StructureElement(int[] amask, int width) {
        this.width = width;
        this.height = amask.length / width;
        this.shift = 0;
        setMask(amask);
        vect = calcVect(mask, width);
    }

    public StructureElement(String tokenString) {

        this.shift = 0;
        mask = inputMask(tokenString);
        vect = calcVect(mask, width);
    }

    public StructureElement(int type, int shift, float radius, int[]
offset) {
        this.shift = shift;
        this.radius = radius;
        if ((offset[0] * offset[0] + offset[1] * offset[1]) >= radius /
2) {
            offset = OFFSET0;
        }
        this.offset = offset;
        this.type = type;
        switch (type) {
        case CIRCLE: {
            mask = createCircularMask(shift, radius, offset);
            break;
        }
        }
        vect = calcVect(mask, width);
    }

    private int[] createCircularMask(int shift, double radius, int[]
offset) {
        this.width = ((int) (radius + shift + 0.5)) * 2 + 1;
        this.height = width;
        int[] mask = new int[this.width * this.width];
        double r = width / 2.0 - 0.5;
        double r2 = radius * radius + 1;
        int index = 0;
        for (double x = -r; x <= r; x++) {
            for (double y = -r; y <= r; y++) {
                if ((x - offset[0]) * (x - offset[0]) + (y - offset[1])
                        * (y - offset[1]) < r2) {

                    mask[index] = 255;
                }
                index++;
            }
        }
        return mask;
    }

    private int[] inputMask(String tokenString) {

        StringTokenizer st = new StringTokenizer(tokenString);
        int n = st.countTokens();
        int kw = (int) Math.sqrt(n);
        int kh = kw;
        int sz = kw * kh;
        int[] k = new int[sz];
        for (int i = 0; i < sz; i++) {
            k[i] = (int) getNum(st);
        }
        this.width = kw;
        this.height = width;
        return k;
    }

    public int[] getMask() {
        return mask;
    }

    public int getMaskAt(int index) {
        if (index <= mask.length) {
            return mask[index];
        } else {
            return -1;
        }
    }

    public int getMaskAt(int x, int y) {
        if (x <= 0) {
            x = 0;
        }
        if (x > height) {
            x = height;
        }
        if (y <= 0) {
            y = 0;
        }
        if (y >= width) {
            y = width - 1;
        }

        int index = x + this.width * y;
        return mask[index];
    }

    public void setMask(int[] amask) {
        this.mask = amask;
    }

    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }

    public int getShift() {
        return shift;
    }

    public double getR() {
        return radius;
    }

    public int getType() {
        return type;
    }

    public void setType(int type) {
        this.type = type;
    }

    public int[] getOffset() {
        return this.offset;
    }

    public void setOffset(int[] offset) {
        this.offset = offset;
        offsetmodified = true;
    }

    double getNum(StringTokenizer st) {
        Double d;
        String token = st.nextToken();
        try {
            d = new Double(token);
        } catch (NumberFormatException e) {
            d = null;
        }
        if (d != null) {
            return (d.doubleValue());
        } else {
            return 0.0;
        }
    }

    public int[] T(int[] h) {
        int m, n, index, ind;
        int[] strel = new int[mask.length];

        for (int i = 0; i < height - 1; i++) {
            for (int j = 0; j < width; j++) {
                m = i + h[0]; // y - direction
                n = j + h[1]; // x - direction
                if (m < 0) {
                    m = 0;
                }
                if (n < 0) {
                    n = 0;
                }
                if (n > width) {
                    n = width;
                }
                if (m > (width - 1)) {
                    m = width - 1;
                }
                index = n + width * m;
                ind = j + width * i;
                try {
                    strel[ind] = mask[index];
                } catch (ArrayIndexOutOfBoundsException ex)
{
                }
            }
        }
        return strel;
    }

    public StructureElement Tr(int[] h) {
        return new StructureElement(T(h), this.width);
    }

    public int[] Delta(int[] offset) {
        int[] astrel = this.T(offset);
        for (int index = 0; index < mask.length; index++) {
            astrel[index] = mask[index] - astrel[index];
        }
        return astrel;
    }

    public int[] H(int[] strel, int sign) {
        for (int i = 0; i < strel.length; i++) {
            if (strel[i] * sign >= 0) {
                strel[i] = 0;
            } else {
                strel[i] = 255;
            }
        }
        return strel;
    }

    public int getArea() {
        int maskSize = 0;
        for (int i = 0; i < mask.length; i++) {
            if (mask[i] != 0) {
                maskSize++;
            }
        }
        return maskSize;
    }

    public int[] getBorder() {
        int sz = this.mask.length;
        int[] perim = new int[sz];
        int k, l, m, n = -1;
        int i, j = 0;

        for (int c = 0; c < sz; c++) {
            i = c / width;
            j = c % width;
            if (j + 1 > width - 1) {
                k = 0;
            } else {
                k = mask[i * width + j + 1];
            }
```

```
                if (i + 1 > height - 1) {
                    l = 0;
                } else {
                    l = mask[(i + 1) * width + j];
                }
                if (j == 0) {
                    m = 0;
                } else {
                    m = mask[i * width + j - 1];
                }
                if (i == 0) {
                    n = 0;
                } else {
                    n = mask[(i - 1) * width + j];
                }
                if (mask[c] > k || mask[c] > m || mask[c] > l || mask[c]
> n) {
                    perim[c] = mask[c];
                }
            }

            return perim;
        }

        private int[][] calcVect(int[] perim, int w) {
            int N = 0;
            int sz = perim.length;
            for (int i = 0; i < perim.length; i++) {
                if (perim[i] > 0) {
                    N++;
                }
            }
            int h = sz / w;
            int p = (int) Math.floor(h / 2);
            int q = (int) Math.floor(w / 2);
            int[][] pg = new int[N][4];
            int i, j, counter = 0;

            for (int c = 0; c < sz; c++) {
                i = c / w;
                j = c % w;
                if (perim[c] > 0) {
                    pg[counter][0] = i - p;
                    pg[counter][1] = j - q;
                    pg[counter][2] = perim[c];
                    int[] a = { pg[counter][1], pg[counter][0] };
                    double d = getDistance(a, this.type);
                    pg[counter][3] = (int) Math.round(d);
                    counter++;
                }
            }
            return pg;
        }

        public double getDistance(int[] X, int metrics) {
            double d = 0d;
            double g = 0;
            switch (metrics) {
            case CIRCLE: {
                g = X[0] * X[0] + X[1] * X[1];
                d = Math.sqrt(g + 1) + cor3;
                break;
            }
            }
            return d;
        }

        public int[][] VectTransform(int opt) {
            if (opt == PERIM) {
                return calcVect(getBorder(), this.width);
            } else {
                return calcVect(this.mask, this.width);
            }
        }

        public int[][] getVect() {
            return vect;
        }
}


package morph;

public class OtsuBinarize {

        private final int RGB_BLACK = -16777216;
        private final int RGB_WHITE = -1;
        private int[][] binarized_matrix;
        private int thresh;

        public OtsuBinarize() {
        }

        public int[][] binarize(int[] image, int width, int height, int
threshold) {
            int new_pixel;

            if (threshold == -1) {
                threshold = otsuTreshold(image, width, height);
            }
            thresh = threshold;

            this.binarized_matrix = new int[height][width];

            int i, offset;
            for (int y = 0; y < height; y++) {
                offset = y * height;
                for (int x = 0; x < width; x++) {
                    i = offset + x;

                    if (image[i] > threshold) {
                        new_pixel = RGB_WHITE;
                    } else {
                        new_pixel = RGB_BLACK;
                    }
                    this.binarized_matrix[x][y] = new_pixel; // note
before [y][x]
                }
            }
            return binarized_matrix;
        }

        private static int otsuTreshold(int[] original, int width, int
height) {
            int[] histogram = imageHistogram(original, width,
height);
            float sum = 0;
            for (int i = 0; i < 256; i++) {
                sum += i * histogram[i];
            }

            float sumB = 0;
            int wB = 0;
            int wF = 0;

            float varMax = 0;
            int threshold = 0;

            for (int i = 0; i < 256; i++) {
                wB += histogram[i];
            if (wB == 0) {
                continue;
            }
            wF = (width * height) - wB;

            if (wF == 0) {
                break;
            }

            sumB += i * histogram[i];
            float mB = sumB / wB;
            float mF = (sum - sumB) / wF;

            float varBetween = (float) wB * (float) wF * (mB -
mF) * (mB - mF);

            if (varBetween > varMax) {
                varMax = varBetween;
                threshold = i;
            }
        }
        return threshold;
    }

    public static int[] imageHistogram(int[] input, int width, int
height) {
        int[] histogram = new int[256];
        for (int i = 0; i < histogram.length; i++) {
            histogram[i] = 0;
        }
        int i, offset;
        for (int y = 0; y < height; y++) {
            offset = y * height;
            for (int x = 0; x < width; x++) {
                i = offset + x;
                histogram[input[i]]++;
            }
        }
        return histogram;
    }

    public int getThresh() {
        return this.thresh;
    }
}


package morph;

public class LocalHistogram implements Constants {

    private static final int MAX = 255, MIN = 0;
    private int[] counts = new int[256];
    private int min = MIN, max = MAX;
    private int binCount = 0;

    public LocalHistogram(int[] cnt) {
        this.counts = cnt;
    }

    public LocalHistogram() {
    }

    public LocalHistogram(int index, int width, int height, int[]
pixels,
            int[][] pg, int type) {
        counts = new int[256];
        initialize(index, width, height, pixels, pg, type);
    }

    public void initialize(int index, int width, int height, int[]
pixels,
            int[][] pg, int type) {
        int[] histogram = new int[256];
        int k = 0;
        int x, y = 0;
        int bin = 0;
        int i = index / width;
        int j = index % width;
        int tmin = 255;
        int tmax = 0;
        for (int g = 0; g < pg.length; g++) {
            y = i + pg[g][0];
            x = j + pg[g][1];
            try {
                if ((x >= width) || (y >= height) || (x < 0) || (y <
0)) {
                    if (type == 0) {
                        k = 0;
                    } else {
                        k = 255;
                    }
                } else {
                    k = pixels[x + width * y] & 0xFF;
                }
            } catch (ArrayIndexOutOfBoundsException ex) {
                k = x + width * y;
            }

            if (type == 0) {
                bin = k + pg[g][2] - 255;
            } else {
                bin = k - pg[g][2] + 255;
            }
            if (tmin > bin) {
                tmin = bin;
            }
            if (tmax < bin) {
                tmax = bin;
            }
            histogram[bin]++;
        }
        this.min = tmin;
        this.max = tmax;
        counts = histogram;
    }

    public void Log() {
        StringBuffer sb = new StringBuffer(200);
        for (int h = 0; h <= 255; h++) {
            if (counts[h] != 0) {
                sb.append(h + " " + counts[h] + " \n");
            }
        }
    }

    public int[] getCounts() {
        return counts;
    }

    public void count() {
        int counter = 0;
        for (int i = 0; i < 256; i++) {
            if (counts[i] >= 0) {
                counter++;
            }
        }
        this.binCount = counter;
    }
```

```
	public int getBinCount() {
		return this.binCount;
	}

	public int getMaximum() {
		return this.max;
	}

	public int getMinimum() {
		return this.min;
	}

	public void doMaximum() {
		int smax = this.max;

		while (counts[smax] == 0 && (smax >= MIN)) {
			smax--;
		}

		this.max = smax;
	}

	public void doMinimum() {
		int smin = this.min;
		while (counts[smin] == 0 && (smin <= MAX)) {
			smin++;
		}
		this.min = smin;
	}

	public void add(LocalHistogram bh) {
		int u = Math.min(min, bh.min);
		int v = Math.max(max, bh.max);
		binCount = Math.max(this.binCount, bh.binCount);
		for (int i = u; i <= v; i++) {
			counts[i] = counts[i] + bh.counts[i];
		}
		min = u;
		max = v;
	}

	public void sub(LocalHistogram bh) {
		int u = Math.min(min, bh.min);
		int v = Math.max(max, bh.max);
		int tmin = u;
		int tmax = v;
		for (int i = u; i <= v; i++) {
			counts[i] = counts[i] - bh.counts[i];
			if (counts[i] < 0) {
				counts[i] = 0;
			}
		}
		while (counts[tmin] == 0 && (tmin < u)) {
			tmin++;
		}
		while (counts[tmax] == 0 && (tmax > tmin)) {
			tmax--;
		}
		min = tmin;
		max = tmax;
	}
}


package morph;

public class FloodFiller {
	int maxStackSize = 600;
	int[] xstack = new int[maxStackSize];
	int[] ystack = new int[maxStackSize];
	int stackSize;
	int max;
	boolean isFloat;
	int[][] image;
	int width;
	int height;

	public FloodFiller(int[][] image, int width, int height) {
		this.image = image;
		this.width = width;
		this.height = height;
	}

	public boolean fill(int x, int y, int fill_color) {
		int color = image[x][y];
		fillLine(image, x, x, y, fill_color);
		int newColor = image[x][y];
		image[x][y] = color;
		if (color == newColor)
			return false;
		stackSize = 0;
		push(x, y);
		while (true) {
			x = popx();
			if (x == -1)
				return true;
			y = popy();
			if (image[x][y] != color)
				continue;
			int x1 = x;
			int x2 = x;
			while (x1 >= 0 && image[x1][y] == color) {
				x1--;
			}
			x1++;
			while (x2 < width && image[x2][y] == color) {
				x2++;
			}
			x2--;
			fillLine(image, x1, x2, y, fill_color);
			boolean inScanLine = false;
			for (int i = x1; i <= x2; i++) {
				if (!inScanLine && y > 0 && image[i][y - 1] ==
color) {
					push(i, y - 1);
					inScanLine = true;
				} else if (inScanLine && y > 0 && image[i][y -
1] != color)
					inScanLine = false;
			}
			inScanLine = false;
			for (int i = x1; i <= x2; i++) {
				if (!inScanLine && y < height - 1 &&
image[i][y + 1] == color) {
					push(i, y + 1);
					inScanLine = true;
				} else if (inScanLine && y < height - 1
						&& image[i][y + 1] != color)
					inScanLine = false;
			}
		}
	}

	int count = 0;

	final void push(int x, int y) {
		stackSize++;
		if (stackSize == maxStackSize) {
			int[] newXStack = new int[maxStackSize * 2];
			int[] newYStack = new int[maxStackSize * 2];
```

```
                System.arraycopy(xstack, 0, newXStack, 0,
maxStackSize);
                System.arraycopy(ystack, 0, newYStack, 0,
maxStackSize);
                xstack = newXStack;
                ystack = newYStack;
                maxStackSize *= 2;
            }
            xstack[stackSize - 1] = x;
            ystack[stackSize - 1] = y;
    }

    final int popx() {
        if (stackSize == 0)
            return -1;
        else
            return xstack[stackSize - 1];
    }

    final int popy() {
        int value = ystack[stackSize - 1];
        stackSize--;
        return value;
    }

    final void fillLine(int image[][], int x1, int x2, int y, int
fill_color) {
        if (x1 > x2) {
            int t = x1;
            x1 = x2;
            x2 = t;
        }
        for (int x = x1; x <= x2; x++) {
            image[x][y] = fill_color;
        }
    }
}



/**
 * @author jhesed tacadena
 */

package segmentation;

import java.awt.image.BufferedImage;
import java.io.File;
import java.util.ArrayList;
import java.util.Collections;
import java.util.concurrent.Callable;

import featureextraction.Nodule;
import fileprocessing.DicomImage;
import fileprocessing.KnownData;

import util.ImageUtil;
import morph.FloodFiller;
import morph.OtsuBinarize;

public class NodulesExtraction implements
Callable<NodulesExtraction> {

    final static int WHITE = -1;
    final static int BLACK = -16777216;
    final static int RED = -65536;
    final static int GREEN = -16711936;
    final static int BLUE = -16776961;
    final static int YELLOW = -256;
    final static int ORANGE = -23296;
    final static int VIOLET = -1146130;
    final static int LIGHT_PINK = -18751;
    final static int TURQUIOSE = -16714241;
    final static int INDIAN_RED = -5236961;
    final static int KHAKI = -2417;
    final static int CYAN = -16711681;
    final static int SGI_CHARTREUSE = -9320847;
    final static int BISQUE = -58172;
    final static int SEA_GREEN = -13726889;
    final static int VIOLET_RED = -3137392;
    final static int GOLDEN_ROD = -2448096;
    final static int SPRING_GREEN = -16711809;
    final static int MINT = -4326199;
    final static int OLIVE_DRAB = -9728477;
    final static int TAN = -2968436;
    final static int ROSY_BROWN = -4419697;
    final static int FUSCHIA = -65281;
    final static int DEEP_PINK = -60269;
    final static int SLATE_BLUE = -8163329;
    final static int GREEN_YELLOW = -5374161;
    final static int SGI_TEAL = -13070706;
    final static int CRIMSON = -2354116;
    final static int LAVENDER_BLUSH = -1122075;
    final static int RASP_BERRY = -7920041;
    final static int PURPLE = -6606593;
    final static int ROYAL_BLUE = -12490271;
    final static int PALE_GREEN = -6751336;
    final static int LEMON_CHIFFON = -1331;
    final static int GOLD = -10496;
    final static int PAPAYA_WHIP = -4139;
    final static int BURLY_WOOD = -2180985;
    final static int CARROT = -1208031;
    final static int TOMATO = -40121;
    final static int MISTY_ROSE = -6943;
    final static int MAROON = -8388608;
    final static int LIGHT_GRAY = -4539718;
    final static int BROWN = -5952982;
    final static int LIGHT_CORAL = -1015680;
    final static int SIENNA = -6270419;
    final static int PEACH_PUFF = -9543;
    final static int PERU = -3308225;
    final static int DARK_KHAKI = -4343957;
    final static int POWDER_BLUE = -5185306;
    final static int THISTLE = -2572328;
    final static int PLUM = -2252579;
    final static int ORCHID = -2461482;
    final static int ORCHID2 = -1148183;
    final static int ORCHID3 = -3315255;
    final static int ORCHID4 = -7649399;
    final static int CADMIUM_ORANGE = -40701;
    final static int BURNT_SIENNA = -7719409;

    private ArrayList<Nodule> nodules = new
ArrayList<Nodule>();

    final double THRESHOLD_CIRCULARITY = 0.60;
    final int THRESHOLD_AREA_MIN = 35;
    final int THRESHOLD_AREA_MAX = 1100;
    final double THRESHOLD_RATIO = 1.5;
    final int THRESHOLD_X_left = 128;
    final int THRESHOLD_X_right = 384;
    final int THRESHOLD_Y_up = 192;
    final int THRESHOLD_Y_down = 320;
    private String colors_cancerous_nodules;
    private String colors_noncancerous_nodules;

    final static int[] colors = { RED, GREEN, BLUE, YELLOW,
ORANGE, VIOLET,
            LIGHT_PINK, TURQUIOSE, INDIAN_RED,
KHAKI, CYAN, SGI_CHARTREUSE,
			BISQUE, SEA_GREEN, VIOLET_RED,
GOLDEN_ROD, SPRING_GREEN, MINT,
			OLIVE_DRAB, TAN, ROSY_BROWN, FUSCHIA,
DEEP_PINK, SLATE_BLUE,
			GREEN_YELLOW, SGI_TEAL, CRIMSON,
LAVENDER_BLUSH, RASP_BERRY,
			PURPLE, ROYAL_BLUE, PALE_GREEN,
LEMON_CHIFFON, GOLD, PAPAYA_WHIP,
			BURLY_WOOD, CARROT, TOMATO,
MISTY_ROSE, MAROON, LIGHT_GRAY, BROWN,
			LIGHT_CORAL, SIENNA, PEACH_PUFF, PERU,
DARK_KHAKI, POWDER_BLUE,
			THISTLE, PLUM, ORCHID, ORCHID2, ORCHID3,
ORCHID4, CADMIUM_ORANGE,
			BURNT_SIENNA };

	final static String[] colors_string = { "RED", "GREEN",
"BLUE", "YELLOW",
			"ORANGE", "VIOLET", "LIGHT PINK",
"TURQUIOSE", "INDIAN RED",
			"KHAKI", "CYAN", "SGI CHARTREUSE",
"BISQUE", "SEA GREEN",
			"VIOLET RED", "GOLDEN ROD", "SPRING
GREEN", "MINT", "OLIVE DRAB",
			"TAN", "ROSY BROWN", "FUSCHIA", "DEEP
PINK", "SLATE BLUE",
			"GREEN YELLOW", "SGI TEAL", "CRIMSON",
"LAVENDER BLUSH",
			"RASP BERRY", "PURPLE", "ROYAL BLUE",
"PALE GREEN",
			"LEMON CHIFFON", "GOLD", "PAPAYA WHIP",
"BURLY WOOD", "CARROT",
			"TOMATO", "MISTY ROSE", "MAROON",
"LIGHT GRAY", "BROWN",
			"LIGHT CORAL", "SIENNA", "PEACH PUFF",
"PERU", "DARK KHAKI",
			"POWDER BLUE", "THISTLE", "PLUM",
"ORCHID", "ORCHID2", "ORCHID3",
			"ORCHID4", "CADMIUM ORANGE", "BURNT
SIENNA" };

	int counter = 0;
	ArrayList<Integer> to_be_eliminated = new
ArrayList<Integer>();

	private DicomImage dicom_image;
	private int[] image_not_colored;
	private int[] image_morph;
	private int[][] image_colored;
	private int[] wavelet_nodules;
	private ArrayList<ArrayList<KnownData>> known_data_list
= new ArrayList<ArrayList<KnownData>>();
	private int width;
	private int height;
	FloodFiller filler;

	public NodulesExtraction(DicomImage dicom_image, int[]
wavelet_nodules,
			int morph[], int width, int height,
			ArrayList<ArrayList<KnownData>> known_data,
File file,
			int threshold, String svm_operation) {

		colors_cancerous_nodules = "";
		colors_noncancerous_nodules = "";

		this.dicom_image = dicom_image;
		this.width = width;
		this.height = height;
		this.known_data_list = known_data;
		this.image_morph = new int[width * height];
		this.image_not_colored = new int[width * height];
		this.image_colored = new int[width][height];
		this.wavelet_nodules = new int[width * height];
		this.image_morph = morph;
		this.wavelet_nodules = wavelet_nodules;
		combineImages();
		morphTo255(image_morph);
		OtsuBinarize bin = new OtsuBinarize();
		this.image_colored = bin.binarize(image_not_colored,
width, height,
				threshold);
		colorNodules(this.image_colored);
		extractNodules();
		eliminateNonNodules();
		coloredToMorph();
		compareToKnownData(file);
		computeColorsOfCancerousAndNonCancerousNodules();
	}

	private void combineImages() {
		for (int i = 0; i < height; i++) {
			int offset = i * height;
			for (int j = 0; j < width; j++) {
				if (wavelet_nodules[offset + j] == WHITE) {
					image_morph[offset + j] = WHITE;
				}
			}
		}
	}

	public void morphTo255(int[] morph) {
		for (int i = 0; i < height; i++) {
			int offset = i * height;
			for (int j = 0; j < width; j++) {
				this.image_not_colored[offset + j] =
morph[offset + j];
				image_not_colored[offset + j] = ((int)
(0.298912D
						* (0xFF & image_not_colored[offset +
j] >> 16)
						+ 0.586611D
						* (0xFF & image_not_colored[offset +
j] >> 8) + 0.114478D * (0xFF & image_not_colored[offset
						+ j]));
			}
		}
	}

	public void colorNodules(int[][] image) {
		filler = new FloodFiller(image, width, height);
		for (int y = 0; y < height; y++) {
			for (int x = 0; x < width; x++) {
				if (image[x][y] == WHITE) {
					filler.fill(x, y, colors[counter]);
					counter++;
				}
			}
		}
	}

	public void extractNodules() {
		for (int i = 0; i < counter; i++) {
			this.nodules.add(new
Nodule(dicom_image.getFileName(), dicom_image
					.getRefSOPClassUID(),
dicom_image.getPatientID(),
					dicom_image.getStudyInstanceUID(),
dicom_image
```

```
                                        .getSeriesInstanceUID(),
dicom_image
                                        .getRefSOPClassUID(),
colors_string[i], colors[i]));
        }

        double[] mean = new double[counter];
        double[] sd = new double[counter];
        int[] perimeter = new int[counter];
        int[] area = new int[counter];
        double[] circularity = new double[counter];
        double[] aspect_ratio = new double[counter];

        for (int i = 0; i < counter; i++) {
            mean[i] = 0;
            sd[i] = 0;
            perimeter[i] = 0;
            area[i] = 0;
        }

        int pixel = 0;
        for (int y = 0; y < height; y++) {
            int offset = y * height;
            for (int x = 0; x < width; x++) {
                if (image_colored[x][y] != BLACK) {
                    innerloop: for (int i = 0; i < counter; i++) {
                        pixel = ((int) (0.298912D
                                * (0xFF &
image_morph[offset + x] >> 16)
                                + 0.586611D
                                * (0xFF &
image_morph[offset + x] >> 8) + 0.114478D * (0xFF &
image_morph[offset
                                + x])));

                        if (image_colored[x][y] == colors[i]) {
                            pixel = ((int) (0.298912D
                                    * (0xFF &
image_morph[offset + x] >> 16)
                                    + 0.586611D
                                    * (0xFF &
image_morph[offset + x] >> 8) + 0.114478D * (0xFF &
image_morph[offset
                                    + x])));

                            mean[i] += pixel;

    nodules.get(i).addXCoord(Integer.valueOf(x));

    nodules.get(i).addYCoord(Integer.valueOf(y));
                            nodules.get(i).addXYCoord(x + ","
+ y);
                            area[i]++;

                            if (x - 1 > -1) {
                                if (image_colored[x - 1][y] !=
colors[i]) {
                                    perimeter[i]++;
                                }
                            }

                            if (x + 1 < width) {
                                if (image_colored[x + 1][y] !=
colors[i]) {
                                    perimeter[i]++;
                                }
                            }

                            if ((y + 1) < height) {
                                if (image_colored[x][y + 1] !=
colors[i]) {
                                    perimeter[i]++;
                                }
                            }

                            if ((x + 1) < width) {
                                if (image_colored[x][y - 1] !=
colors[i]) {
                                    perimeter[i]++;
                                }
                            }
                            break innerloop;
                        }
                    }
                }
            }
        }

        for (int i = 0; i < counter; i++) {
            nodules.get(i).setMean(mean[i] / 262144);
            mean[i] = mean[i] / 262144;

    nodules.get(i).setXMin(Collections.min(nodules.get(i).getXCo
ord()));

    nodules.get(i).setXMax(Collections.max(nodules.get(i).getXC
oord()));

    nodules.get(i).setYMin(Collections.min(nodules.get(i).getYCo
ord()));

    nodules.get(i).setYMax(Collections.max(nodules.get(i).getYC
oord()));
            nodules.get(i).setXCenter(
                    nodules.get(i).getXMin()
                            + ((int) Math

    .floor((nodules.get(i).getXMax() - nodules

    .get(i).getXMin()) / 2)));
            nodules.get(i).setYCenter(
                    nodules.get(i).getYMin()
                            + ((int) Math

    .floor((nodules.get(i).getYMax() - nodules

    .get(i).getYMin()) / 2)));
            Integer xc_max = 0;
            Integer xc_min = 1000;
            Integer yc_max = 0;
            Integer yc_min = 1000;

            for (int j = 0; j < nodules.get(i).getXCoord().size();
j++) {
                if (nodules.get(i).getXCoord().get(j)
                        .equals(nodules.get(i).getXCenter())) {
                    if (yc_max <
nodules.get(i).getYCoord().get(j)) {
                        yc_max =
nodules.get(i).getYCoord().get(j);
                    }
                    if (yc_min >
nodules.get(i).getYCoord().get(j)) {
                        yc_min =
nodules.get(i).getYCoord().get(j);
                    }
                }
                if (nodules.get(i).getYCoord().get(j)
                        .equals(nodules.get(i).getYCenter())) {
```

```
                    if (xc_max <
nodules.get(i).getXCoord().get(j)) {
                        xc_max =
nodules.get(i).getXCoord().get(j);
                    }
                    if (xc_min >
nodules.get(i).getXCoord().get(j)) {
                        xc_min =
nodules.get(i).getXCoord().get(j);
                    }
                }
            }
            nodules.get(i).setHeight(yc_max - yc_min);
            nodules.get(i).setWidth(xc_max - xc_min);

            if (nodules.get(i).getWidth() >
nodules.get(i).getHeight()) {
                nodules.get(i).setAspectRatio(
                        (double) nodules.get(i).getWidth()
                                / nodules.get(i).getHeight());
            }

            else {
                nodules.get(i).setAspectRatio(
                        (double) nodules.get(i).getHeight()
                                / nodules.get(i).getWidth());
            }
            aspect_ratio[i] = nodules.get(i).getAspectRatio();
        }

        double temp_sd = 0;
        for (int y = 0; y < height; y++) {
            int offset = y * height;
            for (int x = 0; x < width; x++) {
                if (image_colored[x][y] != BLACK) {
                    innerloop2: for (int i = 0; i < counter; i++) {
                        if (image_colored[x][y] == colors[i]) {
                            temp_sd = ((int) (0.298912D
                                    * (0xFF &
image_morph[offset + x] >> 16)
                                    + 0.586611D
                                    * (0xFF &
image_morph[offset + x] >> 8) + 0.114478D * (0xFF &
image_morph[offset
                                    + x])));

                            sd[i] += ((temp_sd - mean[i]) *
(temp_sd - mean[i]));
                            break innerloop2;
                        }
                    }
                }
            }
        }

        for (int i = 0; i < counter; i++) {

            circularity[i] = (4 * Math.PI * area[i])
                    / (perimeter[i] * perimeter[i]);

            if (nodules.get(i).getYCenter() > 400) {
                if (!to_be_eliminated.contains(i)) {
                    to_be_eliminated.add(i);
                }
            }

            if (circularity[i] < 0.36 && area[i] > 240) {
                if (!to_be_eliminated.contains(i)) {
                    to_be_eliminated.add(i);
                }
            }
            if (aspect_ratio[i] > 3.0 && area[i] > 200) {
                if (!to_be_eliminated.contains(i)) {
                    to_be_eliminated.add(i);
                }
            }

            if (area[i] <= 60 && circularity[i] < 0.45) {
                if (!to_be_eliminated.contains(i)) {
                    to_be_eliminated.add(i);
                }
            }
            if (area[i] <= 60 && aspect_ratio[i] > 1.40) {
                if (!to_be_eliminated.contains(i)) {
                    to_be_eliminated.add(i);
                }
            }

            if (area[i] < THRESHOLD_AREA_MIN
                    || area[i] > THRESHOLD_AREA_MAX
                    || (area[i] <= 240 && area[i] > 60 &&
circularity[i] < THRESHOLD_CIRCULARITY)
                    || (area[i] <= 200 && aspect_ratio[i] >
THRESHOLD_RATIO)) {
                if (!to_be_eliminated.contains(i)) {
                    to_be_eliminated.add(i);
                }
            } else {
                nodules.get(i)
                        .setSd(Math
                                .sqrt((sd[i] + (((mean[i] *
mean[i]) * (262144 - area[i])))) / 262144));
                nodules.get(i).createCoordString();
                nodules.get(i).setPerimeter(perimeter[i]);
                nodules.get(i).setArea(area[i]);
                nodules.get(i).setCircularity(circularity[i]);
            }
        }

        Collections.sort(to_be_eliminated);
        for (int i = to_be_eliminated.size() - 1; i >= 0; i--) {
            nodules.remove(to_be_eliminated.get(i).intValue());
            counter--;
            to_be_eliminated.set(i,
colors[to_be_eliminated.get(i)]);
        }
    }

    private void eliminateNonNodules() {
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                if
(to_be_eliminated.contains(image_colored[x][y])) {
                    filler.fill(x, y, BLACK);
                }
            }
        }
    }

    public void compareToKnownData(File file) {
        try {
            int index;

            for (int i = 0; (i < nodules.size()); i++) {
                inner: for (int k = 0; k < known_data_list.size();
k++) {
                    for (int z = 0; z <
known_data_list.get(k).size(); z++) {
                        index =
known_data_list.get(k).get(z).getImageSopUid()
```

```
	.indexOf(nodules.get(i).getPrimarySopUid());
					if (index != -1) {
						if
(!Collections.disjoint(nodules.get(i)
							.getXYCoord(),
known_data_list.get(k)

	.get(z).getXYCoordinates(index))) {

	this.nodules.get(i).setMalignancy(

	known_data_list.get(k).get(z)

	.getMalignancy());

	this.nodules.get(i).setIsCancerous("yes");
								break inner;
							}
						}
					}
				}
			}
		} catch (Exception e) {
		}
	}

	private void coloredToMorph() {
		for (int i = 0; i < height; i++) {
			int offset = i * height;
			for (int j = 0; j < width; j++) {
				if (image_colored[j][i] == BLACK) {
					image_morph[offset + j] = BLACK;
				}
			}
		}
	}

	public ArrayList<Nodule> getNodules() {
		return this.nodules;
	}

	public int[][] getColoredArray() {
		return this.image_colored;
	}

	public BufferedImage getColoredImage() {
		return
ImageUtil.arrayToColoredBufferedImage(this.image_colored,
width,
				height);
	}

	public BufferedImage getRuleBasedMorphImage() {
		return ImageUtil.getImageFromArray(this.image_morph,
width, height);
	}

	public BufferedImage getSegmentedImage() {
		return
ImageUtil.arrayToColoredBufferedImage(this.image_colored,
width,
				height);
	}

	public void
computeColorsOfCancerousAndNonCancerousNodules() {
		for (int i = 0; i < nodules.size(); i++) {
			if (nodules.get(i).nativeGetMalignancy().equals("0"))
{
				colors_noncancerous_nodules +=
nodules.get(i).nativeGetColor()
						+ ", \n";
			} else {
				colors_cancerous_nodules +=
nodules.get(i).nativeGetColor()
						+ ", \n";
			}
		}
	}

	public String getColorsOfCancerousNodules() {
		return this.colors_cancerous_nodules;
	}

	public String getColorsOfNonCancerousNodules() {
		return this.colors_noncancerous_nodules;
	}

	@Override
	public NodulesExtraction call() throws Exception {
		return this;
	}
}


/**
 * LIBSVM library is being used here:
 * Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for
support
 * vector machines. ACM Transactions on Intelligent Systems and
 * Technology, 2:27:1--27:27, 2011. Software available at
 * http://www.csie.ntu.edu.tw/~cjlin/libsvm
 *
 * @author jhesed tacadena
 */

package libsvm;

import java.io.*;
import java.util.*;

public class svm_train {
	private svm_parameter param;
	private svm_problem prob;
	private svm_model model;
	private String input_file_name;
	private String model_file_name;
	private String error_msg;
	private int kernel;
	@SuppressWarnings("unused")
	private int nr_fold = 5;
	private double gamma;
	private int degree;
	private double c;

	public svm_train(String file) {
		this.input_file_name = file;
	}

	public svm_train(String file, int kernel, double gamma, int
degree, double c) {
		this.input_file_name = file;
		this.kernel = kernel;
		this.gamma = gamma;
		this.degree = degree;
		this.c = c;
	}

	public void run() throws IOException {
		setParameters();
		read_problem();

		if (error_msg != null) {
			System.err.print("ERROR: " + error_msg + "\n");
			System.exit(1);
		}
		model = svm.svm_train(prob, param);
		svm.svm_save_model(model_file_name, model);
	}

	private void setParameters() {
		param = new svm_parameter();
		param.svm_type = svm_parameter.C_SVC;
		param.probability = 1;
		param.kernel_type = kernel;
		param.degree = degree;
		param.gamma = gamma; // 1/num_features
		param.coef0 = 1;
		param.cache_size = 100;
		param.C = c;
		param.eps = 1e-3;
		param.p = 0.1;
		param.shrinking = 1;
		param.nr_weight = 0;
		model_file_name = input_file_name + ".model";
	}

	private void read_problem() throws IOException {
		BufferedReader fp = new BufferedReader(new
FileReader(input_file_name));
		ArrayList<Double> y_arr = new ArrayList<Double>();
		ArrayList<svm_node[]> attr_list_ptr_arr = new
ArrayList<svm_node[]>();
		int max_index = 0;

		while (true) {
			String line = fp.readLine();
			if (line == null)
				break;

			StringTokenizer st = new StringTokenizer(line, "
\t\n\r\f:");
			y_arr.add(retDouble(st.nextToken()));
			int iter = st.countTokens() / 2;
			svm_node[] node_arr = new svm_node[iter];

			for (int j = 0; j < iter; j++) {
				node_arr[j] = new svm_node();
				node_arr[j].index = retInt(st.nextToken());
				node_arr[j].value = retDouble(st.nextToken());
			}

			if (iter > 0)
				max_index = Math.max(max_index,
node_arr[iter - 1].index);
			attr_list_ptr_arr.add(node_arr);
		}

		prob = new svm_problem();
		prob.l = y_arr.size();
		prob.x = new svm_node[prob.l][];
		for (int i = 0; i < prob.l; i++)
			prob.x[i] = attr_list_ptr_arr.get(i);

		prob.y = new double[prob.l];
		for (int i = 0; i < prob.l; i++)
			prob.y[i] = y_arr.get(i);

		if (param.gamma == 0 && max_index > 0)
			param.gamma = 1.0 / max_index;
		fp.close();
	}

	private static double retDouble(String s) {
		return Double.valueOf(s).doubleValue();
	}

	private static int retInt(String s) {
		return Integer.parseInt(s);
	}
}



/**
 * LIBSVM library is being used here:
 * Chih-Chung Chang and Chih-Jen Lin, LIBSVM : a library for
support
 * vector machines. ACM Transactions on Intelligent Systems and
 * Technology, 2:27:1--27:27, 2011. Software available at
 * http://www.csie.ntu.edu.tw/~cjlin/libsvm
 *
 * @author jhesed tacadena
 */

package libsvm;

import java.io.*;
import java.util.*;

public class svm_train {
	private svm_parameter param;
	private svm_problem prob;
	private svm_model model;
	private String input_file_name;
	private String model_file_name;
	private String error_msg;
	private int kernel;
	@SuppressWarnings("unused")
	private int nr_fold = 5;
	private double gamma;
	private int degree;
	private double c;

	public svm_train(String file) {
		this.input_file_name = file;
	}

	public svm_train(String file, int kernel, double gamma, int
degree, double c) {
		this.input_file_name = file;
		this.kernel = kernel;
		this.gamma = gamma;
		this.degree = degree;
		this.c = c;
	}

	public void run() throws IOException {
		setParameters();
		read_problem();

		if (error_msg != null) {
			System.err.print("ERROR: " + error_msg + "\n");
			System.exit(1);
		}
		model = svm.svm_train(prob, param);
		svm.svm_save_model(model_file_name, model);
	}
	private void setParameters() {                                                    }
		param = new svm_parameter();
		param.svm_type = svm_parameter.C_SVC;
		param.probability = 1;
		param.kernel_type = kernel;
		param.degree = degree;
		param.gamma = gamma; // 1/num_features
		param.coef0 = 1;
		param.cache_size = 100;
		param.C = c;
		param.eps = 1e-3;
		param.p = 0.1;
		param.shrinking = 1;
		param.nr_weight = 0;
		model_file_name = input_file_name + ".model";
	}

	private void read_problem() throws IOException {
		BufferedReader fp = new BufferedReader(new
FileReader(input_file_name));
		ArrayList<Double> y_arr = new ArrayList<Double>();
		ArrayList<svm_node[]> attr_list_ptr_arr = new
ArrayList<svm_node[]>();
		int max_index = 0;

		while (true) {
			String line = fp.readLine();
			if (line == null)
				break;

			StringTokenizer st = new StringTokenizer(line, "
\t\n\r\f:");
			y_arr.add(retDouble(st.nextToken()));
			int iter = st.countTokens() / 2;
			svm_node[] node_arr = new svm_node[iter];

			for (int j = 0; j < iter; j++) {
				node_arr[j] = new svm_node();
				node_arr[j].index = retInt(st.nextToken());
				node_arr[j].value = retDouble(st.nextToken());
			}

			if (iter > 0)
				max_index = Math.max(max_index,
node_arr[iter - 1].index);
			attr_list_ptr_arr.add(node_arr);
		}

		prob = new svm_problem();
		prob.l = y_arr.size();
		prob.x = new svm_node[prob.l][];
		for (int i = 0; i < prob.l; i++)
			prob.x[i] = attr_list_ptr_arr.get(i);

		prob.y = new double[prob.l];
		for (int i = 0; i < prob.l; i++)
			prob.y[i] = y_arr.get(i);

		if (param.gamma == 0 && max_index > 0)
			param.gamma = 1.0 / max_index;
		fp.close();
	}

	private static double retDouble(String s) {
		return Double.valueOf(s).doubleValue();
	}

	private static int retInt(String s) {
		return Integer.parseInt(s);
	}
```

XI. Acknowledgement

First, I would like to thank my personal savior and God, Lord Jesus Christ, for giving me wisdom to successfully finish this special problem. During the stressful days of programming the system, there were so many times that I felt hopeless in finishing the SP in time – because of lack of understanding in algorithms and some unexpected results. But His words gave me strength and hope to finish this SP.

Second, I want to thank my family for supporting me all the way. To my mother, who never forgets to pray for me each morning during her daily devotion; for being so understanding and supportive. To my father, for giving me food and beverages to enjoy during my overnight stays just to finish this SP. To my sister, for forcing herself to stay awake during my stressful nights just to provide me with moral support; for her initiative in borrowing the printer and asking our church mates to support me in prayer during the night before my defense. To my brother, for constantly asking me to relax; for helping me think of optimal solutions to the bugs and problems that I encountered. To Tita Tess, for letting me use her printer. To Koonikey, our dog, for being awake to guard and to be with me during those nights.

Third, I want to thank the UP professors who gave their time just to help me. I would like to thank my adviser, Sir Solano, for being very supportive. He never forgot to ask me from time to time the progress and the problems that I encounter. He gave me resources that helped me in understanding the topics. He also arranged meetings with experts so that we will be enlightened about our questions about the topic. During the times I was struggling about unexpected results of the system, he guided and helped me in accomplishing it successfully. To Doc Gonzaga, though he was very busy because he is the dean of our college, he gave time to answer my questions about wavelets and gave me very useful advice. To Ma'am Pia: she gave effort in answering my chat messages. She also encouraged and cheered me up. She was very supportive. To Ma'am Perl, for being ready to answer our SVM questions. To the panel who listened to me during the proposal and defense.

Fourth, I would like to thank God for my church mates who prayed for me. Especially during the night before my scheduled defense, a simple miracle happened. I would like to thank them because I know they are the key to it. To kuya Rommel, for his continuous encouraging texts to me, to kuya Nald, Ria, Jm, Roselyn, tito Boy, Zoren, Raymark, and to all my churchmates.

Fifth, to two of my most special friends. To Hannah, for being so patient to me. Even though I know she has her own personal problems, she never got tired of listening to me as I released all my stress about this SP. Her support, prayers, texts, and even calls, gave me inspiration to finish it. She became my absorber. She comforted me specially those times. To my best friend Kelvin, for being awake as we chat about our present situations – this is a stress reliever. I would like to thank him for being able to relate to the hardness of UP thesis, and cheering me up.

Sixth, I would like to thank my block mates, especially those who really encouraged and helped me – Althom, Loann and Jeselle, for answering my consultations to them; For trying to give me algorithms and references to solve some of my problems. For even calling me to propose their solutions.

This SP will not be successful without all of you. I would like to express my gratitude for supporting me all the way. God bless us all.