Fall 2010

Content based image retrieval for bio-medical images

Vikas Nahar

Department:

CONTENT BASED IMAGE RETRIEVAL

FOR BIO-MEDICAL IMAGES

by

VIKAS NAHAR

A THESIS

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN COMPUTER SCIENCE

2010

Approved by

Fikret Ercal, Advisor
R. Joe Stanley, Co-advisor
Ralph Wilkerson

ABSTRACT

Content Based Image Retrieval System (CBIR) is used to retrieve images similar to the query image. These systems have a wide range of applications in various fields. Medical subject headings, key words, and bibliographic references can be augmented with the images present within the articles to help clinicians to potentially improve the relevance of articles found in the querying process. In this research, image feature analysis and classification techniques are explored to differentiate images found in biomedical articles which have been categorized based on modality and utility. Examples of features examined in this research include: features based on different histograms of the image, texture features, fractal dimensions etc. Classification algorithms used for categorization were

1) Mean shift clustering

2) Radial basis clustering

Different combinations of features were selected for classification purposes and it was observed that features incorporating soft decision based HSV histogram features give the best results. A library of features was then developed which can be used in RapidMiner. Experimental results for various combinations of features have also been included.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my co advisor, Dr. R. Joe Stanley, for providing me with an opportunity to work under him. I am extremely grateful to him for his excellent guidance, technical help, motivation and continued patience throughout this research, and the course of my graduate program and during the writing of this thesis. I would also like to express my gratitude towards Dr. Fikret Ercal and Dr. R. Wilkerson for the valuable suggestions, discussions and comments. I would also like to thank Dr. Shamik Sural for helping me in my research.

Dr. Kapil Gupta, Soumya De and Mohammad Das of the "ECE 212 Lab" have played a significant role in my research work and I would like to thank each one of them for the same.

Special thanks to my parents and brother for their love and understanding all these years and during the course of my graduate studies. Their support and faith in me, has always encouraged me and helped me come out stronger from all the hard times so far. Last but not the least; I would like to thank my friends Pavitra and Abhinav who have provided valuable insight into certain areas of my research.

TABLE OF CONTENTS

Page

LIST OF ILLUSTRATIONS

Page

LIST OF TABLES

Page

1. INTRODUCTION

1.1 CONTENT BASED IMAGE RETRIEVAL

CBIR systems retrieve images from that database which are similar to the query image. This is done by actually matching the content of the query image with the images in database. Content of an image can be described in terms of color, shape and texture of an image. Primarily research in Content Based Image Retrieval has always focused on systems utilizing color and texture features [1]. There has also been some work done using some local color and texture features. These account for Region Based Image Retrieval (RBIR) [2]. Apart from this, there has been wide utilization of color, shape and texture features for devising techniques to measure similarity amongst images, which can improve retrieval efficiency. This research focuses on combining texture features as well as features based on Luminance histogram, circular HSV histogram and fractal dimensions. The images in the database having similar content as that of the query image are the output of CBIR systems. CBIR systems have wide range of applications. Some of them are listed below:

- Art Collection
- Photograph Archives
- Military Purposes
- Crime Prevention

Recently, CBIR systems have found their way into biomedical field. The following section overviews the usage of CBIR systems in biomedicine.

1.2 CBIR IN BIOMEDICINE

Evidence Based practice (EBP) is a type of practice where the professionals seek evidence before making any professional decisions. The evidence could be sought by carefully examining the research done in the area or looking at similar situations in the past. EBP is often used by clinicians to access medical cases. Clinicians can use the information in biomedical publications to aid in assessing patient cases. For searching a relevant article a clinician may use medical subject headings, key words or bibliographic references in querying biomedical article databases such as Medline or Grateful Med. Image content in biomedical publications may also provide relevant information and potentially enhance the information and relevance of articles found in the querying process [3].

1.3 RESEARCH OBJECTIVE

In this research, various techniques for Content Based Image Retrieval (CBIR) systems have been studied and a number of features for classifying images extracted from biomedical journal articles into categories based on modalities have been investigated. These features were combined into different groups and used for classification. Experiments were performed on various such groups for achieving best results. Libraries were formed based on different groups of features. These libraries can be used with open source data mining tool RapidMiner.

1.4 DATA SET

In this research, the modality case was used for evaluating the feature groups and the different classification algorithms investigated. For summary, Table 1.1 presents the nine categories for image modality. Overall, the experimental data set consisted of 742 images obtained from the Communications Engineering Branch of the National Library of Medicine. Table 1.2 contains the distribution of the images in from different categories of image modalities given in Table 1.1.

Table 1.1. Categories for image modality

Category	Definition
Chart / Graph	A geometric diagram consisting of dots, lines, and bars.
Drawing	A hand drawn illustration.
Flowchart	A symbolic representation of sequence of activities.
Form	A compilation of textual data and/or drawings related to patient and/or clinical process.
Histology	An image of cells and tissue on the microscopic level.
Photograph	Picture obtained from a camera.
Radiology	A 2D view of an internal organ or structure.
Table	Data arranged in a grid.
Mixed	Images combining modalities.

Table 1.2. Number of images from each category for image modality case

Category	Number of Images Used in Study
Chart / Graph	108
Drawing	68
Flowchart	7
Form	11
Histology	134
Photograph	252

Table 1.2. Number of images from each category for image modality case (cont.)

Radiology	108
Table	46
Mixed	8

1.5 TOOLS AND SOFTWARE

A part of this research involves developing libraries for software called RapidMiner. RapidMiner is an open source data mining tool. RapidMiner has a wide range of operators and nested operators which help in training the system for efficient mining. It also has an interactive GUI to setup the mining processes. RapidMiner scripts can also be written in xml. Some image mining operators have been developed, which help in calculating various image features and save them as text file which can again be used by classification algorithms to classify various images.

RapidMiner is built in Java and hence libraries that were developed have also been written in Java. The programming environment consisted of the following tools:

- Eclipse IDE
- Java Advanced Imaging Library (JAI)
- Java Development Kit (JDK 5)
- Java Runtime Environment (JRE 1.5)
- Apache Ant

Eclipse IDE is an open source project used to write, compile and build Java projects. Apache Ant is an open source build tool which helps building java projects. Eclipse comes with the built-in Ant compiler and builds the projects using build.xml file. Earlier build tools were OS dependent, but Ant is not OS dependent and this is the major advantage of using Ant to build projects. By using Ant to build projects, it can be ensured that projects can again be modified and built on any kind of platform.

1.6 OUTLINE OF THESIS

The remainder of the thesis is outlined as follows. Section 2 describes the literature review. Section 3 speaks about the features that were investigated as a part of the research and also about the classification algorithms that were used. Section 4 describes the process of converting the code from Matlab implementation to Java version. It also describes how libraries are created for RapidMiner and how they can used in RapidMiner environment. In Section 5, various experiments and their results have been presented. The conclusions derived from the results are discussed in Section 6, along with future directions to improve the studied approaches.

2. LITERATURE REVIEW

Some preliminary background is presented for some of the different types of features found in the literature which provided the basis for the features and classification techniques investigated in this research. The goal of image retrieval system is to retrieve all the images which are similar to the given query image. Some image retrieval systems use global color and texture features while some use local color and texture features. The later approach segments image into regions based on color and texture. These systems are called Region Based Image Retrieval (RBIR) systems and have proven to be more effective in terms of image retrieval.

Color, shape and texture features of an image are widely used to measure similarity between images. These features are combined to achieve higher retrieval efficiency [1]. This paper [1] presents a method to combine all the three features within a multiresolution multigrid framework. First, the image is partitioned into non-overlapping tiles of equal size. Then color moments and the Gabor filter response of these tiles are used as local descriptors. This information is captured at two resolutions and two grid layouts which give different details of the same image. Gradient Vector Flow fields are used to calculate edge images which are used to gather information about shape of the image. Shape features are then recorded using invariant moments. A combination of Most Similar Highest Priority (MSHP) principle and the adjacency matrix of a bipartite graph formed using the tiles of query image and the target image is provided for image matching. The dataset consists of 1000 Corel images, 100 images each of the following 10 categories: Africa, Beaches, Building, Bus, Dinosaur, Elephant, Flower, Horses,

Mountain and Food. The experimental results of the proposed system are compared with the results from SIMPLIcity [3] and FIRM [4] image retrieval systems.

A combination of texture features and grey level features has been explored for Content Based Image Retrieval Systems [5]. This paper [5] examines the hypothesis that two images are likely to have similar texture features. It also examines a new texture feature called "N x M gram" which is based on "N-gram" technique which is widely used to measure similarity between text. Three different methods are suggested for comparing the N x M- gram [5]:

1) Dot product of the N x M- gram vector.
2) Similarity can be improved by subtracting the average N x M –gram vector from the query image as well as image database.
3) The third method gauges similarity only in terms of the number of common N x M –grams.

In addition to N x M –gram following grey level distribution features were also calculated [5]:

1) Grey level features: in this method two different similarity measures have been used; first, histogram intersection is used to measure the similarity between the histogram of the images and the second method uses statistics on grey level distribution which includes; mean, standard deviation, RMS, skew and kurtosis. Euclidean is used as similarity measure for the statistical features.
2) Local standard deviation: it is the standard deviation in the local window. Histogram intersection and Euclidean distance is used for similarity measure.

3) Grey level co-occurrence: it is defined as "the measure of two-dimensional spatial dependency of the grey level for a fixed distance and/or angular spatial relationship" [5]. Weighted Euclidean distance is used as similarity measure.
4) Laws texture feature: in this method first the image is first convolved with spatial filter and then the texture features are extracted and used to classify the images. Weighted Euclidean distance is used as a similarity measure.

The method was tested on 200 images from the following 22 categories: fingerprints, floor plans, comics, music notes, tables, mug shots, houses, aircraft, animals, topological maps, road maps, satellite, aerial, Arabic, Chinese, English Print, English Lyrics, English Hand, Heb. script, Heb. hand, Heb. print1 and Heb. print2. N x M-gram and all other features of a query image are calculated and compared with the images in the database using the similarity measure mentioned along with each feature. Images in databases are then ranked with respect to the query image. Retrieval accuracy is evaluated using the normalized recall metric [6]. The N x M-grams work very well for simple images like music notes and documents but it gives poor results for rich images like aircraft and aerial views.

A combination of color and texture features with different weights has been investigated for Content Based Retrieval Systems [7]. In [7], color features are generally extracted from the color histogram of the image. The color histogram is generated using the HSV representation of the image. The color histogram is soft decision based i.e. each pixel contributes weighted values of its hue and intensity based on its saturation. Thus, the histogram contains two components the "true color" and "gray color". Texture features are extracted using Haar or Daubechies' wavelet. These two feature vectors are

then normalized so that the bin value is always between [0, 1]. During the image retrieval process, the texture and color features of the query image is combined and weighted and compared with feature vectors of the images in the database using the Manhattan distance Metric. The retrieved results depend on the weight given to each of the feature vector. Experiments were performed on 50 random images from a database of 28168 images and results have shown that feature vector weight W_t (weight for texture feature) in the range of W_c +-0.1 to W_c +-0.2 (W_c is weight for color feature) gives best results.

3. IMAGE FEATURES

3.1 IMAGE PREPROCESSING

In this research, numerous global measures and histogram-based, texture-based, and wavelet-based features have been investigated. The image data set consisted of 742 images of various categories as described in Table 1.1. These images were in .gif and .jpg format. The feature values calculated were inconsistent for the same image using different file formats. Since most of the images were .jpg format, all of the files were converted to this format to try and be as consistent as possible. All the images of the data set were converted to .jpg format using Paint Shop Pro. These .jpg images were then used for feature calculation.

3.2 GLOBAL FEATURES

The following global image features were examined for distinguishing the images in the different categories found in Table 1.1: 1) the standard deviation of the standard deviation computed over the columns of the image. Let stdColRed, stdColGreen, and stdColBlue denote these standard deviation features computed for the red, green, and blue planes, respectively, for a color image. stdColRed, stdColGreen, and stdColBlue are the same values for grayscale images. These features were examined to evaluate the change in contrast across the image. 2) The ratio of the pixels in the image in which the green value is less than the red value and the green value is less than the blue value, given as pixelsG, to the area or size of the image. The ratio is denoted as shown in following expression.

$$\text{percentG} = \frac{\text{pixelsG}}{\text{area}} \tag{1}$$

This feature provides a basic measure of the greenness of the image. 3) The ratio of the pixels in the image with luminance value greater than or equal to 250, given as pixelsWhite, to the area/size of the image. The ratio is denoted as shown in following expression.

$$\text{percentWhite} = \frac{\text{pixelsWhite}}{\text{area}} \tag{2}$$

This feature gives a whiteness metric which is prevalent in images from several modality categories. 4) The square root of the area/size of the image, denoted as sqrtArea. This feature was used to see if image area/size for the different categories was a distinguishing characteristic. 5) The ratio of the sum of the absolute differences between the red and green values and the red and blue values for each pixel in the image to the area/size of the image. Let $(R_{(i,j)},G_{(i,j)},B_{(i,j)})$ denote red, green, and blue values at pixel location (i,j), respectively. The ratio is given as denoted in following expression.

$$\text{sumDiff} = \frac{\sum_{\text{(i,j) within the image}} \left|R_{(i,j)} - G_{(i,j)}\right| + \left|R_{(i,j)} - B_{(i,j)}\right|}{\text{area}} \tag{3}$$

This feature provides a basic homogeneity measure. 6) the ratio of the pixels in the image with luminance value less than or equal to 30, given as pixelsDark, to the area/size of the image. The ratio is denoted as shown below.

$$\text{percentDark} = \frac{\text{pixelsDark}}{\text{area}} \tag{4}$$

7) the ratio of the pixels in the image with luminance values between 50 and 150 (inclusive), labeled as pixels50to150, to the area/size of the image. The ratio is defined as shown in expression below:

$$\text{percentMiddle} = \frac{\text{pixels50to150}}{\text{area}} \tag{5}$$

The goal with this feature is to distinguish the images within particular categories as being bright (white), dark or moderate. 8) Estimates of the fractal dimension [8]. The first estimate of the fractal dimension is based on second order discrete derivatives computed column-wise over the image, given as fractDim1. The second estimate is based on second order discrete derivatives computed column-wise over the image using symlets (sym5), labeled as fractDim2. Fractal dimension was investigated here as a global image measure.

3.3 HISTOGRAM BASED FEATURES

3.3.1 LUV Color Space. Several descriptors computed from the L histogram of the image (from the LUV color space). 1) The first of the descriptors is the most frequently occurring L value in the image, denoted as mostFrequentGray. If more than one value had the same (maximum) frequency of occurrence, the lowest of those L values

was chosen. 2) The second of the descriptors is the ratio of the number of pixels with the most frequently occurring L value, given as the numberPixelsMostFrequentGray, to the area/size of the image. The ratio is defined as shown in following:

$$\text{percentMostFrequent} = \frac{\text{numberPixelsMostFrequentGray}}{\text{area}} \tag{6}$$

3) The third descriptor is the average L value over the image, given as avgGray. The fourth descriptor is the standard deviation L value over the image, labeled as stdGray. Several studies have cited descriptors from the LUV color space for content-based image retrieval.

3.3.2 Luminance Histogram. Several color/luminance count indices are computed over the luminance histogram. 1) The first index is computed as the square root of the number of luminance histogram bins with counts greater than or equal the area of the image times 0.001 (0.001 is chosen to discount histogram bins with very small counts), denoted as colorCount. 2) The second index is computed as the square root of the number of luminance histogram bins with counts greater than 0, denoted as colorCount1. 3) The third index is computed as the square root of the number of luminance histogram bins with counts greater than or equal to the area of the image times 0.01, denoted as colorCount2. colorCount, colorCount1, and colorCount2 provide three grayscale counting metrics for quantifying grayscale variation within the images in different categories.

3.3.3 HSV Color Space. Mostly used method of generating a color histogram is by counting the number of pixels having the same color or generating three separate

histograms for RGB colors and then combining them. This research examines a one-dimensional HSV histogram [9]. Each pixel in the image contributes weighted values of its hue 'H' and intensity 'V' based on its saturation 'S' to the histogram. Thus, the histogram consists of two components, the 'color components' which store the contribution of hue from each pixel and the 'gray component' which store the contribution from intensity value at each pixel. The histogram retains the smoothness between the adjacent components and this allows us to perform a window based smoothing of the histogram.

3.3.3.1 Histogram generation and window based smoothing. "Saturation projection" is used to determine the weights by which each pixel contributes its hue to the color component of the histogram and its intensity to the gray component of the histogram. Thus, it is clear that every pixel contributes to the two components of the histogram. The weight is dependent on saturation level. The weight of hue component, $w_h(s)$ and the weight of intensity of component $w_i(s)$ are computed using the following equations [9]:

$$w_h(s) = s^r \text{ where } r \in [0,1] \tag{7}$$

$$w_i(s) = 1 - w_h(s) \tag{8}$$

The number of bins in histogram are now determined. As the histogram consists of two components viz**.** the color component and the gray component, the total number of bins by adding the total number of bins required for the color component and the total number of bins required for the gray component. Let N_h be the number of bins for color

component, N_g be the number of bins for gray component and N be the total number of bins in the histogram [9].

$$N_h = Round(2\pi MULT\_FCTR) + 1 \tag{9}$$

$$N_g = Round(I_{\max} / DIV\_FCTR) + 1 \tag{10}$$

$$\mathrm{N} = N_h + N_g \tag{11}$$

$MULT\_FCTR$: is the multiplying factor that determines the quantization level for the hues. $I_{\max}$: is the maximum intensity generally 255. DIV_FCTR: is the division factor that determines the number of quantized gray levels.

The algorithm for generating the HSV histogram can be written as follows [9]:

For each pixel in image:

Convert RGB values to HSV

Update histogram as follows:

$$Hist[Round(H.MULT\_FCTR)] = Hist[Round(H.MULT\_FCTR)] + w_h(s)$$

$$Hist[Round(2\pi MULT\_FCTR) + ROUND(V / DIV\_FCTR)] = Hist[Round(2\pi MULT\_FCTR) + ROUND(V / DIV\_FCTR)] + w_i(s)$$

All the traditional histograms do not provide perceptual gradation of colors, thus operations like smoothing is not provided**,** but the HSV histogram retains this property, and hence window based smoothing can be performed using the following equation [9]:

$$Hist_w(j) = \sum_{i=j-N}^{j+N} w(i-j)Hist(i) \tag{12}$$

where: $j \in [0, N_h + N_g - 1]$ and

$$w(i-j) = 2^{-|i-j|}$$

3.3.3.2 Features. Following features were computed based on the HSV histogram, 1) the bin number of the histogram which has the maximum count (mostFrequentComponent), 2) the average value of the color and gray component in an image (avgVal), 3) the standard deviation of color and gray components in the image HSV histogram (stdVal). The first set of feature values were calculated with using the HSV histogram without smoothing, and the second set of feature values were calculated after smoothing the HSV histogram. Both the set of features were used in classification of images.

3.4 WDD FEATURES

The features investigated included the basis function features computed based on correlating the luminance histogram, smoothened and unsmooth HSV histogram with a set of six weight density distribution (WDD) functions [10], i.e. basis functions. The WDD functions are shown in Figure 3.1 below.

Twelve WDD-based features are computed. Each of the WDD functions is decomposed into 256 discrete points for point-to-point correlation with the luminance histograms of the images.

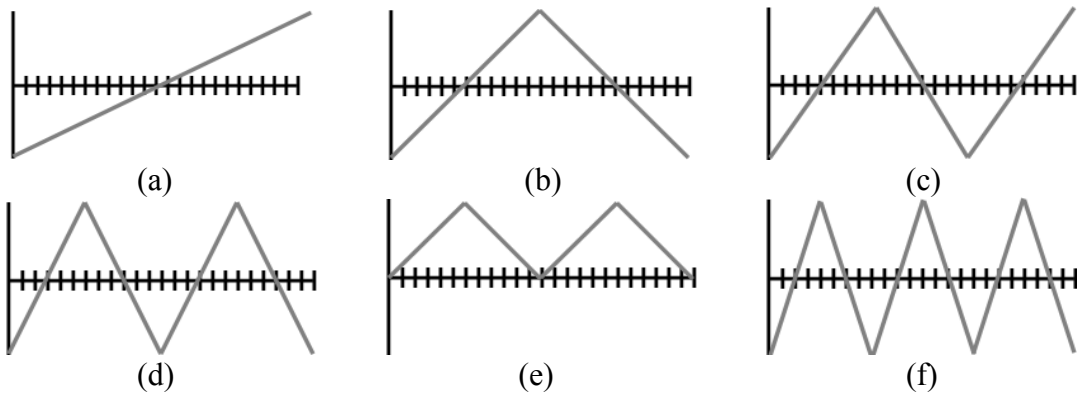


Figure 3.1. WDD functions [10] used for computing basis function features

Let W_1 denote the WDD function in Figure 3.1 (a), W_2 denote the WDD function in Figure 3.1 (b) and so on. For an image, six WDD features ($f_{\Gamma 1}, \ldots, f_{\Gamma,6}$) are computed using the luminance histogram Γ according to the expression,

$$f_{\Gamma,k} = \sum_{i=0}^{255} \Gamma(i) W_k(i) \quad \text{for k = 1, 2, …, 6.} \tag{13}$$

The WDD functions were adjusted to have 256 points that were point-to-point correlated and multiplied with the WDD functions, with the feature values as the sums of the point-to-point multiplications. Six additional features ($f_{\Gamma,7}, \ldots, f_{\Gamma,12}$) are computed by correlating the six WDD functions with the sequence of absolute differences between the histogram frequencies for consecutive luminance values as

$$f_{\Gamma,k} = \sum_{i=0}^{255} \left| \Gamma(i) - \Gamma(i-1) \right| W_k(i) \text{ for k = 7, 8, …, 12, where } \Gamma(-1) = 0. \quad (14)$$

The WDD features provide different ways of quantifying the variation, symmetry and distribution of gray values within the images.

3.5 TEXTURE FEATURES

These features are texture measures based on the Generalized Gray Level Spatial Dependence Models for Texture using the implementation given in [11]. The following features were computed with the luminance image for finding the gray level co-occurrence matrix with a radius of 1. This means that gray level co-occurrence was found for each pixel (i,j) over the 3x3 neighborhood of (i,j). Let Q represent the gray level co-occurrence, K represent the maximum intensity in the image with k = 1, …, K, and S provide the neighborhood count (maximum of 9=3x3) with s = 1, …, S. The features calculated from the luminance image include:

$$T1 = \frac{\sum_k \sum_s \frac{Q(k,s)}{s^2}}{D} \quad (15)$$

$$T2 = \frac{\sum_k \sum_s s^2 Q(k,s)}{D} \quad (16)$$

$$T3 = \frac{\sum_s \left[\sum_k Q(k,s) \right]^2}{D} \quad (17)$$

$$T4=\frac{\sum_k \sum_s Q(k,s)^2}{D} \tag{18}$$

$$T5=\frac{-\sum_k \sum_s Q(k,s)\log_{10}(Q(k,x)/D)}{D} \tag{19}$$

$$T6=\frac{\sum_k \sum_s (k-m_r)(s-m_c)(Q(k,s)/D)}{o_r o_c} \tag{20}$$

$$T7=\sum_k \sum_s (k-s)^2 (Q(k,s)/D) \tag{21}$$

$$T8=\sum_k \sum_s (Q(k,s)/D)^2 \tag{22}$$

$$T9=\sum_k \sum_s \frac{\left(Q(k,s)/D\right)}{1+|k-s|} \tag{23}$$

For the expressions above $D=\sum_k \sum_s Q(k,s)$; m_r, and m_c are the row and column means for Q, respectively; o_r and o_c are the row and column standard deviations for Q, respectively. The features above include entropy, contrast, homogeneity, correlation, energy uniformity, and inverse difference moments. The goal with these features is to provide rotation and size invariant global texture measures using the entire image as the region of interest for these feature calculations.

3.6 CLASSIFICATION ALGORITHMS

In order to evaluate the discrimination capability of the different features groups, radial clustering, and a modification for a gradient density function clustering-based [12]

classification techniques were examined. For both approaches, the each category of the image set was partitioned using 90% of the images within the category for the training set and the remaining 10% of the images within the category for the test set. Feature normalization was performed using a fuzzy set approach. Using the training set for each feature, the minimum and maximum feature values, denoted as m and x, respectively are determined. Let A denote the fuzzy set for a particular feature. The membership function for A is labeled as $\mu_A(p)$ for feature value p and is defined as follows:

$$\mu_A(p) = \begin{cases} 1 \text{ if } p > 0.95\ x \\ (p - m)/(x - m) \\ 0 \text{ if } p < m \end{cases} \tag{24}$$

μ_A is found for all features for each category from the training set of images. All feature values for each training and test set image are normalized using μ_A found for the respective features. The normalized training feature vectors are used to find cluster centers for the radial clustering and gradient density function clustering-based methods.

The radial clustering method determines cluster centers from the training data for each modality category by specifying an intra-cluster distance, i.e. radius for the cluster, δ. For each category, cluster centers are determined from the training data using the following steps. First, select an initial cluster center as the first normalized training feature vector for a specified category. Second, for the next normalized training feature vector, compute the Euclidean distance to the initial cluster center. If the distance is greater than δ, create a new cluster and initialize its cluster center as the current normalized feature vector. Otherwise, select the cluster for membership for which the feature vector has a minimum Euclidean distance. The cluster center for the selected

cluster is updated as the mean feature vector of all feature vectors belonging to the cluster. Third, repeat step 2 for all training feature vectors for the specified category. Fourth, repeat steps 2 and 3 until the cluster centers do not change for an iteration through the training data or a specified number of iterations through the training data, whichever requires fewer iterations through the training data.

The gradient density function clustering-based technique (also referred to as mean shift clustering method) is implemented based on the algorithm presented in [12] with revision given in [13]. This technique automatically determines the number of cluster centers from the training set and is applied to test set for classification into the different categories. This technique has been shown to provide improvements in classification capability over other clustering methods such as K-means [14]. As part of this technique, there is a bandwidth parameter (α) which is used for data partitioning for cluster and number of cluster determination.

4. IMPLEMENTATION DETAILS

Initially, all the features and classification algorithms were developed in Matlab. Then the features which gave good classification rates were converted Java and a library of features was created which could be used by the open source data mining tool RapidMiner. This section explains in detail the code conversion process from Matlab to Java and then the process of building of plug-ins which could be used by RapidMiner. The challenges faced during code conversion and some of the design issues are also pointed out in this section. The development environment consisted of following software tools and languages:

1. Eclipse IDE for Java
2. Java Development Kit (JDK) 5
3. Java Runtime Environment (JRE) 1.5
4. Java Advanced Imaging (JAI) 1.1
5. Apache Ant
6. RapidMiner 4.2

Most of the features were developed by using the various histograms of the image. The next section explains the Histogram class in detail.

4.1 HISTOGRAM CLASS

In this research, five different types of histograms have been used for feature calculations. The different types of histograms are as follows:

1. Luminance histogram

2. Histogram of L space from LUV color image
3. Color histogram
4. Smooth HSV histogram
5. Unsmooth HSV histogram

In order to use the external libraries such as JAI, they are added to the classpath of program or project in which they are going to use the external libraries. JAI contains a class Histogram, objects of this class can be used to generate various types of histograms like the luminance histogram and color histogram. But in order to have more control on the way image histogram is generated a class Histogram was written externally which had methods to generate different types of histograms. The class diagram for the class Histogram is shown below in Figure 4.1:

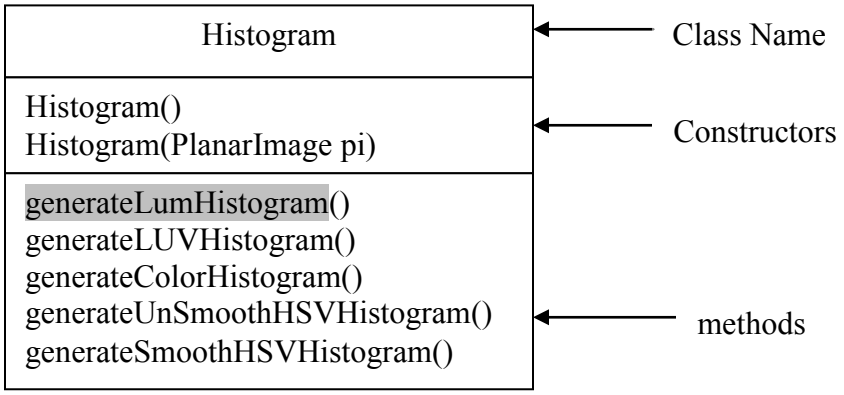


Figure 4.1. Class diagram for Histogram class

Histogram class has two constructors and five methods. The first constructor Hsitogram() is the default constructor and is a no parameter constructor. The second constructor Histogram(PlanarImage pi) is the second constructor for the class and this constructor takes object of type PlanarImage which is a class in JAI library. PlanarImage holds the image data. Pixel values of an image can be read from PlanarImage objects. This class also contains more information about the image like the height, width and the number of planes of the image. The constructor uses the PlanarImage object to initialize various local variables like the imgHeight, imgWidth and the numOfPlanes. The constructor also initializes the various histograms of the image. The class then has five methods to generate five different types of histograms. The signatures of each of the methods and method are described below.

The first method is for generating the luminance histogram. The signature of the method is as shown in equation 1:

public double[] generateLumHistogram() (1)

This class is of the public type. Objects of the class can directly access the method. The return type of the method is array of double. The array contains the luminance histogram of the image. In order to generate the histogram the luminance value at each pixel is calculated using equation 2 and then using the luminance value the particular bin is updated based on rounding the luminance value computed from the red, green and blue value at each pixel.

$$lum = 0.299\ red + 0.587\ green + 0.118\ \ blue \quad (2)$$

The luminance of the gray scale image is the same as the pixel value at that particular equation.

The second method in the class is for generating the histogram for L space in LUV color image. Signature of the method is shown in equation 3.

```
public double[] generateLUVHistogram()
```
(3)

Access type is public and hence the method can be accessed directly by the object of the class. The return type of the method is an array of double. The array contains the L space histogram. In order to generate the L space histogram the image is first converted to LUV space and then the histogram is generated by updating the required bins in the histogram.

The third method in the class is to generate the color histogram for an image. The signature of the method is shown in equation 4.

```
public double[][][] generateColorHistogram()
```
(4)

The method can be accessed directly by the objects of the class because the access type is public. This method has a return type as 3 dimensional array double data type 'double'. This array holds the count of red green blue pixels. For black and white image

the pixel values for all the three colors are same and hence the count of each color in each bin will be the same.

The fourth and the fifth method of the class are used for generating the unsmooth and smooth versions of HSV histogram. The two versions of HSV histograms are explained in Section 3.3.3. Equations 5 and 6 give the signatures of the respective methods.

public double[] generateUnSmoothHSVHistogram() (5)

public double[] generateSmoothHSVHistogram() (6)

The access type for both the methods is public and hence they can be accessed directly by the object of the class. Also the return type of the method is an array of double which holds the respective histograms for the image.

Histogram class is used in calculating many features of an image. The histogram class along with different features was then packed into a jar file. Following three jar files were formed:

1. image_features.jar: this file contains classes for calculating various image features.
2. groupfeatures.jar: this file contains classes for generating various groups of features. The different groups of features are explained in Section 5.1 and 5.3.
3. classification.jar: this jar file contains classes which perform classification of images.

jar file were generated using apache ant compile which is embedded in eclipse. The procedure for generating jar files is explained in detail in Section 4.3. These jar files were

then used to create operators for RapidMiner. The procedure for creating operators for RapidMiner and how to install and use these operators has been in explained in detail in next sections.

4.2 EXTENDING RAPIDMINER

In order to use RapidMiner for feature calculations, plug-ins which are compatible with RapidMiner are created. Plug-ins compatible with RapidMiner contains operators which actually work on the data to be mined. In order to write operators, 'rapidminer.jar' needs to be added to the classpath of the project. After adding the jar file to the classpath, operator class to be extended is chosen [15]. Also, in order to use the feature libraries that have been created are added on the classpath for the project.

There are four different types of operator class described as below:

- com.rapidminer.operator.Operator: this class just performs actions on the inputs and gives the output.
- com.rapidminer.operator.OperatorChain: this class has the capability of handling the operator inside it.
- com.rapidminer.operator.learner.Learner: this is an interface that is implemented while the operator will be in learning scheme.
- com.rapidminer.learner.AbstractLearner: this is a class which has the similar functionality as that of Learner interface.

In order to create custom operators, com.rapidminer.operator.Operator class has been extended. As it is an abstract class, all the unimplemented methods of the class are implemented in the child class. The following section explains all the methods in details.

The operator which calculates the mostFrequentGray as explained in Section 3.3.1 is used as an example to explain the operator class. The name of the operator class is LMostFrequentOperator.

4.2.1. Operator Class Details. Operator class being an abstract class, the methods needs to be implemented. The following methods were overridden by the child class.

1) One argument constructor. This constructor takes the in object of class OperatorDescription. It then calls the constructor of the super class using the argument. The body of the constructor is shown below in Figure 4.2:

```
public LMostFrequentOperator(OperatorDescription description) {
            super(description);
            // TODO Auto-generated constructor stub
      }
```

Figure 4.2. Body of the constructor for the class

2) Class[] getInputClassses(): This class specifies the number and classes which will be used as the input classes by the operator. The classes can be only of com.rapidminer.operator.IOObject type. The operator does not have any input classes of the type IOObject and hence this method will return null value as shown in Figure 4.3:

```
public Class[] getInputClasses() {
            // TODO Auto-generated method stub
            return null;
      }
```

Figure 4.3. Body of getInputClasses

3) Class[] getOutputClasses(): This class specifies the number and classes which are returned as output of the operator. The output class or classes have to be of the type com.rapidminer.operator.IOObject. In the example discussed the class of the type IOObject has been created and the name of the class is LMostFrequentResult and is described in detail in Section 4.2.2. The body of the method is shown in Figure 4.4:

```
public Class[] getOutputClasses() {
            // TODO Auto-generated method stub
            return new Class[]{LMostFrequentResult.class};
      }
```

Figure 4.4. Body of the method getOutputClasses

4) List<ParameterType> getParameterType(): This method specifies the name and the type of the parameters that may be queried by the operator. Parameter values

are generally passed by the user as input to the operator. In this particular operator image file name is used as the parameter to the operator. The body of the method is shown in Figure 4.5:

```
public java.util.List<ParameterType> getParameterTypes()
      {
            java.util.List<ParameterType> types =
super.getParameterTypes();
            types.add(new
ParameterTypeFile(Image_File_Name,"image_file_name","jpg",false)
);
            return types;
      }
```

Figure 4.5. Body of the method getParameterTypes

5) IOObject[] apply(): this is the main method of the operator. This is the method where the main processing takes place. In case of LMostFrequentOperator the operator first creates an object of class "LMostFrequentGray" and then uses the object to get the mostFrquentGray feature of the image. Then an object of class LMostFrequentResult is created the mostFrequentGray value is set in the result object and the object is returned. All this is depicted in Figure 4.6:

```
public IOObject[] apply() throws OperatorException {
            // TODO Auto-generated method stub

            String imgFileName=
getParameterAsString(Image_File_Name);
            mostFrequentGray = new
LMostFrequentGray(imgFileName);
                LMostFrequentResult result = new
 LMostFrequentResult(mostFrequentGray.calculateLMostFrequentGr
                                ay());

            return new IOObject[] {result};
      }
```

Figure 4.6. Body of apply() method

4.2.2. ResultObject Class Details. The return object of the operator is object of type IOObject. In order for the result object to be of IOObject type the ResultObjectAdapter class is extended. This section explains in detail the LMostFrequentResult which is used as an output object by the operator described in the previous section. For each operator that has been developed the corresponding results objects have also been developed. In order for the result object to be of the type IOObject the ResultObjectAdapter class has been extended and the necessary methods have been implemented alongwith some additional methods. The methods and constructors of the LMostFrequentResult are described as follows:

1. The class has two constructors. One is default and no parameter constructor. The next constructor is a two parameter constructor and the two parameters are the mostFrequentVlaue and imgFileName. These parameters are used to assign the values to the local variables of the class. The constructor with two arguments is shown in Figure 4.7:

```
      public LMostFrequentResult(double mostFrequent, String
imgFileName)
      {
            this.msotFrequentLevel = mostFrequent;
            this.imgFileName = imgFileName;
      }
```

Figure 4.7. Two parameter constructor for the class LMostFrequentResult

2. double getMostFrequentLevel(): is the method which returns the value of the mostFrequentLevel. The body of the method is shown in Figure 4.8:

```
public double getMostFrequentLevel()
      {
            return this.msotFrequentLevel;
      }
```

Figure 4.8. Body of method getMostFrequentLevel

3. String getName(): This is the first method to be overridden. This method returns the name of the class in form of a String object. The body of the method is shown in Figure 4.9:

```
public String getName() {
            // TODO Auto-generated method stub
            return Tools.classNameWOPackage(this.getClass());
      }
```

Figure 4.9. Body of the method getName

4. Component getVisualizationComponent(): This method is overridden and as the name of the class suggest it returns the graphical object in which the result will be displayed. This method is called by RapidMiner after the operator is finished processing and the output needs to be displayed. The method uses Swing objects in order to display the results. The body of the method is shown in Figure 4.10:

```
public Component getVisualizationComponent(IOContainer arg0) {

            // TODO Auto-generated method stub
            String str = toResultString();

            JEditorPane resultText = new JEditorPane();
        resultText.setContentType("text/html");
        resultText.setText(str);

      resultText.setBorder(javax.swing.BorderFactory.createEmptyBorde
r(11, 11, 11, 11));
        resultText.setEditable(false);
        resultText.setBackground((new JLabel()).getBackground());
        return new ExtendedJScrollPane(resultText);
            }
```

Figure 4.10. Body of the method getVisualizationComponent

5. String toResultString(): This method converts the result object to string object. In LMostFrequentResult class this method converts the mostFrequentLevel to string object and returns it. The body of the method is shown in Figure 4.11:

```
public String toResultString() {
            // TODO Auto-generated method stub
            String result = new
String(Double.toString(msotFrequentLevel));
            return result;
      }
```

Figure 4.11. Body of the method toResultString

6. boolean isSavable(): This method returns true or false depending on whether the result object can be saved or not. In this example result object to be saved can be saved, and hence, true is retturned. The body of the method is shown in Figure 4.12:

```
public boolean isSavable()
      {
            return true;
      }
```

Figure 4.12. Body of method isSavable

7. void save(): This method is overridden in case the result object has to be saved. Since the isSavable() method returns a true value this method is overridden. In this method the mostFrequentLevel and imgFileName is written on to a text file and save the text file with a name given specified by the user. The body of the method is shown in Figure 4.13:

```
public void save(File file) throws IOException
	{
		PrintWriter out = null;
		try
		{
			out = new PrintWriter(file);
			out.print(this.imgFileName + " ");
			out.print(toResultString());
		} finally {
			if(out != null )
				out.close();
		}
	}
```

Figure 4.13. Body of the method save

These are the minimum requirements to create custom operators and result objects for the same. This operator along with other operators is converted to a jar file which is used as a plug-in with RapidMiner. In order to use the operators, they first need to be declared to RapidMiner. This is done by adding operators.xml file to jar file containing these operators. Operators.xml file contain the details about the operator. The operators.xml file is explained in detail in the next section.

4.2.3. Operators.xml in Detail. In order to declare the custom operator, the following conditions have to be complied with:

- name: every operator should have a meaningful and unique name.
- classname: the fully qualified class name of the operator.
- description: a short description of the operator and the task it performs.
- group: the name of the group in which the operator belongs. This group name is used for organizing the operators in groups while displaying them in GUI.
- icon: this is an option filed and helps in identifying the operator with ease.

An xml file is formed with all these entries, the name of the file is operators.xml and this file is then packed in side the jar file. The operator.xml file with LMostFrequentOperator is shown in Figure 4.14:

```
<operators>
      <operator
            name = "L_Most_Freq"
            class =
"com.rapidminer.image.luv.operators.LMostFrequentOperator"
            description = " Gets the most frequent L level"
            group = "ImageOperators.LUV" />
</operators>
```

Figure 4.14. operators.xml

The name “L_Most_Freq” appears on the RapidMiner GUI. The class variable specifies the path to the class of the operator. The description appears in the RapidMiner GUI when the cursor is moved over the name of this particular operator and this operator

is found under ”ImageOperators.LUV ” group. This file has to be embedded into the jar file in order for the plug-in to work with RapidMiner. The procedure for making any kind of jar file is explained in the next section.

4.3. PACKAGING INTO JAR FILES FOR RAPIDMINER

In order to package all the image features into one file or make a library of all image features and also to make a plug-in for RapidMiner, all the class files and libraries are to be packaged into a jar file. ‘jar’ stands for Java Archive. Apache Ant has been used to compile and build jar files. The apache Ant compiler comes with eclipse IDE. The compiler reads the build.xml file and performs the necessary actions. The build file used for packaging the operators for RapidMiner and making a plugin is explained in detail in this section. The Figure 4.15 shows the build file used for packaging the operators.

The Ant compiler looks for build.xml file. The Ant compiler searches for the target specified while giving the Ant run command in build.xml. If no target is specified at the runtime, then the compiler look for “build” target. In the example the build target depends on “compile” target, and hence, the compiler shifts the execution to the compile target. The compile target depends on copy-resources which in turn depend on the init target and hence the compiler first executes the init target.

In the init, target two folders have been created, one is build and the second one is final folder. After making the two folders the execution shifts to copy-resources. In this target all the resources i.e. Java files and operators.xml file are copied to the build folder. Then the compiler executes the compile target in which first the classpath is set and all the external libraries required for compiling the source code are added to the classpath.

After setting the classpath, the compiler compiles all the Java files and the finally the execution returns to the build target where the all the class files, the source code, operators.xml and all external libraries are packaged into a jar called Rapidminer-image.jar.

The contents of the jar file can be found either by using the unjar command or simply by using and unzipping software. Files can also be added externally if need occurs. In order to use the plug-in, the jar file needs to be pasted it into the plugins folder in RapidMiner home directory and then start RapidMiner. The operators packaged into the jar file will appear in the GUI. The next section explains how to use the operators in RapidMiner.

4.4 USING OPERATORS IN RAPIDMINER

This section explains how to use the RapidMiner plugin for calculating groups of features. The different groups of features are explained in detail in next chapter. In order to use the plug-in, the RapidMiner-GroupFeatures.jar needs to be pasted in the plug-ins folder of RapidMiner home directory. After copying the 'RapidMiner-GroupFeatures.jar' file into the 'plugins' folder of RapidMiner, RapidMiner is started and the home screen looks as shown in Figure 4.16:

```
<project>
       <description>
           Build file for the image operators RapidMiner plugin
         </description>
       <property name="src"           location="src"/>
       <property name="build"   location="build"/>
       <property name="final"         location="final"/>
       <path id="classpath">
           <pathelement location="${java.home}/jre/lib/rt.jar"/>
           <pathelement location="${java.home}/../jre/lib/rt.jar"/>
           <pathelement location="${java.home}/lib/tools.jar"/>
           <pathelement location="${java.home}/../lib/tools.jar"/>
           <pathelement location="${java.home}/classes"/>
              <pathelement location="lib/image_features.jar"/>
              <pathelement location="lib/rapidminer.jar"/>
         </path>
        <target name="build" depends="compile">
       <unjar src="${build}/lib/image_features.jar"
dest="${build}/image_features"/>
   <jar jarfile="${final}/Rapidminer-image.jar" basedir="${build}">
       <fileset dir="${build}/image_features" includes="**/*"
excludes="META-INF/*"/>
       </jar>
 </target>
        <target name="init">
              <echo message="init"/>
           <mkdir dir="${build}"/>
              <mkdir dir ="${final}"/>
         </target>

         <target name="compile" depends="copy-resources"
description="Compile all java files.">
              <echo message="using java version ${java.version}"/>
           <javac  destdir="${build}">
             <classpath refid="classpath"/>
           <src path = "src"/>
              <include name="**/*.java"/>
           </javac>
         </target>
       <target name="copy-resources" depends="init">
           <copy todir="${build}/src">
             <fileset dir="src"  includes="**/*"/>
            </copy>
         </target>
</project>
```

Figure 4.15. build.xml file

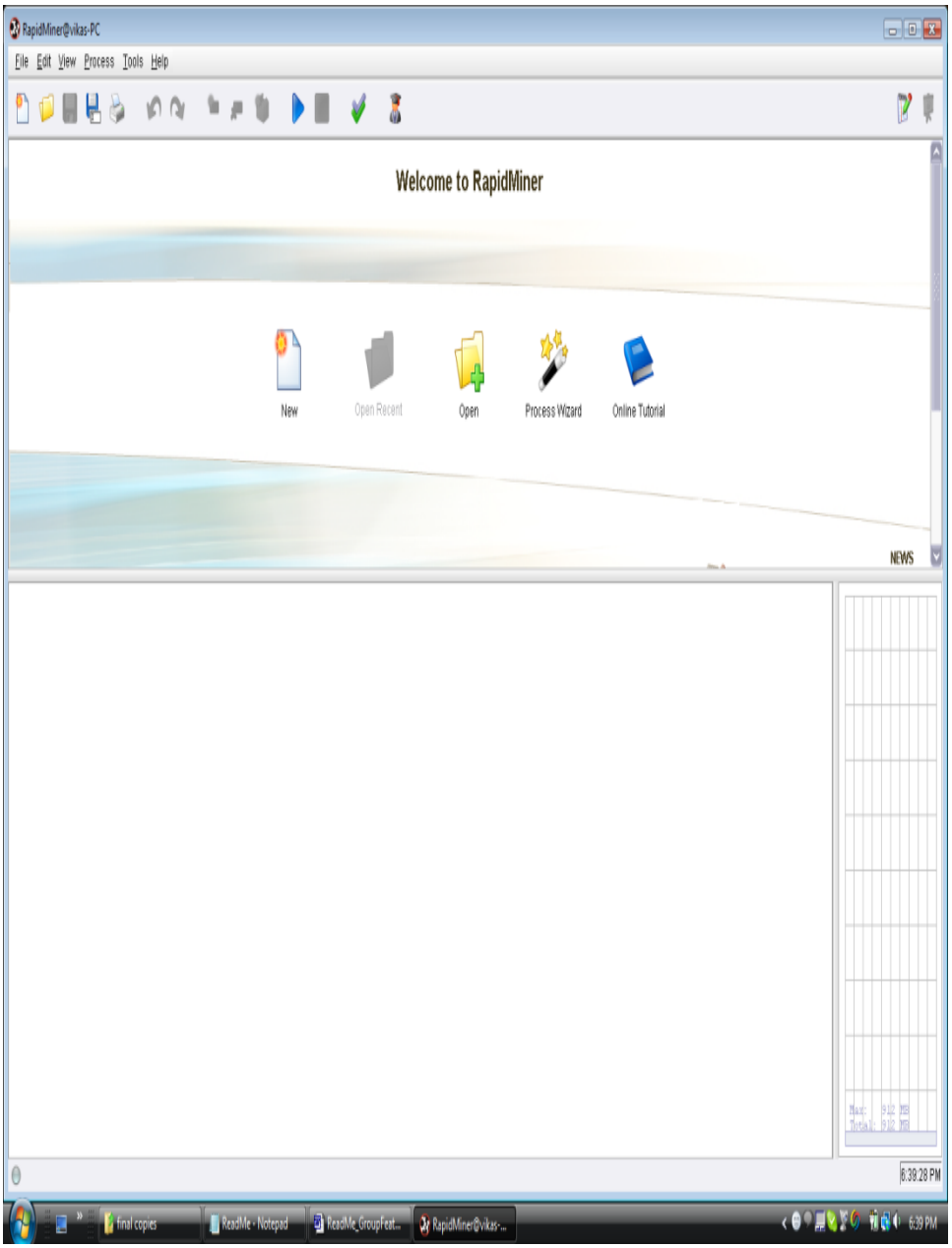


Figure 4.16. RapidMiner home screen

A new process is started by selecting the 'new' icon from the home screen. The output of which is shown in Figure 4.17:

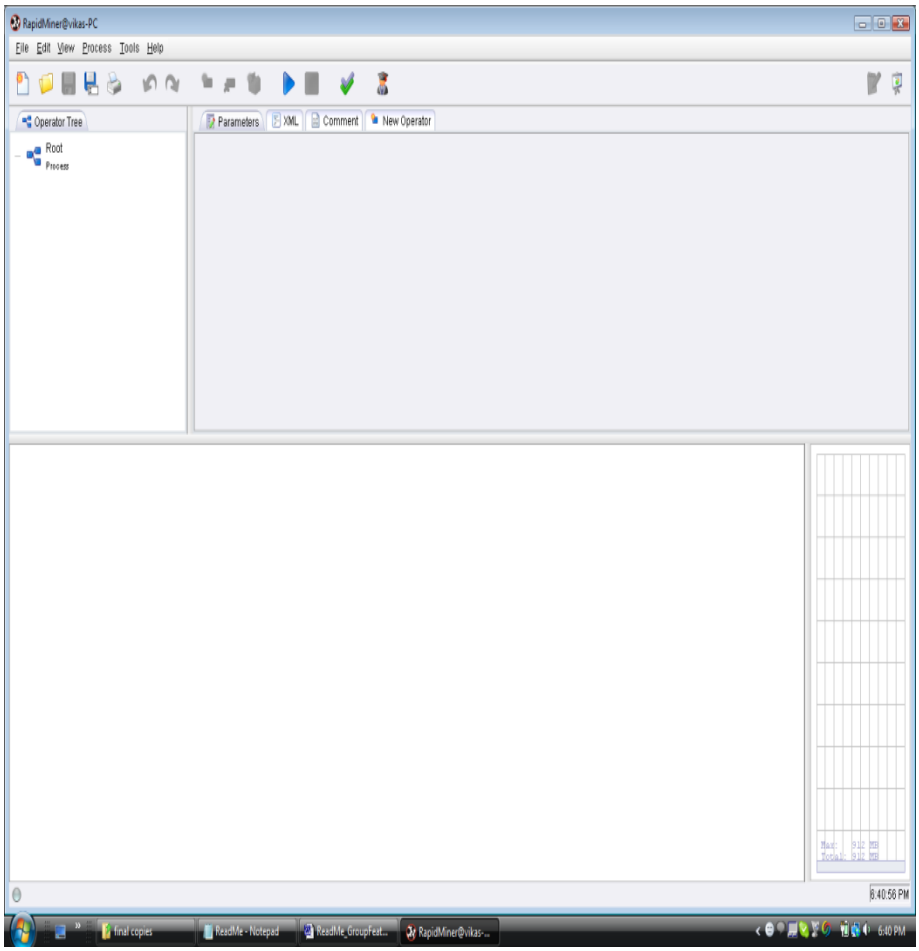


Figure 4.17. RapidMiner screen on starting a new process

To calculate the groups of features, the required operator is laoded. This is done by right clicking on the 'Root Process' and then selecting new operator and navigating all the way down to 'Group_Feature' and then to the required features. This is shown in Figure 4.18:

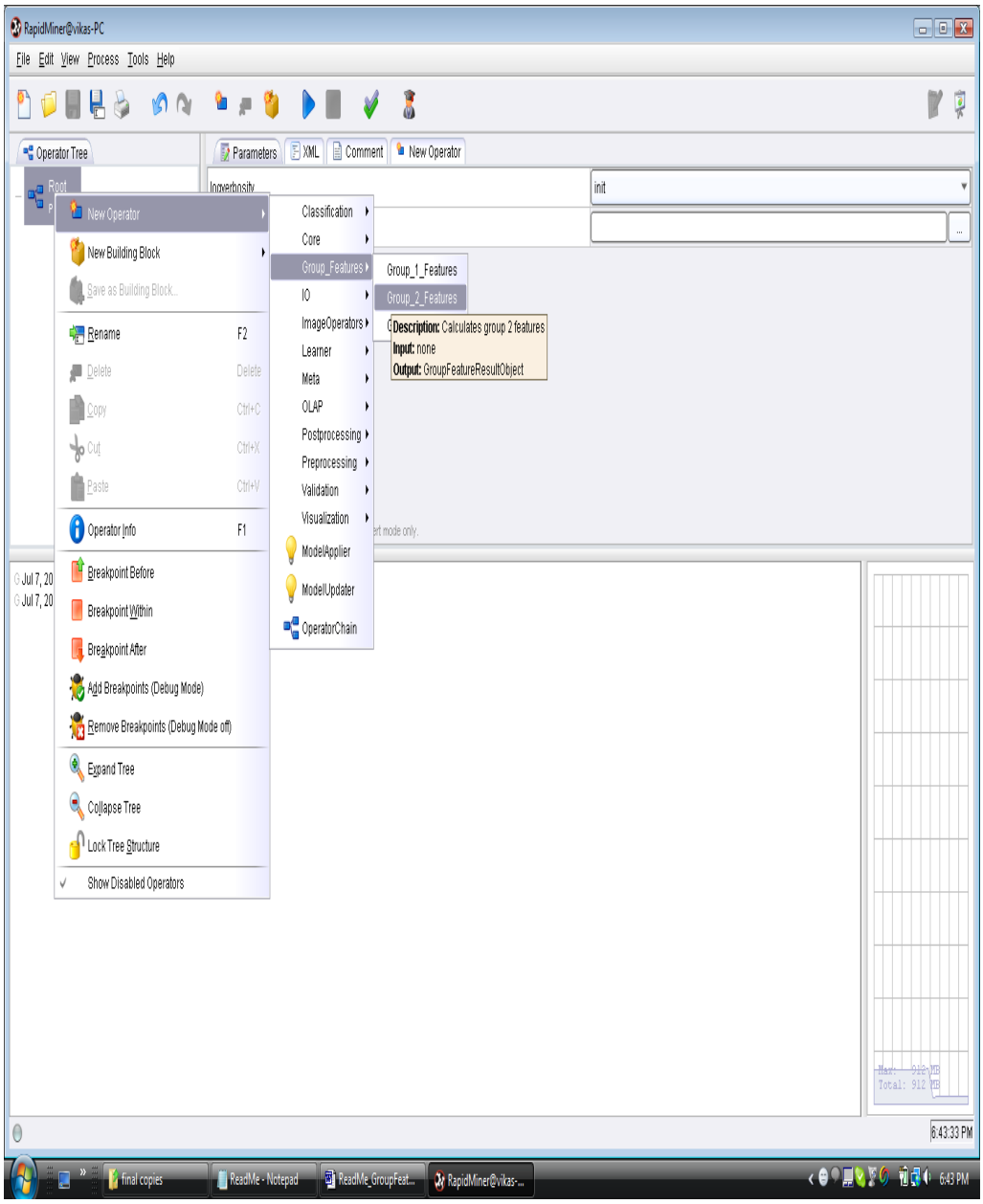


Figure 4.18. Screen while loading an operator

After selecting the required feature, it appears below the 'Root Process'. Upon clicking on the feature, it is observed see that the screen changes and now it shows two

text areas which are used as input fields to for the group feature calculation operator. This is shown in Figure 4.19:

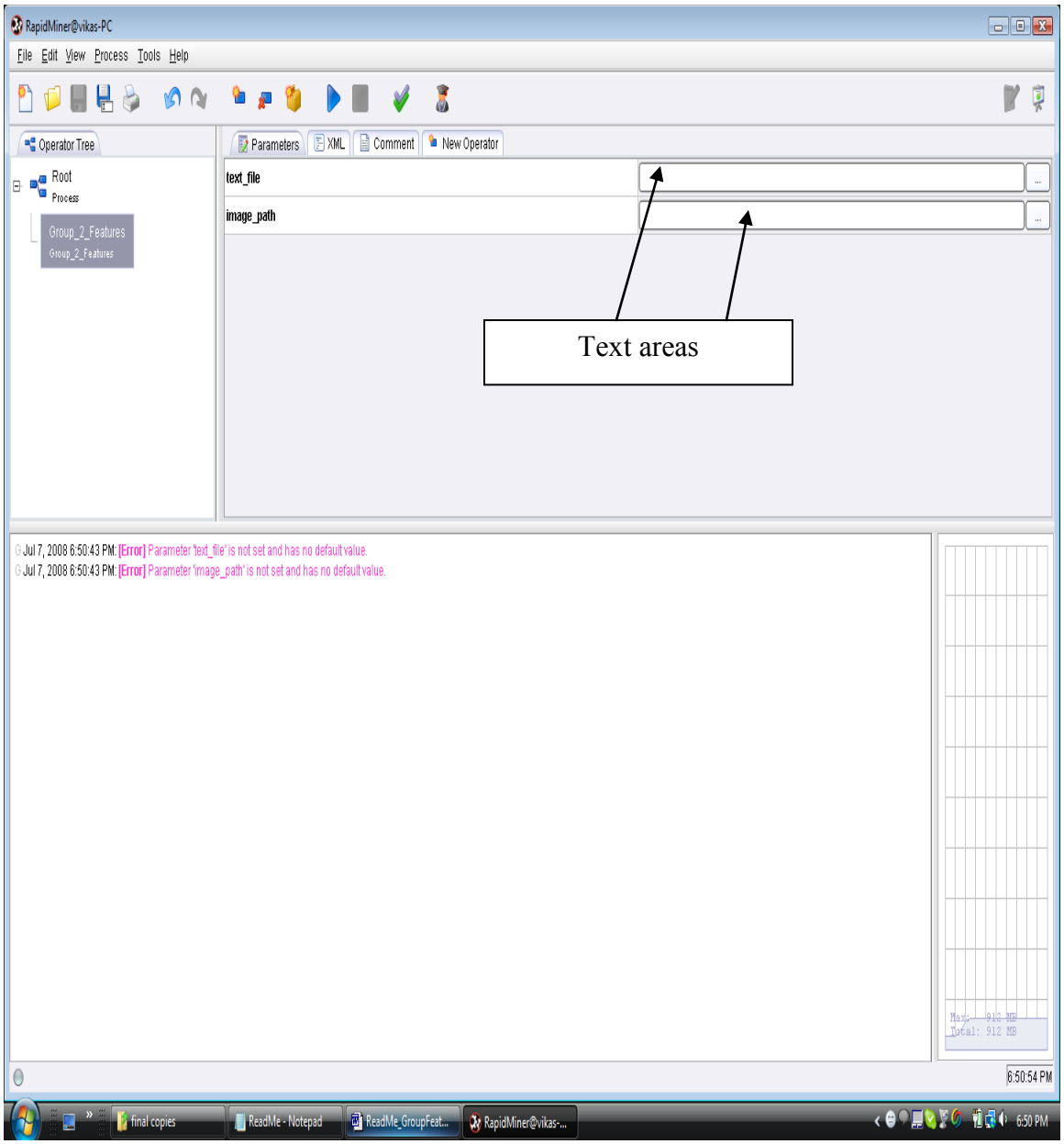


Figure 4.19. RapidMiner screen after loading the operator

The first input field is for a text file. The text file should contain the names of all the images for which feature values have to be calculated. The names of the files should be stored in the text file without the '.jpg' extension. The operator will process all the images mentioned in the text file and so any number of images can be processed at once. Also it should be noted that all these files should be stored in one directory as the second input for the operator is the path to the directory where all these images exist. After selecting the valid inputs for both, the fields can be validated by clicking on the 'correct' button on the top of the screen. If there is some error, it will be pointed out in the console as shown in Figure 4.20. If the inputs are valid, which can be seen on the console as shown in Figure 4.21, the operator can be started by clicking on the triangular button as shown in Figure 4.21:

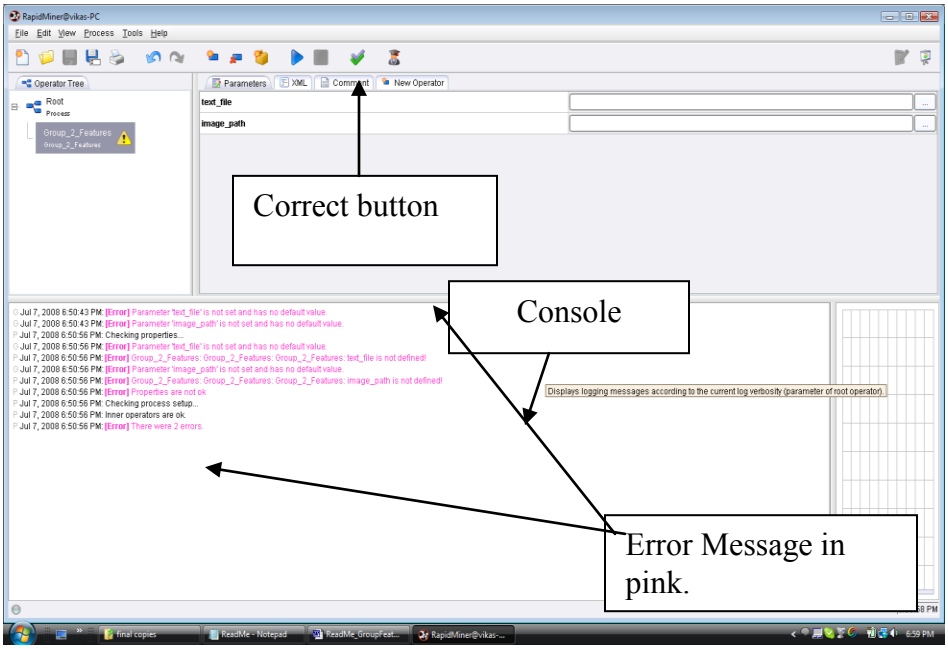


Figure 4.20. Errors during the validation process

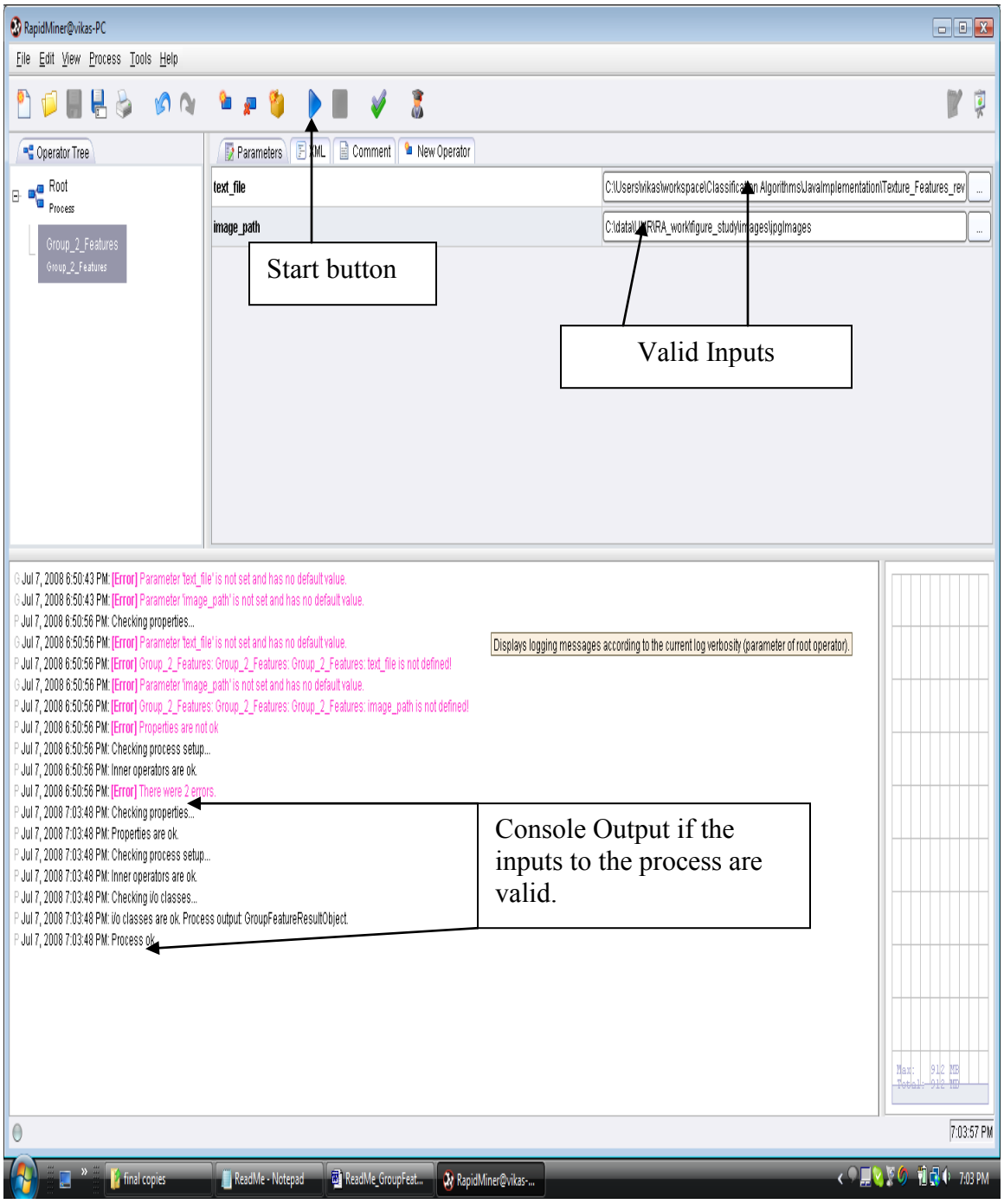


Figure 4.21. Console output for a validated process

On starting the process, the console changes as shown in Figure 4.22 and then once the process has finished, the output is displayed on the output screen as shown in

Figure 4.23. The output of all the groups of features consists of image file name followed by a default class label '0' which is followed by the feature values. The output can be saved in a text file by using the save button on the output screen.

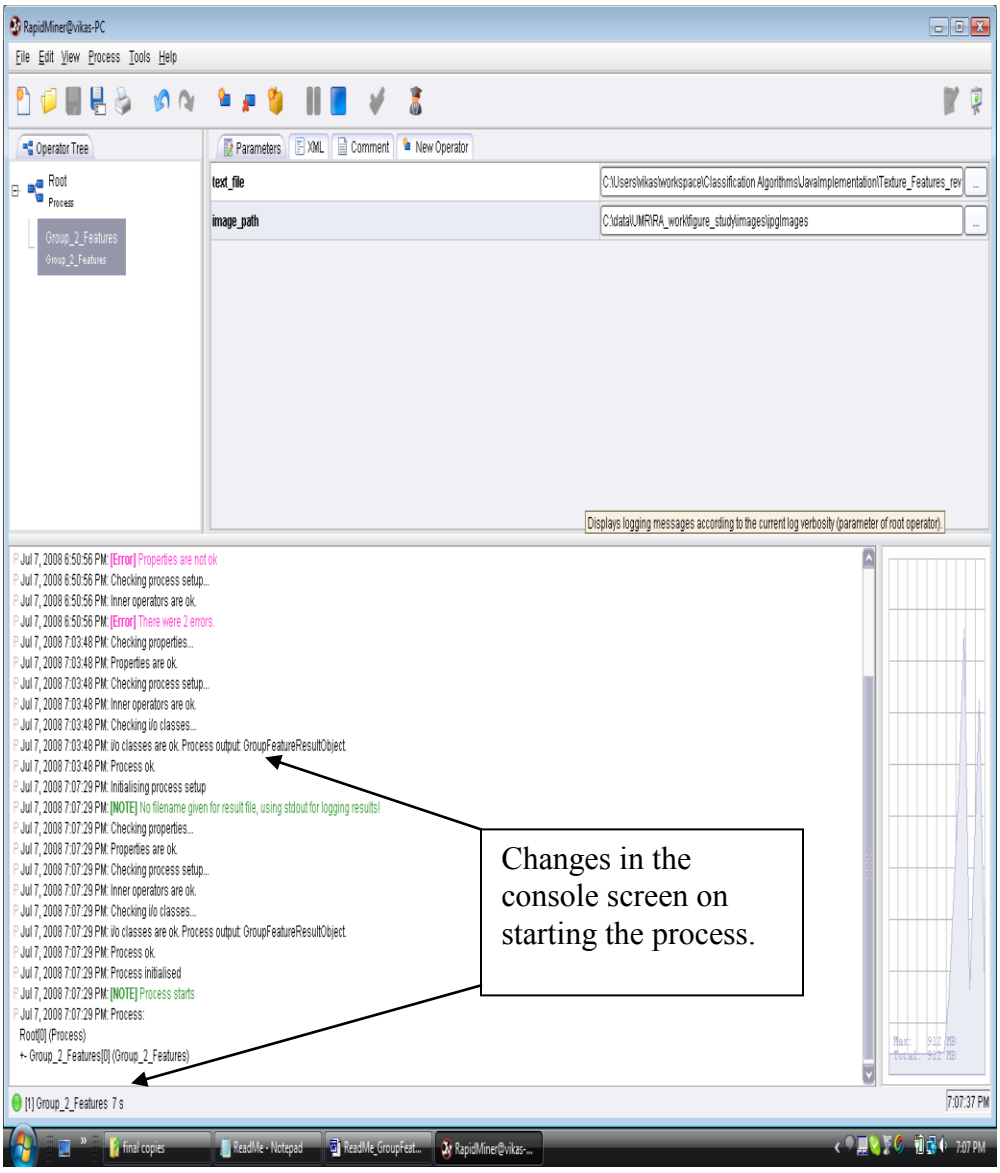


Figure 4.22. Screen when the process starts

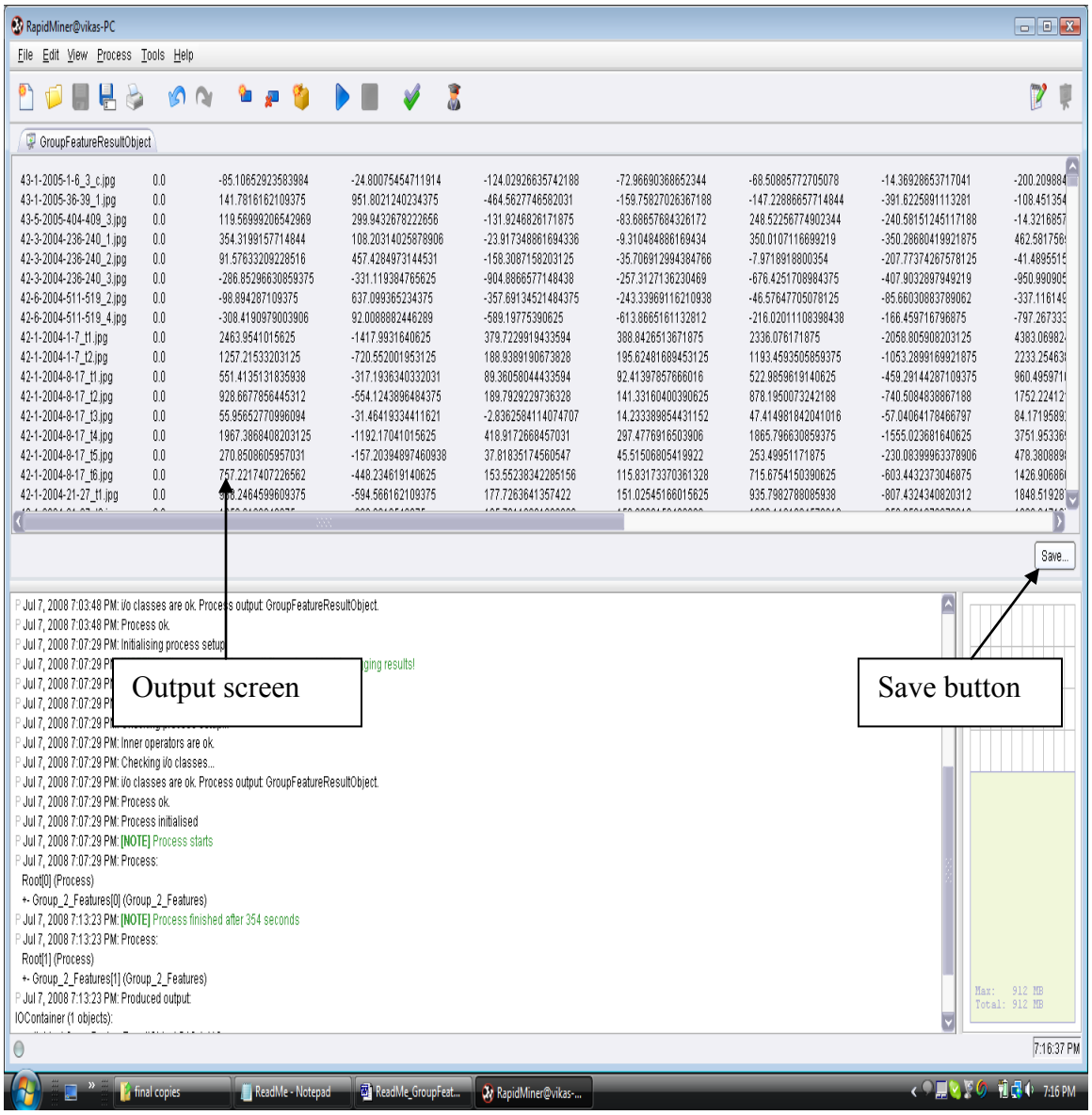


Figure 4.23. Screen after the process has finished

4.5. ISSUES DURING CODING

To fasten the process of feature development initially, features were developed in Matlab. The features which gave good classification rates were then converted into Java and finally into a plug-in which could be used by Rapidminer. In Matlab, there are many

inbuilt functions which fasten up the process of feature development while it is needed to code each and every method in Java. Some of the issues and the way they were handled are explained in following sections.

4.5.1. Matlab Data Types Vs Java Data Types. Most of the data types which are available in Matlab are also available in Java except one that is unsigned versions of the primitive data types. For the calculation of the luminance histogram in Matlab, the image pixels were handled with the uint data type while in Java it was handled with double data type. This led to difference in the histograms generated by Matlab and Java. Equation 1 is used to calculate the luminance histogram.

$$lum = 0.299\ red + 0.587\ green + 0.118\ \ blue \quad (1)$$

While generating the histogram in Java Math.round() method was used in order to get the same histogram as that generated by Matlab code. The modified equation is given in equation 2.

lum = Math.round(0.299*red) + Math.round(0.587*green) +Math.round(0.118* blue) (2)

4.5.2 Array Index. In Matlab, array index always starts from 1 while in Java the array index always starts from 0. Many times array indices are used while in equations and at other times array indices are generated with some kind of calculations. In order to have consistent result while coding in Java and Matlab, the array index problem needs to be handled. In many cases this problem was handled mathematically by modifying the

equation or at times subtracting 1 before using the variable as index of an array. But, these methods are not full proof and do not work perfectly. In order to solve the problem, all the arrays in Java were created of the required size + 1 elements and the first element that is with the index was initialized to null value. This approach works fine in most of the cases and is easier to implement and there is no overhead of maintaining the correct index in Java by subtracting one in the end or by taking care programmatically.

4.5.3. Matlab Inbuilt Functions. Matlab has many inbuilt functions which perform operations on array and matrices. Functions like mean and std are useful while calculating mean or standard deviation of an array. The transpose of a matrix can also be calculated by using the " ' " operator. On the other hand, Java does not have any such inbuilt methods or functionalities. A program or code has to be written for each and every such function. Matlab supports matrix operations while in Java a matrix has to be declared as a 2 dimensional array and for every operation done on a matrix, a program/ code has to be written for it. Use of Matlab's inbuilt functions make programs more reliable and bug free as compared to Java where errors could occur at the boundary values of a for loop or mishandling of 2 dimensional arrays.

4.5.4. Design Issues. This section describes two design issues. This design of the Histogram class is not optimal and uses a lot of memory space which can be saved for other functionalities. Histogram class is used for calculation of various features. Most of the features just require one type of histogram to be generated at a time. The histogram class constructor always initializes all the three types of histogram irrespective of the type of histogram required for feature calculation. All the histograms are arrays of double and the color histogram is a 3 dimensional array. So initializing all the arrays when just one is

needed, leads to lot of memory wastage and hence affecting the performance of the entire system.

As mentioned earlier, there is one result object for each operator. But, many operators just return one double value as output and hence one result object class which will contain one variable of type double can be used as result object for all the operators that return only one value as output.

5. EXPERIMENTS AND RESULTS

5.1 GROUPS OF FEATURES

This section explains the various features that were combined together to form different groups of features for classification of images. The features have already been described in detail in Section 3. Group 1 consisted of following global and histogram based features:

1. stdColRed, stdColGreen, stdColBlue
2. percentG
3. percenWhite
4. mostFrequentGray
5. percentMostFrequent
6. avgGray
7. stdGray
8. sqrtArea
9. percentMiddle
10. sumDiff
11. percentDark
12. colorCount, colorCount1, colorCount2
13. fractDim1 and fractDim2.

Group 2 consisted of 12 WDD features over the luminance histogram as explained in Section 3.4. Group 3 consisted of 9 texture features as explained in Section 3.5.

5.2. EXPERIMENTS PERFORMED

Three feature files were generated for each of the feature groups. 90% of the images were used to train the radial basis classification algorithm and 10% of the images were used as test images. Experiments were performed on 25 random test and train sets. Table 5.1 presents the test results using the radial clustering method for $\delta = 0.5$ for feature groups 1-3. The test results for each of the 25 randomly generated training/test sets are given with the mean and standard deviation for the respective feature groups. Table 5.2 provides the average number of cluster centers determined from the training data for $\delta = 0.5$ for each category over the 25 randomly generated training/test sets. These are the cluster centers used for classifying the image feature vectors into the different categories. Table 5.3 presents the average and standard deviation test results over the 25 randomly generated training/test sets based on the features computed for Groups 1-3 using the radial clustering method with $\delta = 0, 0.5, 1.0, 5.0, 10.0$, and 30.0.

Table 5.1. Test results using radial clustering method for $\delta = 0.5$ for feature groups 1-3 for 25 training/test sets are presented with mean and standard deviation

Training/Test Set	% Correct Test Image Results for All Categories – Group 1 Features	% Correct Test Image Results for All Categories – Group 2 Features	% Correct Test Image Results for All Categories – Group 3 Features
1	88.00	96.00	96.00
2	88.00	97.33	96.00
3	89.33	96.00	92.00
4	88.00	96.00	98.67
5	82.67	90.67	90.67
6	86.67	94.67	96.00

Table 5.1. Test results using radial clustering method for δ = 0.5 for feature groups 1-3 for 25 training/test sets are presented with mean and standard deviation (cont.)

7	88.00	97.33	97.33
8	80.00	92.00	92.00
9	92.00	96.00	96.00
10	82.67	93.33	94.67
11	85.33	98.67	96.00
12	86.67	96.00	92.00
13	92.00	96.00	93.33
14	80.00	96.00	92.00
15	84.00	93.33	93.33
16	80.00	94.67	93.33
17	86.67	96.00	97.33
18	85.33	97.33	93.33
19	81.33	94.67	89.33
20	80.00	94.67	92.00
21	93.33	94.67	94.67
22	89.33	97.33	96.00
23	88.00	96.00	94.67
24	93.33	98.67	97.33
25	80.00	96.00	92.00
Average	86.03	95.57	94.24
Standard Deviation	4.34	1.87	2.40

Table 5.2. Average number of cluster centers generated from radial clustering method for δ = 0.5 over 25 randomly generated training/test sets

Category	**Average Number Clusters – Group 1**	**Average Number Clusters – Group 2**	**Average Number Clusters – Group 3**
Chart / Graph	38.84	17.96	9.32
Drawing	39.88	23.80	15.48
Flowchart	4.56	6.00	4.68
Form	10.00	8.24	4.68
Histology	79.76	37.64	10.72
Photograph	146.76	28.72	17.08
Radiology	55.52	25.40	15.12
Table	12.00	12.48	6.60
Mixed	7.00	6.48	6.36

Table 5.3. Summary average and standard deviation test results over 25 training/test sets using radial clustering method for δ = 0, 0.5, 1.0, 5.0, 10.0, and 30.0

	Group 1	**Group 2**	**Group 3**
Average % Correct Test Results			
δ = 0	86.03	95.57	94.24
δ = 0.5	84.91	95.41	85.92
δ = 1.0	84.32	95.31	78.83
δ = 5.0	76.21	93.39	70.24
δ = 10.0	72.16	91.73	68.53
δ = 30.0	57.71	90.88	70.29
Standard Deviation Test Results			
δ = 0	4.34	1.87	2.40
δ = 0.5	4.21	1.93	4.90
δ = 1.0	3.93	2.18	6.27
δ = 5.0	5.96	3.23	4.42
δ = 10.0	6.35	2.55	4.75
δ = 30.0	9.16	2.82	5.79

After looking at the tables above several observations were made. Firstly, the Group 2 features computed by correlating the basis functions with the luminance histogram of the image provide the highest overall classification into the different categories. The average classification rate of Group 2 is 95.57 with a standard deviation of 1.87. From Table 5.3 it is clear that for Group 2 best classification rate achieved was 95. 57 for δ = 0 (nearest neighbor case) while it the lowest for δ = 30 with the classification rate = 90.88. Group 1 and group 3 features also give comparable results for δ = 0.5. The average classification rate for group 1 and 3 are 86.03 and 94.24, respectively. The Group 1 features consist of global image descriptors such as thresholding, color counting, statistical measures such as standard deviation and most

frequently occurring gray level, and Fractal dimension. As global measures, these features do not provide context to the distribution of gray level information within the images for the different categories. The Group 3 features provide texture measures for distinguishing the different modality categories. The Group 2 features quantify the luminance value distribution for the different categories based on correlating the luminance histograms with a set of basis functions to provide global measures for distinguishing the different categories. The experimental results show that the correlation features provide effective similarity measures for images within each category. Furthermore, experimental results show that clustering of these features yield strong discriminators for the different categories. Tables 5.4 – 5.6 show the average correct results over the 25 test sets in confusion matrix form for the features for Groups 1-3, respectively, for $\delta = 1.0$.

Table 5.4. Average confusion matrix test results for Group 1 for $\delta = 1.0$ from radial clustering method

Category	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	10.44	0.48	0.04	0	0	0	0	0.04	0
2	0.76	5.84	0	0	0.12	0.12	0.12	0	0.04
3	0.08	0	0.68	0.16	0	0	0	0.08	0
4	0	0.04	0.12	0.6	0	0	0.08	0.16	0
5	0	0.12	0	0	12	0.52	0.36	0	0
6	0	0	0	0	0.68	21.12	3.2	0	0

Table 5.4. Average confusion matrix test results for Group 1 for δ = 1.0 from radial clustering method (cont)

7	0.04	0.44	0	0	0.56	2.32	7.52	0.04	0.08
8	0.08	0	0	0	0	0	0	4.76	0.16
9	0	0.04	0	0.16	0.08	0	0.4	0.04	0.28

Table 5.5. Average confusion matrix test results for Group 2 for δ = 1.0 from radial clustering method

Category	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	10.44	0.48	0	0	0	0.08	0	0	0
2	0.48	6.36	0	0.04	0	0.12	0	0	0
3	0.16	0	0.68	0	0	0	0.16	0	0
4	0.04	0	0	0.68	0	0	0	0.28	0
5	0	0	0	0	13	0	0	0	0
6	0.12	0.44	0	0	0	24.4	0	0	0.04
7	0.04	0	0.12	0.04	0.2	0.08	10.48	0	0.04
8	0	0	0	0.44	0	0	0	4.56	0
9	0	0.04	0	0	0.08	0	0	0	0.88

Table 5.6. Average confusion matrix test results for Group 3 for δ = 1.0 from radial clustering method

Category	1	2	3	4	5	6	7	8	9
1	8.00	2.96	0	0	0.04	0	0	0	0
2	2.32	4.4	0	0	0.08	0.16	0.04	0	0
3	0	0	1.00	0	0	0	0	0	0
4	0	0	0.04	0.72	0	0	0	0.24	0
5	0	0	0	0	12.16	0.04	0.80	0	0
6	0	0.24	0.08	0	0.84	20.44	3.32	0	0.08
7	0	0	0	0	0.60	2.56	7.72	0	0.12
8	0	0	0.04	0.16	0	0	0.12	4.52	0.16
9	0	0.24	0.2	0	0.12	0.04	0.04	0.2	0.16

For comparison purposes, modality classification was performed over the same 25 training/test sets using the (mean shift) gradient density function clustering-based method. For the experiments performed using this method, the bandwidth parameter (α) was varied as α = 0.25, 0.5, 1.0, 5.0. Table 5.7 presents the average and standard deviation correct test results for Groups 1-3 using the gradient density function clustering-based method. Inspecting Table 5.7, the highest classification results for Groups 2 and 3 were obtained for α = 0.25, and the highest discrimination results for Group 1 was found for α = 0.50. Overall, the discrimination results obtained using the gradient density function clustering-based method were very similar but slightly lower than the radial clustering method for Groups 2 and 3. Group 1 results were slightly

higher for $\alpha = 0.5$ with 86.19% compared to $\delta = 0$ with 86.03%. The experimental results show the utility of both classification techniques for the experimental categories used for image modality.

Table 5.7. Summary average and standard deviation test results over 25 training/test sets using gradient density function clustering-based method $\alpha = 0.25, 0.5, 1.0, 5.0$

	Group 1	**Group 2**	**Group 3**
Average % Correct Test Results			
$\alpha = 0.25$	86.03	95.57	94.13
$\alpha = 0.5$	86.19	95.52	91.52
$\alpha = 1.0$	81.97	93.12	55.57
$\alpha = 5.0$	46.56	90.77	71.15
Standard Deviation Test Results			
$\alpha = 0.25$	4.34	1.87	2.52
$\alpha = 0.5$	4.20	1.88	3.26
$\alpha = 1.0$	4.17	3.58	5.56
$\alpha = 5.0$	6.99	3.08	5.00

5.3. COMBINATION OF FEATURES AND EXPERIMENTS

Different features with and without feature Groups 1-3 were combined to form different feature combinations, and each of these combinations was then used to classify the images by modality using the categories from Table 1.1. The radial clustering technique was used for modality-based classification using 90% of the images from each

category to train the algorithm, and the remaining 10% of the images from each category were used as test images. Classification was performed over 10 randomly generated training/test sets. The details of the different feature combinations are as follows:

1. Combination 1: mostFrequentComponent, avgVal, stdVal and twelve WDD features from the HSV histogram; all these features were computed using the unsmooth HSV histogram.
2. Combination 2: mostFrequentComponent, avgVal, stdVal and twelve WDD features from the HSV histogram; all these features were computed using the smooth HSV histogram.
3. Combination 3: The twelve WDD features computed using the unsmooth HSV histogram.
4. Combination 4: The twelve WDD features computed using the smooth HSV histogram.
5. Combination 5: The twelve WDD features computed using the luminance histograms from Group 2; the texture features from Group 3.
6. Combination 6: mostFrequentGray, avgGray, stdGray from Group 1; the WDD-based features computed using the luminance histograms from Group 2.
7. Combination 7: The WDD-based features computed using the luminance histograms from Group 2.
8. Combination 8: mostFrequentGray, avgGray, stdGray, stdColRed, stdColGreen, and stdColBlue from Group 1; and the WDD-based features computed using the luminance histograms from Group 2; texture features from Group 3.

9. Combination 9: WDD-based features computed using smooth HSV histogram; texture features from Group 3.
10. Combination 10: WDD-based features computed using unsmooth HSV histogram; texture features from Group 3.
11. Combination 11: WDD-based features computed using luminance histograms from Group 2; WDD-based features computed using smooth HSV histogram; texture features from Group 3.
12. Combination 12: WDD-based features computed using luminance histograms from Group 2; WDD-based features computed using unsmooth HSV histogram; texture features from Group 3.

For each feature group ten test cases were evaluated. Tables 5.8-5.10 give the correct percentage test results for different feature combinations. The mean and standard deviation of the ten test cases are also calculated and presented in the tables below.

Table 5.8. Percentage correct test results using radial clustering method classification for combinations 1-4

Iteration	Combination 1	Combination 2	Combination 3	Combination 4
1	98.67	98.67	90.67	92.00
2	96.00	97.33	94.67	93.33
3	97.33	97.33	92.00	90.67
4	100.00	100.00	100.00	97.33
5	100.00	100.00	94.67	94.67
6	97.33	93.33	89.33	86.67
7	98.67	97.33	94.67	97.33
8	97.33	96.00	93.33	96.00
9	98.67	97.33	93.33	92.00
10	96.00	97.33	94.67	94.67

Table 5.8. Percentage correct test results using radial clustering method classification for combinations 1-4 (cont.)

Average	98.00	97.46	93.73	93.46
Standard Deviation	1.44	1.93	2.88	3.29

Table 5.9. Percentage correct test results using radial clustering method classification for combinations 5-8

Iteration	**Combination 5**	**Combination 6**	**Combination 7**	**Combination 8**
1	100.00	100.00	100.00	98.67
2	97.33	96.00	97.33	98.67
3	98.67	96.00	97.33	98.67
4	97.33	97.33	97.33	96.00
5	94.67	94.67	96.00	96.00
6	98.67	97.33	94.67	93.33
7	96.00	100.00	98.66	97.33
8	94.67	97.33	94.67	94.67
9	98.67	94.67	96.00	94.67
10	94.67	94.67	94.67	94.67
Average	96.93	96.80	96.67	96.26
Standard Deviation	1.89	2.00	1.80	1.96

Table 5.10. Percentage correct test results using radial clustering method classification for combinations 9-12

Iteration	**Combination 9**	**Combination 10**	**Combination 11**	**Combination 12**
1	100.00	98.67	100.00	98.67
2	94.67	94.67	96.00	96.00
3	96.00	96.00	96.00	94.67
4	97.33	97.33	98.67	100
5	96.00	97.33	96.00	97.33

Table 5.10. Percentage correct test results using radial clustering method classification for combinations 9-12 (cont.)

6	93.33	93.33	94.67	94.67
7	97.33	97.33	97.33	97.33
8	96.00	96.00	97.33	97.33
9	96.00	97.33	97.33	96.00
10	93.33	96.00	93.33	94.67
Average	96.00	96.40	96.67	96.67
Standard Deviation	1.99	1.55	1.91	1.81

From Tables 5.8-5.10, all of the feature combinations yielded correct classification greater than 93.46%. The maximum accuracy achieved during classification is 98.00% for Combination 1. The features for Combination 1 are based on the HSV histogram compared to other feature combinations which may include features computed using based on the luminance histogram.

6. EXTRACTING CHARACTERS FROM IMAGES

6.1 OVERVIEW

Biomedical images, especially x-ray images, often have characters present on the images. The next task of the research was to correctly recognize the characters using optical character recognition (OCR) process. OCR is a mechanical or electronic process by which images which have characters are converted into files which are machine editable. For example, an image of printed document could be converted to a text using OCR so that the text file can be edited as per requirements. There are a variety of tools performing character recognition. Tesseract –OCR is an open source OCR tool developed by HP Labs and currently maintained by Google [16]. It has been used in this research to perform optical character recognition.

Tesseract-OCR has been written in C++, it was installed on Linux environment. The next sections explain the experiments performed on bio-medical images for character extraction.

6.2 OCR FOR ALL IMAGES

Tesseract-OCR works only with uncompressed TIF images and can be used to recognize a wide range of characters. By default, Tesseract-OCR is trained to recognize English character set along with alpha numeric characters. Tesseract-OCR was used to recognize characters on bio-medical images which belonged to the nine modalities as described in Table 1.1. Images belonging to the “Photograph” modality did not have any characters on them, while images from the “Chart / Graph” and “Radiology” modality

have characters. The images were converted to uncompressed tiff format and given as input to the software. Tesseract-OCR is trained to recognize English characters along with alphanumeric characters; many characters from the images were correctly recognized and saved in a text file. But, the software is not capable of identifying the granular noise present in Radiology and Histology images and so it falsely recognized many alphanumeric characters which were not at all present in the images. This is shown in the Figures 6.1 - 6.12 below:

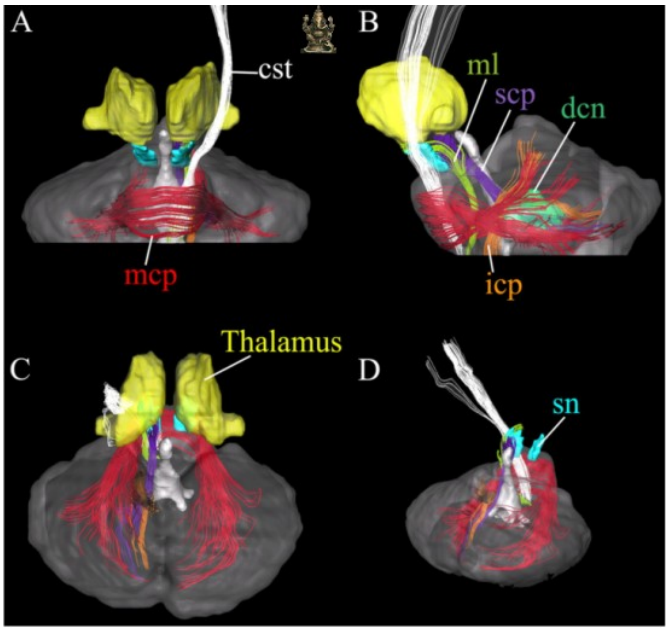


Figure 6.1. Image from modality “Photograph”

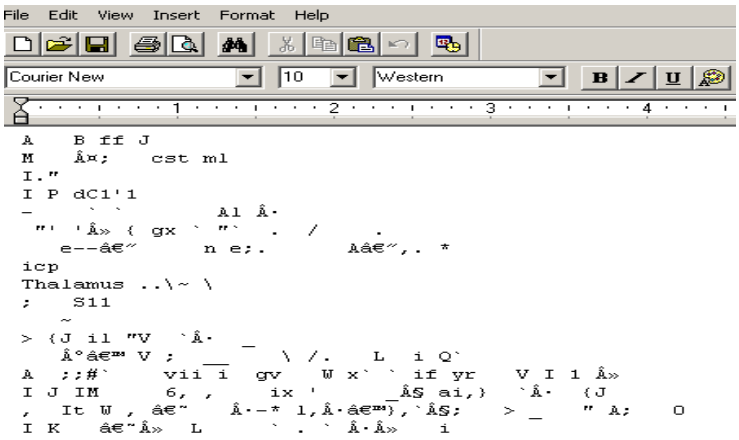


Figure 6.2. Snapshot of characters extracted from the image

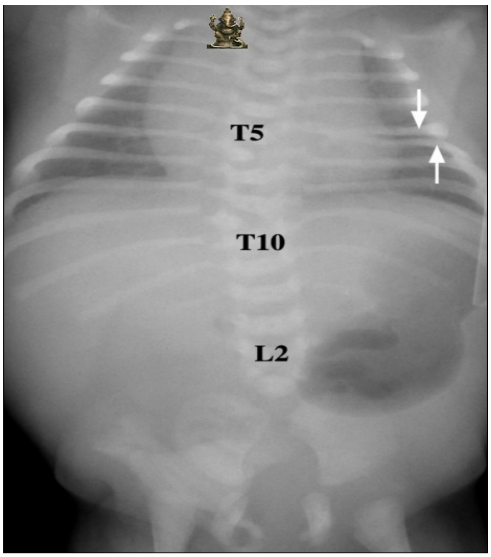


Figure 6.3. Image from modality “Radiology”

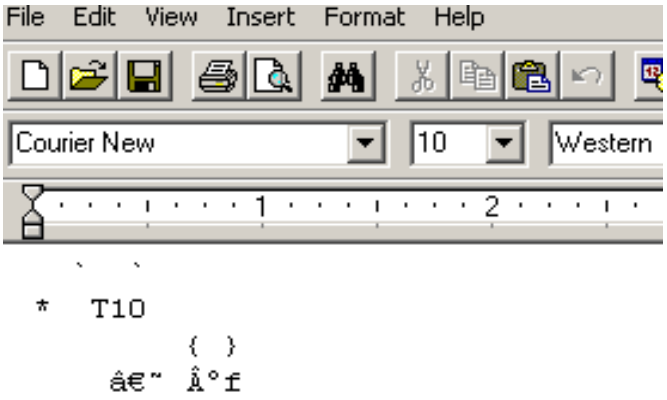


Figure 6.4. Snapshot of characters extracted from the image

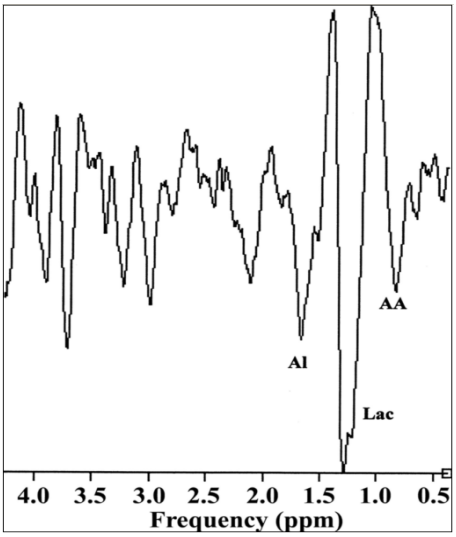


Figure 6.5. Image from modality "Chart/Graph"

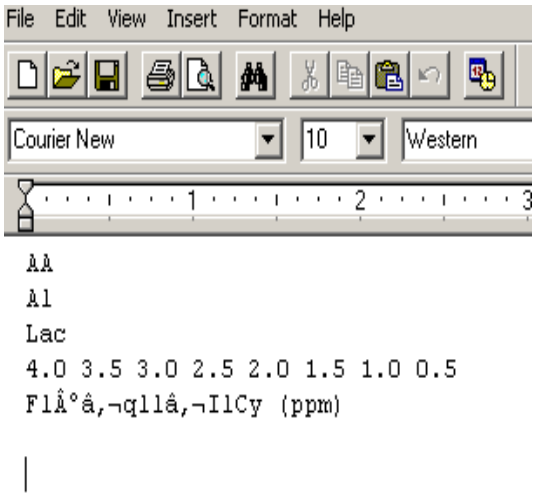


Figure 6.6. Snapshot of characters extracted from the image

PULMONARY VASCULARITY SCORESHEET

I. PULMONARY VASCULARITY: Please indicate grade (+3 thrugh N to -3)

II. CHAMBER ENLARGEMENT (indicate if chamber enlargement is present: grade size from normal (N) to 3+)

Case No.	Oligemia			Normovolemic	Hyperemia			Right Heart Size	Left Heart Size
	+3	+2	+1	0	-1	-2	-3	(0 to 3+)	(0 to 3+)
1.	---	---	---	----	---	---	---	----	----
2.	---	---	---	----	---	---	---	----	----
3.	---	---	---	----	---	---	---	----	----
4.	---	---	---	----	---	---	---	----	----
5.	---	---	---	----	---	---	---	----	----
6.	---	---	---	----	---	---	---	----	----
7.	---	---	---	----	---	---	---	----	----
8.	---	---	---	----	---	---	---	----	----
9.	---	---	---	----	---	---	---	----	----
10.	---	---	---	----	---	---	---	----	----
11.	---	---	---	----	---	---	---	----	----
12.	---	---	---	----	---	---	---	----	----
13.	---	---	---	----	---	---	---	----	----
14.	---	---	---	----	---	---	---	----	----
15.	---	---	---	----	---	---	---	----	----
16.	---	---	---	----	---	---	---	----	----
17.	---	---	---	----	---	---	---	----	----
18.	---	---	---	----	---	---	---	----	----
19.	---	---	---	----	---	---	---	----	----
20.	---	---	---	----	---	---	---	----	----
21.	---	---	---	----	---	---	---	----	----
22.	---	---	---	----	---	---	---	----	----
23.	---	---	---	----	---	---	---	----	----
24.	---	---	---	----	---	---	---	----	----
25.	---	---	---	----	---	---	---	----	----

READER NUMBER: 2

Figure 6.7. Image from modality “Chart/Graph”

File Edit View Insert Format Help

Courier New 10 Western

PULMONARY VASCULARITY SCORESHEET
1. PULMONARY VASCULARITY: Please indicate grade (+3 thrugh N to -3) 11. CHAMBER ENLARGEMENT (indicate if chamber enlargement is present: grade size from normal (N)
tO3+)
Cas; NO. Oligemia Normovulemic Hyperemia Right Heart Size Left Heart Å§izg
+3 +2 +1 O -1 -2 -3 (O1O 3+) (Oto 3+)
1. --- --- --- --Å·- ~-Å· -Å·- ~Å·- â€"-- â€"--
2. â€"-- Å·-- --â€" ---- --- â€"-- --- â€"-- â€"--
3. --- --- --- -â€"-â€" --- â€"-- â€"-Å· â€"-- â€"--
4_ --- --7 Å·~~ --Å»Å» --7 Å»~ Å·~ â€"-- â€"--
5. -Å·â€" -â€"- --- ---- --- --- --- â€"-- â€"--
6. -Å·- -â€"- --- ---- --- --- --- â€"-â€" â€"--
7. -Å·Å· --- â€"-â€" -â€"-- --- â€"-â€" --- â€"-- â€"--
8. --- --- â€"-- -â€"-Å· --- -Å·- â€"-â€" --- Å·--
9. --- --- --Å· ---- --- â€"-- -â€"Å· â€"-- â€"--
10. --- --- Å·-- -â€"â€"- --- --- --Å· â€"-- â€"--
11. --- --- ---- --- --- --- â€"-- â€"--
12. â€"â€"- --- --- ---- --- --- --- â€"-- â€"--
13. â€"â€"- --- â€"-- ---- --- â€"-- â€"-â€" --- â€"--
14. --- --- --- ---- --- --- --- â€"-- â€"--
15. --- --- --- ---- --- --- --- --- ---
16. --- --- --- ---- --- -~~ --- ---
17. --â€" --- --- ---- --- --- --- â€"-- â€"--
18. --- --- --- â€"--- --- Å·-- Å·-â€" --- â€"-â€"
19. â€"-- --- --- --â€"- â€"-- --- --- â€"-- â€"â€"-
20. --- --- --â€" ---- --- --- --- â€"-- â€"--
21. â€"-â€" --- --- ---- --- --- --- â€"-- â€"--
22. --- --- ---Å· ~~ --Å· --- --- â€"~Å·
23. --- -â€"- --- ---- --- Å·-- --- --- â€"--
24. â€"-â€" --- --- ---- --- --- --- â€"-- â€"--
25. -â€"- â€"â€"- -â€"- -â€"-- â€"-- â€"-â€" â€"-- â€"-- â€"--
READER NUMBER: 2

Figure 6.8. Snapshot of characters extracted from the image

Figure 6.9. Image from modality “Photograph”

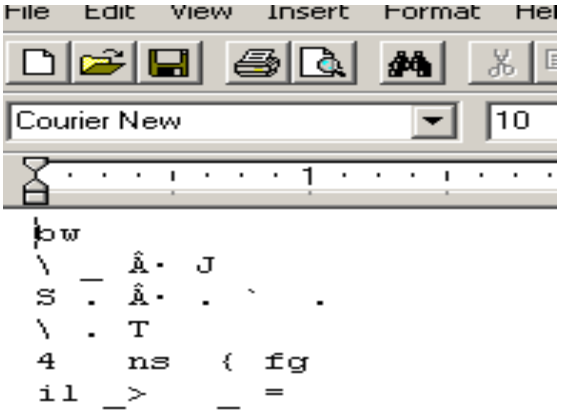


Figure 6.10. Snapshot of characters extracted from the image

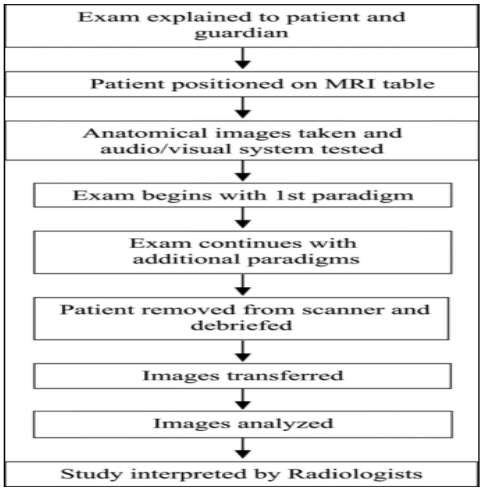


Figure 6.11. Image from modality "Chart/Graph"

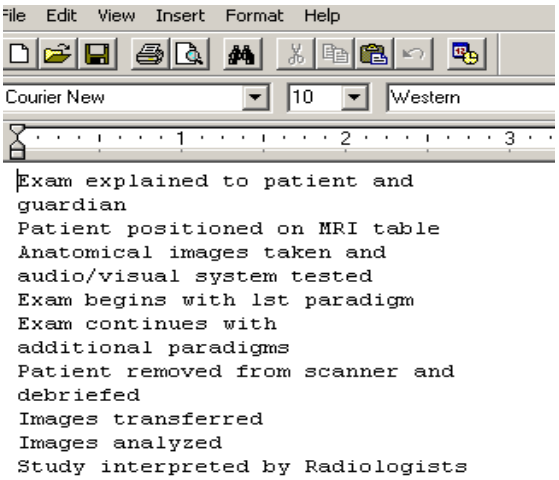


Figure 6.12. Characters extracted from the image

Tesseract-OCR recognizes false characters from photographs which do not actually contain any kind of character on them as shown in Figure 6.9 and 6.10. It also does not recognize characters which are not horizontally oriented. It works the best for images from the 'forms' modality. Images belonging to forms modality have normal English characters, most of which are horizontally oriented. The major reason for this false optical character recognition is that the Tesseract-OCR is trained to recognize English characters of standard size and shape. Accuracy of Tesseract-OCR is known to drop if the size of the characters fall below 10pt x 300 dpi, which is about 20 X 20 pixel size. Optical characters used on the images do not adhere to one size or shape. Also, the orientation and placement of characters is not the same for all images.

6.3 OCR FOR RADIOLOGY IMAGES

Radiology images are mostly grayscale images. These images have a black background and the objects of interest are in the foreground in the lighter shade.

Characters contained in these images are white in color and mostly present at the edges of the images as shown in Figure 6.13:

Figure 6.13. Radiology image

Before using such images as input to Tesseract-OCR, these images were processed in order to convert them into binary images which would have a black background and white characters on them. Figure 6.14 shows the output of the process for the image shown 6.13:

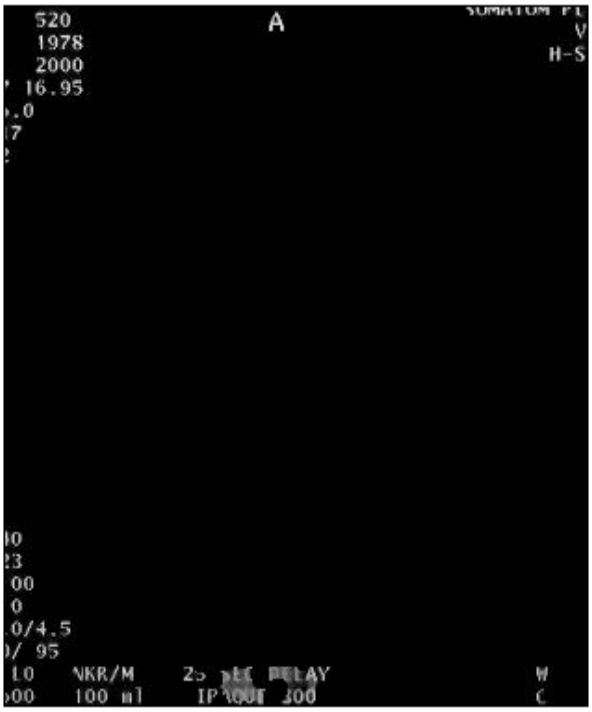


Figure 6.14. Output of the processed radiology image in Figure 6.13

Steps followed to convert the radiology image into binary images with just the characters are described in this section using Figure 6.15 as reference image.

First, define a reference area for the image set (640x480) for calibrating size parameters that are used in the algorithm. Second, convert the input image to luminance. Third, perform wiener filtering of the luminance image (stored in filename_wiener.tif) shown in Figure 6.16:

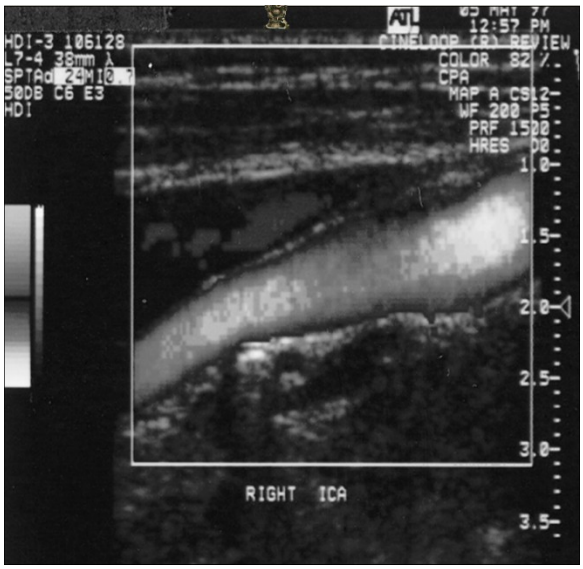


Figure 6.15. Radiology image example for finding text characters

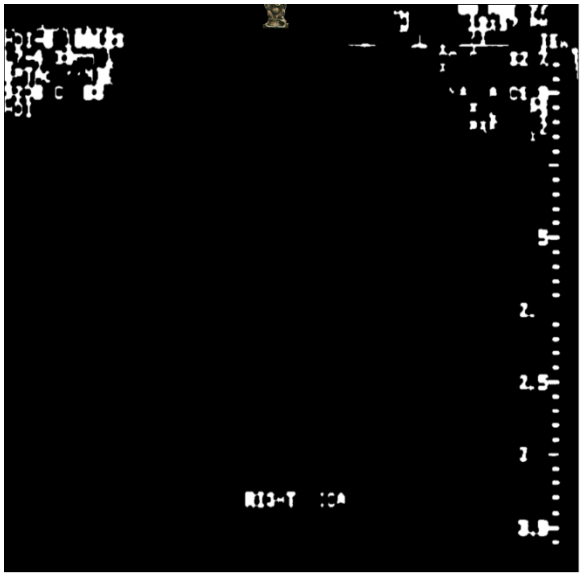


Figure 6.16. Wiener filtered image (step 3)

Fourth, Otsu threshold the wiener filtered image to find textual character-type objects. This mask is intended to find bright textual character-type objects. Accordingly, in order to limit the number of noise objects detected, the Otsu threshold is multiplied by 0.75. The resulting thresholded wiener filtered image is denoted as threshImg. Fifth, Holes in the thresholded images obtained from the previous steps are filled. Sixth, remove large non-text character objects from luminance thresholded images. The resulting image is denoted as updatedThreshImg. Seventh, find the tophat transform image (morphological open the wiener filtered image by a circular structuring element of the radius 10 and subtract this image from the original wiener filtered image) and denote it as grayImg. Eighth, use the Otsu method [11] to threshold the tophat transform image to get a preliminary mask containing textual character-type objects and denote it as maskImg. Ninth, for each object in maskImg, determine if it has any holes (EulerNumber < 1) and see if it intersects with an object in grayImg. If there is an intersection, remove the object. This removes ambiguous textual character-like objects. Tenth, dilate the mask obtained from the previous step and remove the small objects which might have been generated due to previous operations. Eleventh, moderate size objects are added to the mask obtained in the tenth step, objects having Euler number > 1 are considered to be of moderate size, the resulting mask is denoted as maskImg. Twelfth, after performing morphological closing on the mask using structural element of size 5, the resulting mask is denoted as closeImg. Thirteenth, compare the two masks i.e. maskImg and closeImg. Fourteenth, the small objects are compared with the big objects present in the global thresholded image to verify that they belong to the bigger object or not. Fifteenth, check to see the proximity of the accepted objects with the objects of similar size, if they are

close include the latter object with the accepted objects, the resulting mask is denoted as includeOneImg. Sixteenth, remove all small objects from the closed mask. Objects of size < 200 pixels are considered to be small and denoted as includeCornerImg. The resulting mask is denoted as moreClosePruneImg. Seventeenth, if there are no big objects i.e objects of size > 0 in the moreClosePruneImg mask, then the figure does not contain any textual character, else the finalMask is obtained by combining the three masks obtained in the previous steps i.e. includeOneImg, moreClonePruneImg and includeCornerImg using Logical OR operation. Eighteenth, delete the small components from the finalMask. Eighteenth, if no areas are found in the previous step then check across all the bands of the image for objects denoted as finalMask. Nineteenth, delete all components if there are no large blocks of text in finalMask again denoted as finalMask. Twentieth, the outlines from the above mask are removed and denoted as tempFinalMask (stored as filename_final_noOutlines.tif) shown in Figure 6.17.

Twenty-first, in this step, the relatively dark blocks of text are removed; this is done by first calculating the average gray value of the objects present in tempFinalMask denoted by avgGray. Then the avgObjGray value is obtained from the luminance thresholded image, if the avgObjGray < avgGray – 35 then the object is removed (stored as filename_final_noDark.tif) (denoted as finalMask) shown in Figure 6.18.

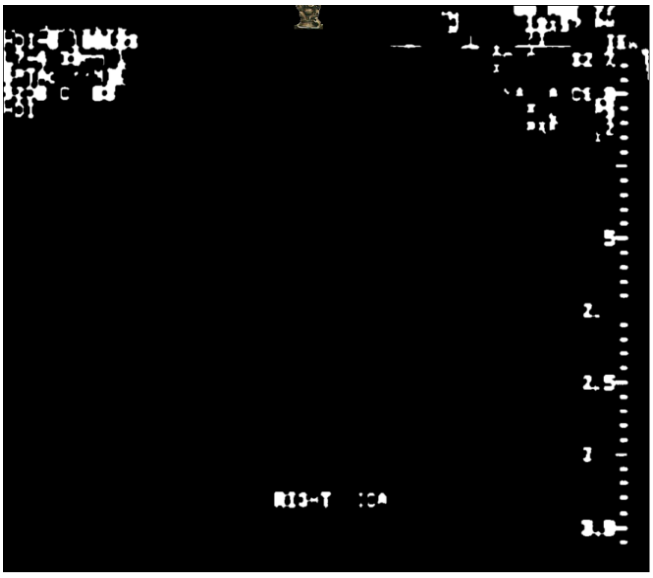


Figure 6.17. Image with no outlines (step 20)

Figure 6.18. Dark text blocks removed (step 21)

Twenty-second, this is the final step of the algorithm, in which only the images with text blocks are created using the finalMask and grayImg (stored as filename_final_combined_gray.tif) (denoted as textOnlyImg) shown in Figure 6.19.

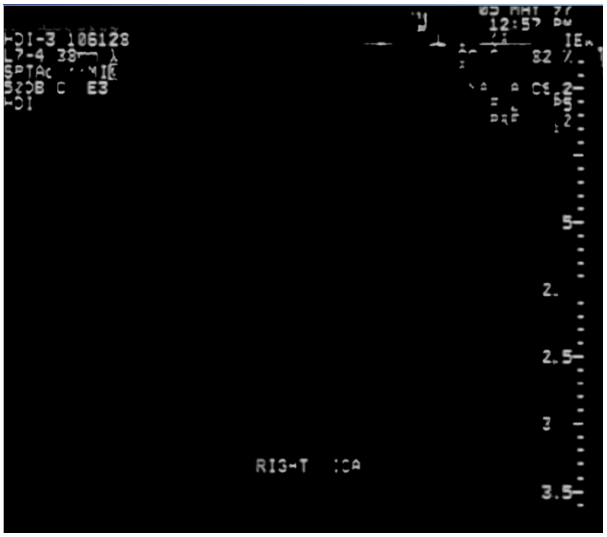


Figure 6.19. Final image with only text (step 22)

These binary images were then converted into uncompressed tif format and given as input to Tesseract-OCR to perform optical character recognition. The results obtained were better than the ones obtained by experiments conducted on raw images i.e. without converting the images into binary image. As the images were binary in nature there was no false recognition of alphanumeric characters. Images which did not have any text on them yielded no results for optical character recognition. For the images which had

characters on them, the characters were identified but were not completely accurate. In most of the cases only some of the characters were identified correctly, rest of them either went unrecognized or were incorrectly recognized as shown below. Figure 6.20 is the binary image which was used as input image for OCR and Figure 6.21 shows the characters extracted from the binary image.

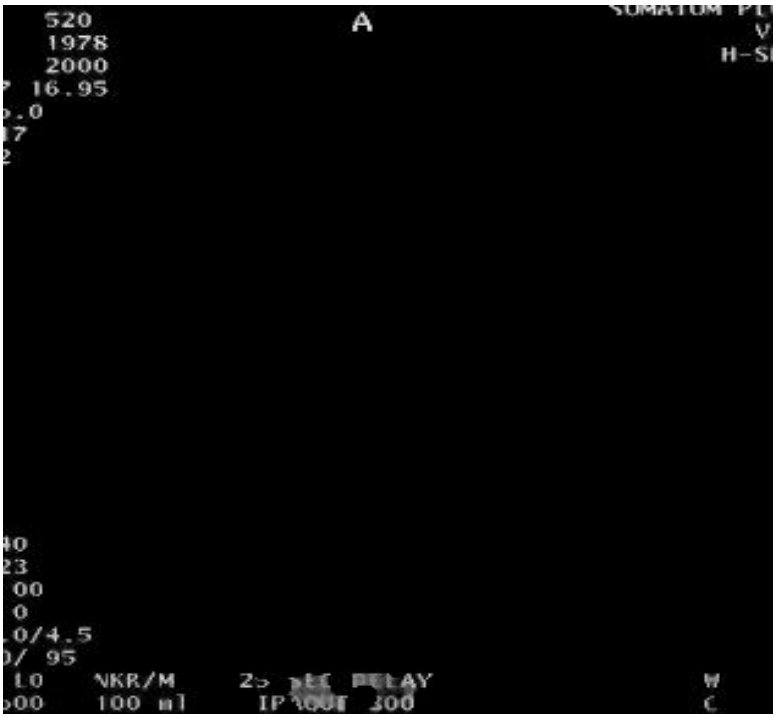
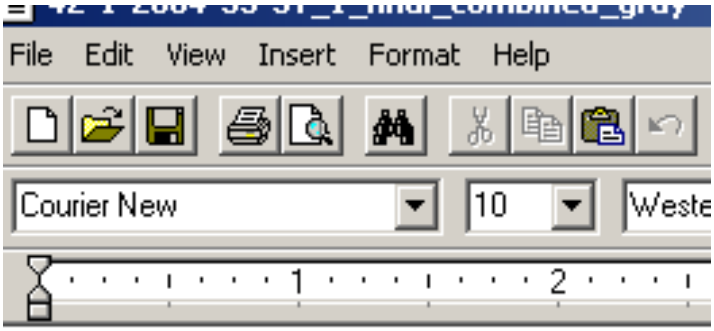


Figure 6.20. Binary image used as input for OCR




```
B
,O.] mw wil. 1.,3 (
```

Figure 6.21. Characters extracted from Figure 6.20

6.4 TESSERACT-OCR ANALYSIS

Tesseract-OCR can be successfully used to perform optical character recognition on bio-medical images. But there is a lot of false character recognition due to Tesseract-OCR not having been trained to recognize English characters of standard size and shape. Optical characters used on the images do not adhere to one size or shape. Also the alignment and placement of characters is not the same for all images.

To overcome the problem of false character recognition the OCR software can be trained. OCR engine will have to be provided with the characters set that is used frequently on the bio-medical images and then run the engine in training mode. This will help the Tesseract-OCR to correctly identify the characters. Tesseract-OCR has been trained in a similar way to recognize characters of many different languages. So it can be concluded that after training the OCR engine with proper character set the results of optical character recognition can be improved.

7. CONCLUSIONS AND FUTURE DIRECTIONS

7.1 CONCLUSIONS

The goal of the research was to successfully classify images from biomedical journals into categories based on their modalities. The different modalities have been presented in Table 1.1. Several global and histogram based features were used to classify the images. Various features were combined to achieve the best rate of classification. The features based on WDD functions gave the best classification rate. The best rate of classification achieved was 98% for combination 1 as described in Section 5.3 with the worst case being 84.91 for group feature 1 as described in Section 5.1. Another goal of the project was to develop libraries which could be used with RapidMiner. Libraries calculating individual features for images and different groups of features have been successfully developed and tested. Also libraries performing classification of images have been successfully implemented and tested.

7.2 FUTURE DIRECTIONS

Soft decision based HSV histogram can be used to develop more features also different variations of the histogram could be used for generating the same features and then classification can be performed to check whether the classification rate improves or deteriorates. Also, the design of the Histogram class could be changed and instead of initializing all the arrays in the constructor they could have been initialized in the respective methods of the Histogram class; thus, making the program more memory efficient. There could just be one result object for all the operators that return only one

feature value. This will reduce the redundant amount of code. RapidMiner itself has a result object. If the results obtained after classification or after calculating features can be mapped to the result object of RapidMiner, then many more operations can be performed on the results. Currently the result object of the custom plug-ins developed as a part of research cannot be used by RapidMiner operators. These result objects are compatible with the custom plug-ins and hence work can be done to map the current result objects with the result object of RapidMiner. For optical character recognition, training the Tesseract-OCR by using the characters used in radiology images can be explored and the accuracy for optical character recognition process can be improved.

BIBLIOGRAPHY

1. Hiremath PS, Pujari J. Content based image retrieval using color texture and shape features,"15th International Conference on Advanced Computing and Communications, pp. 780-784, 2007.

2. Li J, Wang JZ, Widerhold G. IRM: Integrated Region Matching for Image Retrieval. 8th ACM international conference on Multimedia, pp 147-156, October 2000.

3. Demner-Fushman, D., Antani, S., Thoma, G., "Automatically Finding Images for Clinical Decision Support," Seventh IEEE conference on Data Mining Workshops (ICDMW 2007) pp 139-144, 2007.

4. Chen Y, Wang JZ. A region based fuzzy feature matching approach to content-based image retrieval. IEEE Transactions on Pattern and Machine Intelligence, Vol. 24, No. 9, pg 1252-1257 2002.

5. Soffer A. Image categorization using texture features. Fourth International Conference on Document Analysis and Recognition, Vol. 1, pp 233-237, 1997.

6. Niblack W. The QBIC project: Querying images by content using color, texture, and shape. Storage and Retrieval of Image and Video Databases, Vol. 1908, pp173-187, 1993.

7. Vadivel A, Majumdar AK, Sural S. Characteristics of weighted feature vector in content based image retrieval applications. IEEE Conference on Intelligent Sensing and Information Processing, pp.127-132, 2004.

8. Bardet J-M, Lang G, Oppenheim G, Philippe A, Stoev S, Taqqu MS. Semi-parametric estimation of the long-range dependence parameter: a survey. In Theory and Applications of Long-Range Dependency, Doukhan P, Oppenheim G, Taqqu MS, eds. pp 557-577, Birkhauser, 2003.

9. Sural S, Qian G, Pramanik S. Segmentation and histogram generation using the HSV color space for image retrieval. ICIP 2: II-589- II-592, 2002.

10. Piper J, Granum E. On fully automatic feature measurement for banded chromosome classification. Cytometry 1989 May:10 (3):242-255.

11. Haralick RM, Shapiro LG. Computer and Robot Vision, Vol. 1. New York: Addison-Wesley Publishing Co., 1992.

12. Fukunaga K, Hosteler LD. The estimation of the gradient of a density function, with applications in pattern recognition, IEEE Transactions on Information Theory 21(1): 32-40,1975.

13. http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=10161&objectType=file. The algorithm from reference 10 was modified by Bryan Feldman (February 2006) and made available online at the given web site. Last accessed August 13, 2008.

14. Cheng Y. Mean shift, mode seeking, and clustering. IEEE Transactions on Pattern Analysis and Machine Learning 17(5):790-799, 1995.

15. Extending RapidMiner, Chapter 6, rapidminer-4.0-tutorial.pdf, pp 497-521

16. http://code.google.com/p/tesseract-ocr/ - Tesseract-OCR, 09/10/09

VITA

Vikas Nahar was born in Datia, India, on January 01, 1985. He received his Bachelor of Engineering (B.E.) from the Ramrao Adik Institute of Technology (RAIT), Mumbai, India in the field of Information Technology in June 2006.

He joined the Missouri University of Science and Technology (Missouri S&T) in August 2007 and received his Master's degree in May 2010. His areas of interests include image processing, computer networking and computer security.