FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
CENTRO UNIVERSITÁRIO “EURÍPIDES DE MARÍLIA” - UNIVEM
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ANA PAULA DE OLIVEIRA SANTOS

RECUPERAÇÃO DE IMAGENS MAMOGRÁFICAS BASEADA EM CONTEÚDO

MARÍLIA
2006

ANA PAULA DE OLIVEIRA SANTOS

RECUPERAÇÃO DE IMAGENS MAMOGRÁFICAS BASEADA EM CONTEÚDO

Monografia apresentada ao curso de Bacharelado em Ciência da Computação do Centro Universitário Eurípides de Marília UNIVEM, mantido pela Fundação de Ensino Eurípides Soares da Rocha, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientadora:
Profª. Drª. Fátima L. S. Nunes Marques

MARÍLIA
2006

À Deus por ter me dado forças e condições para realizar este trabalho.

Aos meus pais Ilza Alice e José Felix, minha irmã Heloisa e meu namorado Peterson, pelo apoio e compreensão em mais uma etapa de minha vida.

AGRADECIMENTOS

Primeiramente a Deus por ter me dado forças e competência para chegar aonde cheguei e por me sustentar todos os dias da minha vida.

Aos meus pais por todo amor, carinho e compreensão depositados em mim desde que nasci e por sempre estarem ao meu lado, me apoiando, me ajudando e contribuindo para a minha formação pessoal e profissional.

À minha orientadora Fátima pela confiança em meu trabalho, por nunca duvidar da minha capacidade, por sempre me estimular, não me deixando parada ou desanimada, e pela sua ajuda no desenvolvimento deste trabalho.

Ao professor Delamaro pela ajuda e contrbuições neste trabalho e pelas maravilhosas classes do esquema genérico criado.

Ao meu namorado Peterson pelo amor e compreensão nesta fase de minha vida, sempre me tranqüilizando e incentivando a enfrentar novos desafios.

À minha irmã Heloisa que mesmo não estando junto todos os dias sempre me entendeu e torceu muito por mim.

Aos meus amigos da faculdade, Bárbara, Mariana, Danilo, Rodrigo e Adriano e do laboratório LApIS, Ana Cláudia e Larissa pela ajuda, pelo incentivo e pela maravilhosa convivência.

À FAPESP (processo nº. 2006/00463-7) – Fundação de Amparo à Pesquisa do Estado de São Paulo pelo confiança neste trabalho e pelo apoio financeiro.

Muito Obrigada! Sem cada um de vocês eu não estaria aqui!

SANTOS, Ana Paula de Oliveira. Recuperação de Imagens Mamográficas Baseada em Conteúdo. 2006. 154 f. Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

RESUMO

O presente trabalho traz um estudo de conceitos relacionados à recuperação de imagens baseada em conteúdo (CBIR - *Content-Based Image Retrieval*) e a implementação de um sistema para aplicação em um banco de dados de imagens mamográficas, utilizado originalmente para apoiar o desenvolvimento e os testes de sistemas de auxílio ao diagnóstico (CAD – *Computer-Aided Diagnosis*). Os sistemas CBIR permitem o retorno de imagens utilizando outras imagens para a busca. O foco de tais sistemas é pesquisar no banco de dados uma determinada quantidade de imagens similares a uma imagem de consulta, de acordo com um ou mais critérios fornecidos. Os critérios de similaridade são obtidos a partir da extração de características da imagem como cor, textura e forma. Este trabalho foi desenvolvido utilizando a linguagem de programação Java juntamente com a API JAI (*Java Advanced Imaging*).

Palavras-chaves: CBIR. Banco de dados. Imagens médicas. Processamento de imagens. Recuperação de imagens baseada em conteúdo.

SANTOS, Ana Paula de Oliveira. Recuperação de Imagens Mamográficas Baseada em Conteúdo. 2006. 154 f. Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

ABSTRACT

This present work brings a study of the concepts of Content-Based Image Retrieval (CBIR) and the implementation of a system for application in a mammographic image database, used originally to support the development and tests of computer-aided diagnosis (CAD) systems. CBIR systems allow retrieval of images using others images for the search. The focus of such systems is to research in the database a certain amount of similar images, in agreement with one or more supplied criteria. The similarity criteria are obtained starting from the extraction of characteristics of the image as color, texture and shape. This work was developed using the programming language Java together with Java Advanced Imaging (API JAI).

Key-words: CBIR. Database. Medical images. Image processing. Content-based image retrieval.

SANTOS, Ana Paula de Oliveira. Recuperação de Imagens Mamográficas Baseada em Conteúdo. 2006. 154 f. Monografia (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

RESUMEN

Este presente trabajo trae un estudio de los conceptos de Recuperación de la Imagen Basado em Contenido (CBIR – *Content Based Image Retrieval*) y la implementación de un sistema para la aplicación en una Base de Datos de imágenes de pechos, servido originalmente para apoyar el desarrollo y pruebas de sistemas de ayuda a diagnósticos. Los sistemas de ayuda a diagnósticos permiten la recuperación de imágenes que usan otras imágenes para la búsqueda. El enfoque de tales sistemas es investigar en la Base de Datos una cierta cantidad de imágenes similares a la imagen de consulta, de acuerdo con uno o más criterio proporcionado. Los criterios de similariedad son obtenidos a partir del extracto de características de la imagen como color, textura y forma. Este trabajo se desarrollo en el idioma de programación Java juntamente con la API JAI (*Java Advanced Imaging*).

Palabras-clave: CBIR. Base de datos. Imágenes médicas. Processamiento de imagen. Recuperación de imágenes basados en contenidos.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API: *Application Program Interface*

BI-RADS: *Breast Imaging Reporting and Data System*

CAD: *Computer-Aided Diagnosis*

CBIR: *Content Based Image Retrieval* (Recuperação de Imagens Baseada em Conteúdo)

DER: Diagrama Entidade Relacionamento

DICOM: *Digital Imaging and COmmunication in Medicine*

EESC/USP: Escola de Engenharia de São Carlos / Universidade de São Paulo

EPM/UNIFESP: Escola Paulista de Medicina / Universidade Federal de São Paulo

HCFMRP-USP: Hospital das Clínicas da Faculdade de Medicina de Ribeirão Preto - Universidade de São Paulo

INCA: Instituto Nacional de Câncer

IRMA: *Image Retrieval in Medical Applications*

JAI: *Java Advanced Imaging*

LAPIMO: Laboratório de Análise e Processamento de Imagens Médicas e Odontológicas

LApIS: Laboratório de Aplicações de Informática em Saúde

MAMs: *Metric Access Methods*

MAO: Mapa Auto-Organizável

MIAS: *Mammographic Images Analysis Society*

PACS: *Picture Archiving and Communication System*

PxR: Precisão *versus* Revocação (*precision-recall*)

ROI: *Region of Interest*

SOM: *Self-Organizing Maps* (Mapa Auto-Organizável)

SRIM: Sistema de Recuperação de Imagens Mamográficas

SVM: *Support Vector Machines*

ToRM: Tomografia por Ressonância Magnética

UNESP: Universidade Estadual Paulista

SUMÁRIO

INTRODUÇÃO

Uma estimativa da incidência do câncer para 2006 do Instituto Nacional de Câncer (INCA) prevê que o câncer de mama será o de maior número entre as mulheres das regiões Nordeste, Centro-Oeste, Sudeste e Sul, com incidência de 27,16‰, 37,99‰, 70,49‰, 69,04‰, respectivamente (INCA, 2005).

Esquemas de diagnóstico auxiliado por computador têm sido desenvolvidos por vários grupos de pesquisas, visando a auxiliar na detecção precoce do câncer de mama, pois é sabido que a descoberta da doença na fase inicial favorece a sua cura (DENGLER *et al.*, 1993).

A maioria dos trabalhos nessa área é desenvolvida tendo as imagens geradas através de mamografia por raios X como fonte de dados. Devido a deficiências inerentes ao processo de obtenção da imagem com esta técnica, nem sempre o especialista consegue detectar sinais precoces da doença apenas através da inspeção visual sobre o mamograma. Giger (2000) afirma que de 10% a 30% de mulheres que tiveram câncer de mama e foram submetidas à mamografia tiveram mamogramas negativos, isto é, o radiologista interpretou o exame como normal. Nesse sentido, os esquemas CAD podem ser úteis, pois, através da aplicação de técnicas de processamento de imagens, tentam emitir uma segunda opinião ao radiologista, chamando a atenção para áreas suspeitas da imagem. Entre essas áreas, estão, por exemplo, aquelas que podem conter agrupamentos de microcalcificações (*clusters*) ou nódulos em formação. Esquemas completos fornecem a identificação, localização e classificação dessas estruturas, podendo, dessa forma, contribuir para a descoberta mais precoce da doença, inclusive em pacientes assintomáticos.

Uma das maiores dificuldades encontradas durante o desenvolvimento das técnicas acima mencionadas diz respeito à avaliação dos processos. Não é fácil saber se uma determinada técnica é eficiente ou não, pois os resultados podem variar de acordo com o conjunto de imagens utilizado nos testes. Para se atestar a viabilidade do uso de uma técnica, são necessários testes com um vasto conjunto de imagens que tenham, preferencialmente, características de aquisição variadas e que atendam aos requisitos da finalidade da técnica, isto é, conter as estruturas procuradas na detecção. Isso envolve uma pesquisa intensa junto a hospitais e clínicas para obtenção dos filmes radiográficos e dos respectivos laudos médicos, uma exaustiva tarefa de digitalizar essas imagens – muitas vezes mais de uma vez para garantir diferentes características exigidas nos testes – e, juntamente a tudo isso, uma catalogação sistemática e eficiente que permita uma recuperação rápida e precisa das imagens de acordo com as suas características.

Recuperação de imagens baseada em conteúdo (CBIR – *Content-Based Image Retrieval*), consiste em uma busca feita por imagens semelhantes a um determinado padrão fornecido. O que torna isso possível é a comparação feita entre aspectos da imagem dada e as imagens armazenadas na base de dados. O interessante é a realização da busca por semelhança a um caso específico (imagem específica) e não como no processo convencional de busca textual que compara parâmetros do usuário com valores de atributos armazenados (GATO *et al.*, 2004).

Os sistemas CBIR permitem o retorno de imagens utilizando outras imagens para a busca. O foco de tais sistemas é pesquisar no banco de dados uma determinada quantidade de imagens similares a uma imagem de consulta, de acordo com um ou mais critérios fornecidos. Os critérios de similaridade são obtidos a partir da extração de características da imagem como cor, textura e forma.

Segundo Gudivada *et al.* (1995), CBIR pode ser utilizada em diversas e numerosas áreas, entre elas:

- galerias de artes e administração de museu;
- projetos de engenharia e arquitetura;
- *design* de interiores;
- percepção e administração remota dos recursos da terra;
- sistemas de informações geográficas;
- administração de banco de dados científicos;
- previsão de tempo;
- vendas no varejo;
- projetos de tecido e moda;
- administração de banco de dados de marca registrada e direito de cópia;
- execução da lei e investigação criminal;
- sistemas de arquivamento e comunicação de imagem (*Picture Archiving and Communication System* - PACS).

Objetivos e justificativas

Em trabalhos anteriores (NUNES *et al.*, 2004a; NUNES *et al.*, 2004b) foi implementada uma ferramenta para gerenciar uma base de imagens mamográficas usando a Internet. O objetivo era disponibilizar um grande conjunto de imagens mamográficas digitalizadas e suas respectivas informações, a fim de que seu uso pudesse contribuir na avaliação de esquemas CAD (*Computer-Aided Diagnosis*) em mamografia. O sistema foi implementado com tecnologia gratuita, permitindo acesso rápido, eficiente e sem custos

adicionais ao usuário. A base de imagens disponibilizada pode ser usada também como ferramenta didática para o ensino de tópicos relacionados à mamografia.

Para que essa base de imagens possa ser efetivamente utilizada, estratégias eficientes de buscas devem ser implementadas. Atualmente a busca é apenas textual, sendo que o usuário fornece parâmetros como idade do paciente, tipo de anomalia procurada, entre outros. A recuperação baseada em conteúdo vem sendo utilizada como uma solução para buscas em bases de imagens, principalmente quando se deseja obter imagens semelhantes a uma determinada imagem fornecida para comparação.

Com base nas informações descritas, este projeto apresenta a implementação de um sistema de recuperação de imagens mamográficas baseada em conteúdo, a fim de auxiliar os testes de sistemas CAD e também como ferramenta didática para o ensino de áreas da saúde que utilizam as imagens médicas como material didático. Pretende-se, desta forma, gerar material para teste para dar apoio às pesquisas do LApIS - Laboratório de Aplicações de Informática em Saúde.

Organização da monografia

Esta monografia está organizada da seguinte forma:

Capítulo 1 – Recuperação de Imagens Baseada em Conteúdo: apresenta o conceito de CBIR bem como a forma de extração e indexação de suas características e alguns trabalhos encontrados na literatura.

Capítulo 2 – Metodologia: descreve as tecnologias utilizadas e as características das imagens usadas. Também apresenta algoritmos implementados para executar a segmentação

das imagens, assim como um esquema genérico para CBIR, as características implementadas e a interface do sistema.

Capítulo 3 – Resultados e Discussões: relata os testes criados para a recuperação das imagens do sistema proposto assim como a discussão do desempenho apresentado pelo sistema durante a execução dos testes.

Conclusões: apresenta as conclusões finais, assim como os trabalhos futuros.

CAPÍTULO 1 – RECUPERAÇÃO DE IMAGENS BASEADA EM CONTEÚDO

Este capítulo apresenta conceitos básicos de CBIR, mostrando como podem ser feitas a extração e indexação das características das imagens, bem como a apresentação dos trabalhos existentes na literatura.

1.1 Conceitos Básicos

CBIR é um processo que pode exigir muito tempo de processamento, por isso a comparação entre as imagens é feita utilizando um conjunto de características extraídas das imagens, que as descrevem. Estas características podem tanto ser obtidas por especialistas como ser extraídas das imagens, utilizando-se algoritmos automáticos alcançando melhores resultados, também permitindo o processamento em grande quantidade de dados (ARAUJO *et al.*, 2002).

De acordo com Kinoshita *et al.* (2004), sistemas CBIR utilizam para a indexação e recuperação das imagens atributos como cor, textura e forma. A principal busca é pelas imagens mais relevantes segundo características de similaridade entre uma imagem de consulta e as que estão armazenadas no banco de dados.

Sistemas CBIR possibilitam a recuperação de um conjunto finito de imagens similares a uma imagem exemplo; utilizando informações inerentes à própria imagem, similaridade essa com um nível de semelhança pré-determinado pelo usuário. As comparações entre as imagens podem ser realizadas através de características extraídas

automaticamente, estas agrupadas em um vetor de características que tem por finalidade armazenar a *essência* da imagem (MARQUES *et al.*, 2006).

1.2 Extração das Características

Para o processo de recuperação das imagens baseada em conteúdo é necessário um ou mais algoritmos que obtenham de forma automática as características das imagens, chamados de extratores, que resultam em um conjunto numérico representando as características extraídas. Pode-se utilizar um ou mais extratores para representar uma imagem (ARAUJO *et al.*, 2002). Esta extração é um dos pontos mais sensíveis da recuperação de imagens por conteúdo, os extratores fornecem subsídios para obter respostas às consultas por similaridade (MARQUES *et al.*, 2006). Gato *et al.* (2004) lembraram que esta é uma etapa muito importante porque sintetiza as propriedades inerentes, que serão utilizadas para a recuperação das imagens.

1.3 Indexação das Características

Outra etapa importante de sistemas CBIR é a indexação das características, pois possibilita o armazenamento e a recuperação das características em um banco de dados (GATO *et al.*, 2004). Por possibilitar a utilização de mais de um algoritmo extrator, algumas das propostas encontradas na literatura utilizam um vetor de espaço *n*-dimensional, chamado de vetor de características, que é uma maneira de armazenar todas as características obtidas por tais algoritmos e, para a recuperação, é necessário calcular a menor distância entre os

vetores (ANDRÉ *et al.*, 2004; ARAÚJO *et al.*, 2002). A recuperação baseada em similaridade pode ser realizada por meio de uma função que calcule a similaridade dos vetores e, conseqüentemente, das características nele armazenadas (ARAUJO *et al.*, 2002).

Araújo *et al.* (2002) lembram que a função de distância é um algoritmo que compara os vetores das imagens sob consulta, devendo satisfazer algumas propriedades em um domínio métrico, retornando um valor não negativo. Quanto menor esse valor, mais parecidas são as imagens comparadas.

Bueno (2001) apresenta algumas funções de distância que podem ser utilizadas para a comparação entre dois vetores. Apesar das várias funções apresentadas, a mais utilizada é a Distância Euclidiana. Nas equações de 1 a 10, *x* e *y* são os vetores dos quais será calculada a distância e *m* é o número de variáveis de entrada (atributos) da aplicação.

$$\textbf{Minkowsky:}\quad D(x,y)=\left(\sum_{i}^{m}\left|x_i-y_i\right|^{r}\right)^{\frac{1}{r}} \tag{1}$$

$$\textbf{Euclidean:}\quad D(x,y)=\sqrt{\sum_{i=1}^{m}(x_i-y_i)^2} \tag{2}$$

$$\textbf{Manhattan/ }\textit{\textbf{city-block}}\textbf{:}\quad D(x,y)=\sum_{i=1}^{m}\left|x_i-y_i\right| \tag{3}$$

$$\textbf{Camberra:}\quad D(x,y)=\sum_{i=1}^{m}\frac{\left|x_i-y_i\right|}{\left|x_i+y_i\right|} \tag{4}$$

$$\textbf{Chebychev:}\quad D(x,y)=\max_{i=1}^{m}\left|x_i-y_i\right| \tag{5}$$

$$\textbf{Quadratic:}\quad D(x,y)=(x-y)^T Q(x-y)=\sum_{j=1}^{m}\left(\sum_{i=1}^{m}(x_i-y_i)q_{qi}\right)\left(x_j-y_j\right)$$

Q é uma matriz de tamanho *m* x *m* finita, positiva e específica ao problema. (6)

Mahalanobis: $D(x,y) = [\det V]^{\frac{1}{m}} (x-y)^T V^{-1} (x-y)$

V é matriz de covariância de $A_1 \ldots A_m$, e A_j é o vetor de valores para o atributo j que aparece nas instâncias do conjunto de treinamento $1 \ldots n$.

(7)

Correlação: $$D(x,y) = \frac{\sum_{i=1}^{m} (x_i - \bar{x}_i)(y_i - \bar{y}_i)}{\sqrt{\sum_{i=1}^{m} (x_i - \bar{x}_i)^2 \sum_{i=1}^{m} (y_i - \bar{y}_i)^2}}$$

$\bar{x}_i = \bar{y}_i$ é o valor médio para o atributo i que aparece no conjunto de treinamento.

(8)

Chi-quadrado: $$D(x,y) = \sum_{i=1}^{m} \frac{1}{soma_i} \left(\frac{x_i}{tam_x} - \frac{y_i}{tam_y} \right)^2$$

$soma_i$ é a soma de todos os valores para o atributo i que aparece no conjunto de treinamento, e tam_x é a soma de todos os valores no vetor x.

(9)

Correlação de Posição de Kendall:

$$D(x,y) = 1 - \frac{2}{m(n-1)} \sum_{i=1}^{m} \sum_{j=1}^{i-1} sinal(x_i - x_j)\, sinal(y_i - y_j)$$

$sinal(x)$=-1, 0 ou 1 se x<0, x=0, ou x>0, respectivamente.

(10)

Um ponto fraco da distância Euclidiana, apresentado por Bueno (2001), é o fato de que se um dos atributos do vetor tiver uma faixa de valores possíveis muito maior do que os outros atributos, então este atributo pode sobrepujar os demais. Para tentar corrigir este problema, geralmente as distâncias são normalizadas, dividindo-se a distância para cada atributo pela faixa (ou seja, máximo-mínimo) daquele atributo, tal que, para cada atributo, a distância esteja na faixa entre 0 e 1. Para evitar os elementos de exceção, pode-se dividir a distância pelo desvio padrão em vez de usar o intervalo de valores ou, ainda, mapear qualquer valor fora desta faixa para o valor máximo ou mínimo. Ainda relacionados à idéia de normalização, há sistemas que utilizam pesos de atributos para o cálculo da distância.

Os atributos podem ser lineares (podendo ser contínuos ou discretos) ou nominais. Os atributos contínuos são aqueles que representam valores reais, como a massa ou velocidade de um objeto. Os atributos discretos podem assumir apenas valores pertencentes a

um determinado conjunto. E, por fim, os atributos nominais são aqueles que não necessariamente obedecem a uma ordem linear, são simbólicos. As funções apresentadas nas equações de 1 a 10, não tratam apropriadamente de atributos que não sejam contínuos (BUENO, 2001).

Uma vez definida a função de distância, pode-se fazer as consultas por similaridade. As funções mais utilizadas são: *k*-vizinhos mais próximos (*k-nearest neighbor*) e abrangência (*range*). A consulta pelos *k* vizinhos mais próximos retorna os *k* valores mais próximos do ponto de referência e a consulta por abrangência retorna os valores pertencentes ao raio de abrangência (raio de busca), dado que representa a distância de similaridade entre um ponto de referência e a imagem em questão (BUENO, 2001; ARAUJO *et al.*, 2002; KINOSHITA, *et al.*, 2004).

1.4 Trabalhos correlatos

Na literatura são encontradas várias propostas de sistemas CBIR com o objetivo de auxiliar o diagnóstico médico. Alguns destes sistemas encontram-se descritos nesta seção.

1.4.1 *Framework* baseado em aprendizado

El-Naqa *et al.* (2002) exploraram o uso de um *framework* baseado em aprendizado hierárquico para recuperar imagens mamográficas relevantes de um banco de imagens com o propósito de auxiliar o diagnóstico. A preocupação fundamental foi de como caracterizar a noção de similaridade entre as imagens para usá-las no acesso de imagens relevantes do banco

de imagens. Foram investigados o uso de alguns algoritmos, derivados de redes neurais e Máquinas de Vetores de Suporte (SVM – *Support Vector Machines*) em uma rede de aprendizado hierárquica em dois estágios. Em El-Naqa *et al.* (2004) os autores afirmam que, nesta proposta, cada similaridade é aprendida em treinamentos por meio de exemplos vindos de observadores humanos.

Dadas uma imagem de referência e uma base de dados contendo imagens mamográficas, o sistema atua em dois estágios: no primeiro, denominado classificação, a recuperação é agilizada por uma rápida desqualificação de várias imagens mamográficas para futuras considerações; no segundo estágio, denominado regressão, ajusta-se a máquina de aprendizado para predizer as imagens que apresentam alguma similaridade com a imagem de referência. Aquelas imagens que sobreviveram ao primeiro estágio são comparadas com a imagem de referência e, então, é extraído um número representando o coeficiente de similaridade, no qual aquelas imagens com maior coeficiente de similaridade são recuperadas pelo sistema.

Segundo El-Naqa *et al.* (2004), a meta desta proposta é recuperar imagens mamográficas de uma base de dados contendo agrupamentos de microcalcificações similares ao da imagem de consulta. Também foi explorado como incorporar interação humana *on-line* para realizar um *feedback* relevante no *framework* de aprendizado.

Este *framework* foi testado usando um banco de imagens mamográficas do Departamento de Radiologia da Universidade de Chicago (*Department of Radiology at the University of Chicag*o) que continha 76 mamografias diferentes digitalizadas com resolução de 0,1 mm/*pixel* e usando 10 *bit*s de níveis de cinza. Todas as imagens continham agrupamentos de microcalcificações identificados por um radiologista especialista. El-Naqa *et al.* (2002) afirmaram que os resultados iniciais demonstraram que a proposta de uma rede de aprendizado em dois estágios supera a rede de aprendizado de um único estágio.

O desempenho do sistema foi avaliado usando curvas de *precision-recall* (precisão e revocação) calculadas usando um procedimento de validação cruzada. Os autores afirmam que os resultados iniciais demonstram que: (1) o *framework* de aprendizado pode predizer exatamente a similaridade perceptiva reportada por observadores humanos, servindo de base para CBIR; (2) o *framework* baseado em aprendizado pode superar a simples similaridade baseada na distância métrica; (3) o uso de uma rede hierárquica em dois estágios pode melhorar a performance da recuperação; (4) o *feedback* relevante pode ser eficazmente incorporado no *framework* de aprendizado para realizar progresso na recuperação baseada na precisão com interação *on-line* com usuários.

1.4.2 Recuperação através do histograma métrico

Traina *et al.* (2002) apresentaram uma técnica para a captura das características de brilho das imagens, gerando uma recuperação de imagens baseada em conteúdo mais rápida, chamada de histograma métrico. Esta técnica reduz a dimensionalidade do vetor de características da imagem, conduzindo a um processo de indexação e recuperação mais rápido e flexível. Também foi apresentada uma nova função de distância métrica *DM()* para medir a distância entre as imagens por meio de seus histogramas métricos.

Para permitir a recuperação das imagens é necessário indexar as informações da mesma. Os autores desenvolveram os MAMs (Métodos de Acesso Métrico – *Metric Access Methods*) construídos a partir das características da imagem. Como algoritmos para extração de forma e texturas são muito caros computacionalmente e dependem do domínio da aplicação, foram deixados como o último passo na separação das imagens.

Um PACS com recursos para recuperação de imagens baseada em conteúdo, chamado de cbPACS, foi apresentado por Bueno *et al.* (2002).

Primeiramente, foi eliminado o fundo das imagens para poder extrair as características da imagem em si. Em seguida foi extraído o histograma normalizado da imagem e gerado o histograma métrico, criando dois conjuntos de características para cada conjunto de dados. A indexação dos dados de cada conjunto de características foi feita pela estrutura *Slim-tree*, proposta por Traina Jr. *et al.* (2000), que é uma estrutura balanceada e dinâmica. Os histogramas normalizados foram indexados utilizando a função de distância *Manhattan* e para permitir a indexação dos histogramas métricos e, conseqüentemente, suportar a recuperação de imagens baseada em conteúdo, foi necessário o desenvolvimento de uma nova função de distância métrica *DM()*, também apresentada em Traina *et al.* (2002), que atende a três requisitos: simetria, não negatividade e desigualdade triangular.

Os autores avaliaram este sistema utilizando dois conjuntos de imagens, um com 500 imagens de crânio e outro com 4247 imagens de várias partes do corpo, ambos obtidos de tomografia por ressonância magnética (ToRM). Cada imagem possuía 256 níveis de cinza e diferentes resoluções espaciais. A avaliação foi feita comparando o conjunto de imagens resultantes das consultas utilizando histogramas métricos e normalizados, considerando que os resultados dos histogramas normalizados foram os corretos. Como resultado desta comparação, observaram que o tempo para a construção da *Slim-tree*, a fim de indexar os histogramas métricos era muito mais rápido (58% mais rápido para o conjunto de dados menor e 690% mais rápido para o conjunto maior) do que o tempo para indexar os histogramas normalizados. Também foi medido o número de cálculos de distância realizados por segundo, e encontraram 1.289.400 distâncias *DM()* por segundo e 269.430 distâncias de *Manhattan* por segundo, indicando que o cálculo da distância *DM* é muito mais rápido, na ordem de 4 a 10 vezes, do que o da distância de *Manhattan*. Concluindo, os pesquisadores

afirmaram que os histogramas métricos superam os histogramas tradicionais por permitirem a comparação de imagens de tamanhos diferentes e que foram obtidas em níveis de quantização distintos.

Segundo os autores, em Traina *et al.* (2002), o histograma métrico também apresenta efeitos colaterais desejados, como a recuperação de imagens de um modo que a escala, translação e rotação dos objetos seja invariante, assim como o brilho das imagens, possibilitando a recuperação de imagens com brilho, escala, translação ou rotação diferentes da imagem de referência, sem esforço computacional adicional.

1.4.3 Sistema CBIR de características de textura

Outro sistema CBIR que extrai um vetor de atributos de textura de uma imagem e busca, em um banco de imagens, aquelas com características semelhantes à imagem apresentada foi descrito por Marques *et al.* (2002). Como o objetivo do sistema é auxiliar no diagnóstico de lesões da mama, o mesmo apresenta as imagens recuperadas a fim de que o usuário possa verificar os diagnósticos associados a elas.

Este sistema, implementado para trabalhar na plataforma Windows, utilizou a linguagem de programação Delphi para construir a interface com o usuário e o Sistema Gerenciador de Banco de Dados Oracle 8i, que armazenou as imagens com suas informações não textuais. O banco de imagens continha 100 casos de lesões de mama contendo microcalcificações mamárias, sendo 50 lesões malignas e 50 benignas. As mamografias foram digitalizadas com uma resolução espacial de 600 pontos por polegada e 256 níveis de cinza. Cada mamograma foi analisado por médicos especialistas do Hospital das Clínicas da Faculdade de Medicina de Ribeirão Preto (HCFMRP), que identificaram a região de interesse

(ROI – *region of interest*) de cada imagem. As imagens foram agrupadas segundo valores de atributos de textura (medida de correlação, momento da diferença inversa, entropia, entropia da diferença e entropia da soma), calculados para cada ROI.

O sistema foi avaliado da seguinte forma: para cada consulta realizada, um médico radiologista comparou a imagem de referência e as imagens recuperadas de forma visual e atribuiu uma nota entre 1 e 5, com o seguinte significado: 1 (completamente diferente), 2 (parcialmente diferente), 3 (semelhante), 4 (parcialmente idêntico) e 5 (idêntico). Os autores afirmaram que os resultados apresentados foram satisfatoriamente precisos, podendo o sistema ser usado como uma ferramenta de apoio, por possuir um potencial de aplicação para auxiliar um médico radiologista, especialmente por permitir consultas baseadas na descrição gráfica do conteúdo da imagem, podendo otimizar o processo de consulta ao banco.

1.4.4 IRMA (*Image Retrieval in Medical Applications*)

Considerando PACS e CBIR, Lehmann *et al.* (2003) apresentaram os primeiros resultados do sistema IRMA e discutiram a integração deste sistema dentro de um ambiente PACS. Este sistema consistia basicamente de um banco de dados relacional, escalonador e um servidor *web*, o qual foi instalado em um servidor de imagens DICOM (*Digital Imaging and COmmunication in Medicine* – padrão de comunicação e armazenamento de imagens médicas e informações associadas a elas).

O foco principal da integração de CBIR e PACS é a transparência de: (a) local e acesso dos dados, métodos e experimentos; (b) replicação para métodos no desenvolvimento; (c) trabalho de processamento e extração das características realizados de forma simultânea;

(d) sistema por tempo de implementação do método e (e) distribuição do trabalho quando executando uma consulta.

De acordo com Lehmann *et al.* (2004) este sistema não recupera imagens de apenas uma modalidade particular ou um contexto de diagnóstico, mas visa a obter uma estrutura geral para análise do conteúdo semântico que é apropriado para numerosas aplicações. No IRMA, o processamento das imagens é feito passo-a-passo e resulta em seis camadas de modelagem de informação distintas (camada de dados cru, camada de dados registrados, camada de características, camada de esquema, camada de objeto e camada de conhecimento) que incorporam um conhecimento médico especialista. Este processamento multicamadas foi implementado utilizando um sistema distribuído com apenas três elementos de núcleo.

Nas camadas semânticas de modelagem de informação, o processo de recuperação foi dividido em sete passos consecutivos de processamento (categorização, registro, extração das características, seleção das características, indexação, identificação e recuperação).

O sistema contava com 12.479 imagens, sendo que 6.397 já haviam sido rotuladas de acordo com o código IRMA. Os autores, em Lehmann *et al.* (2003), chegaram à conclusão de que os primeiros experimentos conduzem o IRMA a resultados encorajadores, e provam a transparência do sistema, concluindo que o sistema atua de forma autônoma, mas transparente e também que a combinação de PACS e CBIR permite uma melhor seleção da imagem em rotina clínica suportando ambas as informações, logísticas e de suporte à decisão. Em Lehmann *et al.* (2004), os autores concluíram que em contraste com aplicações específicas, o IRMA apresenta uma abordagem genérica de recuperação de imagens baseada em conteúdo para imagens médicas. Afirmam ainda que o sistema suporta uma rápida prototipação e rápida integração de métodos modernos de análise de imagens. Conseqüentemente, estreita a distância entre o cunho semântico de uma imagem e qualquer descritor alfanumérico que é sempre incompleto.

1.4.5 Recuperação usando rede neural do tipo MAO e correlação cruzada

André *et al.* (2004) apresentaram um sistema CBIR que utiliza rede neural do tipo Mapa Auto-Organizável (MAO) de Kohonen e a técnica de correlação cruzada, usadas respectivamente para reduzir o tamanho das imagens e recuperar imagens mamográficas similares à imagem apresentada para a consulta.

A base de dados utilizada foi a de MIAS, apresentada em Suckling *et al*. (1994), que contém imagens com resolução de 200 *microns* de tamanho de *pixel*. Os mamogramas têm 1024x1024 *pixel*s e 8 *bit*s de escala de quantização de cinza. Esta base tem 322 imagens de 161 pacientes. Foram levadas em consideração quatro características: densidade (1 a 4), tamanho da mama (pequeno, médio ou grande), lado (mama esquerda ou direita) e forma da mama (arredondada ou periforme), todas elas com valores atribuídos por um radiologista experiente. O primeiro passo foi a extração do vetor de características de cada imagem utilizando a rede neural, onde foi inicialmente determinada a estrutura do MAO, estrutura esta que foi treinada para garantir a convergência dos pesos dos neurônios da rede. Após esse treinamento foi usado o MAO para construir um mosaico de cada imagem. A técnica de correlação cruzada foi usada sobre esses mosaicos para comparar a imagem de pesquisa com as 322 imagens da base de dados.

O sistema foi testado selecionando aleatoriamente 20 imagens da base de dados citada acima. Para cada imagem selecionada o sistema mostra as imagens recuperadas em ordem crescente de correlação cruzada, podendo também recuperar imagens cujos valores de correlação cruzada foram os menores. Para cada consulta foram calculadas a precisão e revocação, medidas utilizadas para se avaliar o desempenho de um sistema de recuperação de informação, aplicando-se a sistemas de recuperação de imagens. Estas medidas foram

calculadas considerando todas as quatro características e também cada uma separadamente (no capítulo 4 desta monografia são explicadas as medidas de precisão e revocação). Segundo os autores, quando foram observados os resultados de todas as características, o sistema não apresentou um bom desempenho. Quando foi analisada cada característica separadamente, o sistema obteve resultados relativamente bons quanto à forma, tamanho e lado das mamas, demonstrando o seu potencial de aplicação.

1.4.6 SRIM – Sistema de Recuperação de Imagens Mamográficas

Foi apresentado por Gato *et al.* (2004) um outro sistema CBIR denominado SRIM, que faz parte de um esquema CAD do Laboratório de Análise e Processamento de Imagens Médicas e Odontológicas (LAPIMO) do Departamento de Engenharia Elétrica da EESC/USP. Tem por objetivo recuperar imagens de um banco de dados, considerando similaridades de uma dada imagem, servindo como base de apoio para agilizar avaliações de imagens mamográficas com características específicas.

O SRIM conta com um banco de imagens com cerca de 3300 imagens com características variáveis em termos de idade da paciente, densidade e achados mamográficos (microcalcificações, calcificações, nódulos e densidades assimétricas), além de mamogramas normais. Todas as imagens foram digitalizadas com uma resolução espacial de 75 µm ou 150 µm e com resolução de contraste de 12 *bit*s (4096 níveis de cinza). As características extraídas foram densidade geral (analisando-se a mama toda) e área da mama. Para realizar as consultas por similaridade, foi implementada a busca aos vizinhos mais próximos, onde é apresentada uma imagem de referência ao sistema e o mesmo recupera as *N* imagens mais semelhantes.

1.4.7 Investigação de atributos visuais de forma

Kinoshita *et al.* (2004) propuseram um sistema CBIR de imagens mamográficas, no qual o objetivo foi investigar atributos visuais de forma combinada com atributos de Medida de Granulometria e atributos espectrais no Domínio de Radon. O sistema utilizou rede de Mapa Auto-Organizável (MAO) de Kohonen como forma de recuperação de imagem.

Para o teste do sistema, foram utilizadas 1080 imagens pertencentes ao Banco de Imagens do HCFMRP-USP distribuídas em 270 mamografias bilaterais da vista crânio-caudal (CC) e 270 mamografias bilaterais da vista médio-lateral-oblíqua (MLO). Além das descrições, os laudos assinalam a discriminação da mama pela composição dos tecidos (1, 2, 3 e 4) e categoria (0, 1, 2, 3 e 4) de acordo com o critério BI-RADS (*Breast Imaging Reporting and Data System*). BI-RADS é um sistema de padronização dos laudos mamográficos elaborado pelo Colégio Americano de Radiologia (*American College of Radiology* – ACR), cujo objetivo é fazer com que os médicos radiologistas utilizem a mesma linguagem e classificação nos laudos de exames mamográficos (DAY-CARE, 2006).

Primeiramente as imagens foram preparadas, sendo executada uma suavização de ruídos através da equação de Difusão Anisotrópica combinada com o filtro de Wiener. Nesta preparação, a região da mama foi segmentada pelas técnicas de limiarização de Princípio de Máxima Entropia, Método de Preservação, Método de Ridder e Método da Matriz de Co-ocorrência, sendo que cada uma destas técnicas forneceu um limiar para a segmentação, sendo escolhido o limiar "ótimo" de maneira visual supervisionada por um radiologista especialista. Usando-se a transformada de Radon, foi detectada a região do músculo peitoral e da posição do mamilo. Logo após essa preparação das imagens foram extraídos os atributos de forma (área e razão de diâmetro), de solidez, de Medida de Granulometria e no Domínio de Radon.

Tendo extraído os atributos das mamografias, foi feita seleção dos atributos visuais, para definir a combinação de atributos que iriam compor o conjunto de vetores de entrada para o treinamento e teste do sistema de recuperação das imagens. Para recuperar as imagens foi empregada uma rede MAO de Kohonen (SOM) e a comparação entre as imagens foi feita por uma técnica de alinhamento das imagens e uma medida de similaridade (coeficiente de correlação).

O sistema foi avaliado observando visualmente as imagens recuperadas e as classificações, tanto BI-RADS quanto da composição dos tecidos, verificando as imagens recuperadas em relação à imagem de consulta. Como havia uma variabilidade da classificação, os autores adotaram que estavam corretas as imagens recuperadas com composições de tecidos na classificação vizinhas à imagem teste, ou seja, se a imagem teste teve uma composição avaliada pelo número 1, foram consideradas corretas as imagens recuperadas com composição 1 ou 2; caso a imagem teste tenha sido 3, foram consideradas corretas aquelas imagens recuperadas com composição 2, 3 ou 4. A taxa de precisão obtida foi de 81,81% e o coeficiente de correlação médio foi de 0,838, indicando a relevância dos atributos investigados.

1.4.8 Nova função de distância

Moshfeghi *et al.* (2004) apresentaram uma nova função de distância para CBIR para medir a similaridade entre duas imagens, função esta que é uma variante da entropia relativa, ou distância de Kullback-Liebler. Esta nova distância é a soma da entropia relativa de duas imagens, uma para com a outra. É uma função de simetria não-negativa que só é zero quando

duas imagens têm distribuições de probabilidade idênticas. Este método foi implementado em um sistema e foi aplicado para uma base de dados de imagens médicas.

A nova distância usa uma variante de entropia relativa para medir a similaridade entre duas imagens. A entropia da variável aleatória é obtida pela média dos *bits* necessários para representá-la. A entropia relativa ou distância de Kullback-Liebler representa o número adicional de *bits* necessários na média, quando comparada ao limite de entropia, para representar a variável aleatória. A entropia relativa é usada na informação teórica da medida de distância entre duas distribuições possíveis da variável aleatória. Nesta proposta, os histogramas da busca de imagens e de consulta da imagem são usados para apresentar a probabilidade de distribuição.

Este sistema foi testado com uma base de dados contendo 120 imagens médicas digitais, obtidas por diferentes modalidades como: raio-X, ultra-sonografia, tomografia computadorizada e imagens de ressonância magnética. Para uma imagem de consulta, o sistema calcula a distância de similaridade para a base de dados de imagens e ordena o resultado do melhor para o pior. É extraído das imagens, tanto da de consulta como as que estão sendo pesquisadas, o seu respectivo histograma. No histograma, cada barra é conhecida por *bin* e representa a quantidade de *pixels* da imagem com aquele determinado valor. Como as imagens são dimensionadas, as mesmas possuem o mesmo número de *pixels*, tornando esta proposta viável. Segundo os autores, os experimentos realizados mostram que os *bins* de tamanhos 8, 16 ou 32 são boas soluções conciliatórias entre suavizar o ruído do histograma e manter as características do histograma, e também que este método é computacionalmente simples, pois não requer imagens segmentadas.

1.4.9 Recuperação de imagens com regiões anatomicamente normais

Corboy *et al.* (2005) propuseram um sistema CBIR para recuperar imagens de regiões anatomicamente normais, presentes em estudos de Tomografia Computadorizada do pulmão e abdômen. Foram implementadas e comparadas oito medidas de similaridade usando descritores de *pixels* locais e globais de co-ocorrência de texturas: (1) Distância Euclidiana, (2) Distância de Minkowski, (3) Chi-quadrado, (4) Weighted-MeanVariance, (5) Jeffrey-divergence, (6) Cramer-von Mises, (7) Distância Kolmogorov-Smirnov, e (8) Distância Hausdorff.

Para cada órgão segmentado foi calculado um conjunto de dez características de textura de Haralick nos níveis global e individual de cada *pixel*. Conseqüentemente, cada órgão ou *pixel* (dependendo do nível considerado) foi representado como um vetor com 10 elementos utilizados para comparação da similaridade entre as imagens/órgãos. O passo do processamento também é aplicado antes do cálculo de similaridade: os descritores de textura são normalizados de tal forma que suas diferenças de escala não influenciam no resultado da similaridade. Depois da normalização, são calculadas as oito medidas de similaridade entre a imagem de consulta e todas as outras imagens da base de dados.

Os resultados foram obtidos usando uma base de imagens contendo 344 tomografias computadorizadas, obtidas do Hospital *Northwestern Memorial*, com tamanho de 512 x 512 *pixels* e com resolução de 12 *bits* de escala de cinza, imagens estas segmentadas de cinco órgãos: coração e grandes vasos, parênquima renal e *splenic* e espinha dorsal.

Os autores avaliaram os resultados com respeito à precisão métrica da recuperação (precisão calculada dividindo o número de imagens recuperadas que são relevantes pelo número total de imagens recuperadas, sendo consideradas relevantes àquelas imagens

recuperadas que pertencem à mesma região anatômica da imagem de consulta) procurando os cinco melhores resultados recuperados e avaliando as oito medidas com respeito a estes resultados com melhor similaridade. Foi procurada a melhor medida de similaridade para o nível *pixel* e a melhor medida para o nível global, concluindo que em sistemas CBIR usando diferentes medidas de similaridade para cada tipo de órgão, melhoraria significativamente a precisão do sistema.

1.4.10 Levantamento de características

A partir dos trabalhos encontrados na literatura foram levantadas as características mais utilizadas para a recuperação de aplicações que envolvem imagens médicas. A Tabela 1 apresenta as principais características, tanto levantadas da literatura como de obras clássicas de processamento de imagens, e suas respectivas descrições.

Tabela 1 – Características que podem ser extraídas por sistemas CBIR e suas referências

Característica	Descrição	Referência
Desvio Padrão	- Mede a dispersão dos valores dos *pixels* em relação ao valor médio. - Média do contraste, calculado sobre uma área com *k*-vizinhos.	GONZALEZ (2000)
Entropia	- A entropia de uma variável aleatória é uma medida do número médio de *bits* necessários para representá-la. - Define a quantidade média de informações obtida pela observação de uma única saída da fonte.	GONZALEZ (2000)
Entropia Relativa ou distância de Kullback-Lieber	- Representa a número adicional de *bits* necessários na média, em comparação ao limite de entropia, para representar uma variável aleatória.	MOSHFEGHI (2004)
Centro de Massa	- Representa o somatório do posicionamento (do *pixel*) multiplicado pela massa (neste caso, o valor do *pixel*, que é representado pelo nível de cinza do mesmo).	HIRATA (2006)
Histograma	- Fornece a freqüência de cada nível de cinza na imagem. - É representado por um vetor com n elementos, onde n representa a quantidade de níveis de cinza, chamado de *bins*. - Fornece uma estimativa da probabilidade de	BUENO (2002) TRAINA JR (2000) GONZALEZ (2000)

Característica	Descrição	Referência
	ocorrência do nível de cinza.	
Histograma Normalizado	- Permite a comparação de imagens de qualquer tamanho. - Invariante quanto às transformações geométricas (escala, rotação e brilho). - Fornece a freqüência em porcentagem de cada nível de cinza. - Também é indexado em um vetor com n elementos, onde n é número de níveis de cinza da imagem.	BUENO (2002) TRAINA JR (2000)
Histograma Métrico	- Obtido a partir do histograma normalizado. - O *bucket* é equivalente ao *bin* dos histogramas convencionais. - Cada *bucket* corresponde a um segmento de linha aproximado e representa um subconjunto de *bins* do histograma original. - Os *buckets* não precisam ser regularmente espaçados.	BUENO (2002) TRAINA JR (2000)
Medida de Área	- Conta o número de *pixels* de um objeto em uma imagem. - Conta a freqüência de níveis de cinza do objeto. - Considera a quantidade de *pixels* da área segmentada, sem o fundo.	KINOSHITA (2004) GATO (2004)
Medida de Razão de Diâmetro	- Fornece o grau de elongação da mama, quanto maior o valor dele, mais alongada será a forma da mama. - Diâmetro *Dy* (distância da parede torácica e o mamilo). - Diâmetro *Dx* (distância entre os dois extremos da mama). - *Dx* tem direção perpendicular à direção de *Dy* e passa pelo ponto médio desse diâmetro.	KINOSHITA (2004)
Medida de Solidez (ou grau de compactação ou compacidade)	- Razão do perímetro e a área do objeto. - Para mamas semicirculares, o valor da solidez será zero e esse valor é aumentado de acordo com a complexidade e rugosidade da mama.	KINOSHITA (2004)
Média dos valores dos *pixels*	- É o valor médio dos níveis de cinza referentes aos *pixels* pertencentes à mama.	GATO (2004)
Atributos de Medida de Granulometria	- Determinação da distribuição dos tamanhos de objetos em uma amostra.	KINOSHITA (2004)
Atributos no Domínio de Radon	- Soma de intensidade ao longo de uma direção da imagem. - Essa soma se transforma em um ponto no domínio de Radon formado pela distância perpendicular e pelo ângulo entre a distância perpendicular e o eixo *x*.	KINOSHITA (2004)
Densidade	- Após a varredura da área pertencente à mama, é calculada a média de níveis de cinza referentes aos *pixels* que pertencem à mama.	GATO (2004)
Atributo de Textura	- Indica a freqüência de ocorrência de um particular par de níveis de cinza. - Também chamado de matriz de co-ocorrência.	MARQUES (2002)
Correlação Cruzada	- Um mosaico representa um quadro de tamanho 3x3 *pixels*, pertencente à imagem.	ANDRÉ (2004)

Com base nas informações apresentadas nesse capítulo, é proposta a construção de sistema de recuperação de imagens mamográficas baseada em conteúdo, que auxilie no treinamento médico e nas pesquisas do LApIS.

CAPÍTULO 2 – METODOLOGIA

Neste capítulo são apresentadas as características do desenvolvimento do sistema, tais como: tecnologias utilizadas, aspectos da segmentação das imagens, um esquema genérico proposto para CBIR, a extração de algumas características e a elaboração do sistema final.

2.1 Tecnologias usadas

Este sistema foi desenvolvido utilizando a linguagem de programação Java (SUN, 2006b), em virtude de algumas vantagens que esta apresenta como flexibilidade, portabilidade, reúso de código, facilidade de programação, entre outras. Juntamente com esta linguagem de programação foi utilizada a API (*Application Program Interface*) JAI (*Java Advanced Imaging*) (SUN, 2006a) que, segundo Santos (2004), possibilita a representação, o processamento e a visualização de imagens. Como Sistema Gerenciador de Banco de Dados foi utilizado o Derby que é um banco de dados relacional gratuito e inteiramente desenvolvido em Java (APACHE, 2006).

2.2 Características das imagens

As imagens utilizadas neste projeto fazem parte de um banco de imagens desenvolvido pelo LAPIMO (Laboratório de Processamento de Imagens Médicas e

Odontológicas, da EESC/USP). A composição da base de imagens procurou obter a maior quantidade possível de mamogramas, de forma a incluir imagens provenientes de diferentes hospitais com os quais o LAPIMO mantém convênio de colaboração: Hospital das Clínicas da Faculdade de Medicina de Ribeirão Preto (HCFMRP-USP), Santa Casa de Misericórdia de São Carlos, Hospital São Paulo (EPM/UNIFESP), Hospital Pérola Byington (São Paulo), Clínica Segalla (Bauru-SP), Hospital Amaral Carvalho (Jaú-SP) e Hospital das Clínicas da Faculdade de Medicina da UNESP (Botucatu-SP). Na digitalização das imagens foram utilizados dois digitalizadores a *laser*, ambos da marca Lumisys (Lumiscan 50 e Lumiscan 75) e que, segundo o fabricante, possibilitam obter imagens com até 12 *bits* de resolução de contraste (4096 níveis de cinza).

2.3 Segmentação das imagens

A primeira parte do projeto foi a implementação de um algoritmo para a eliminação do fundo da imagem, conforme ilustrado na Figura 1, uma vez que as características que serão obtidas das imagens devem ser computadas considerando apenas a área da mama, e não a imagem inteira. O programa que implementa este algoritmo pode ser observado no Anexo A desta monografia.

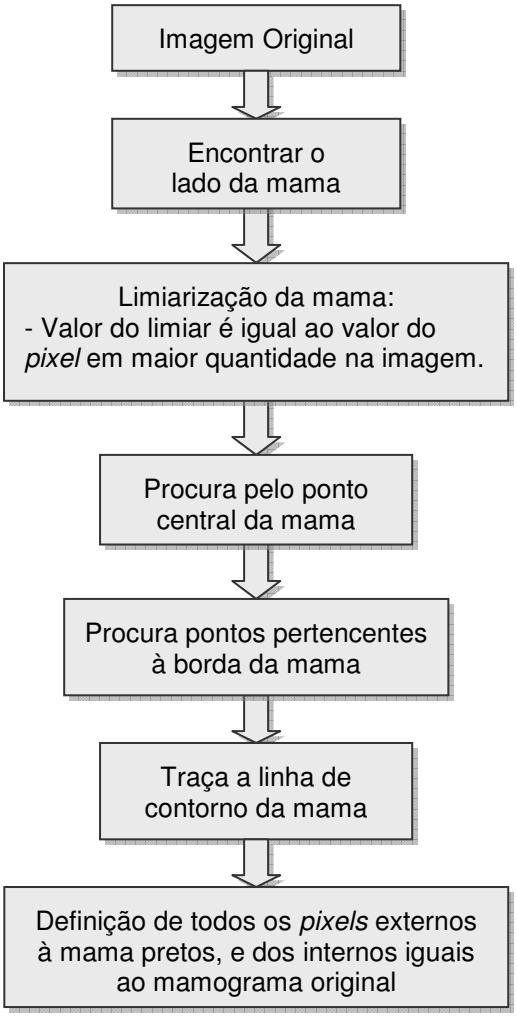


Figura 1 – Esquema geral para o algoritmo de eliminação de fundo

Primeiramente, o algoritmo para a segmentação das imagens detecta o lado onde a mama está, calculando-se a média de cinza das 10ª e 11ª colunas mais próximas do início e do fim da imagem. Foram escolhidas as 10ª e 11ª colunas mais próximas das laterais da imagem para que não houvesse erro caso as imagens não tivessem sido digitalizadas perfeitamente retilíneas. Levando-se em consideração que o fundo da imagem tem os valores de níveis de cinza mais homogêneos do que a mama, é calculada a diferença entre as médias das primeiras e das últimas colunas, considerando-se que a maior diferença pertence ao lado da mama.

Logo após a determinação do lado da mama, é feita a limiarização da imagem, usando o nível de cinza em maior quantidade (encontrado pelo valor máximo do histograma) como valor de limiar. Como resultado desta limiarização é obtido uma imagem com a área da

mama preenchida com *pixels* brancos e a parte do fundo em preto, como exemplificado na Figura 2. Como as imagens utilizadas foram gravadas com uma resolução de contraste de 16 *bits*, e o monitor representa imagens em 8 *bits*, a título de ilustração, as imagens da interface do sistema são mostradas em 8 *bits*, a fim de permitir melhor visualização das estruturas presentes na imagem mamográfica.

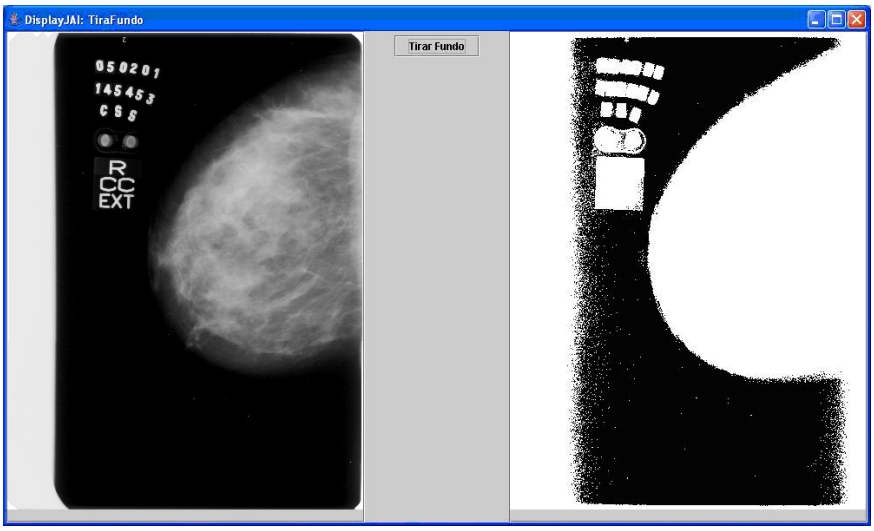


Figura 2 – Resultado da limiarização de uma imagem

A partir da imagem limiarizada, é encontrada a linha central da área que representa a mama. Para isso, percorre-se o lado da mama a partir da coluna mais externa (primeira ou última) à procura de um *pixel* preto. Caso não seja encontrada, o algoritmo desloca-se para a próxima coluna até encontrar o primeiro *pixel* preto. Quando um *pixel* presto é encontrado, verifica-se se este está a uma altura inferior a um terço da altura da imagem. Caso esteja, é considerado pertencente à borda de cima da mama, devendo ser armazenado. Caso o *pixel* encontrado esteja a uma altura superior a um terço da imagem, o mesmo é considerado como pertencente à borda inferior, devendo ser calculada a distância entre este último *pixel* e o primeiro, e encontrado o valor médio que corresponderá à linha central da mama. A linha central de uma imagem mamográfica está sendo representada pela intersecção das linhas cinzas na Figura 3.

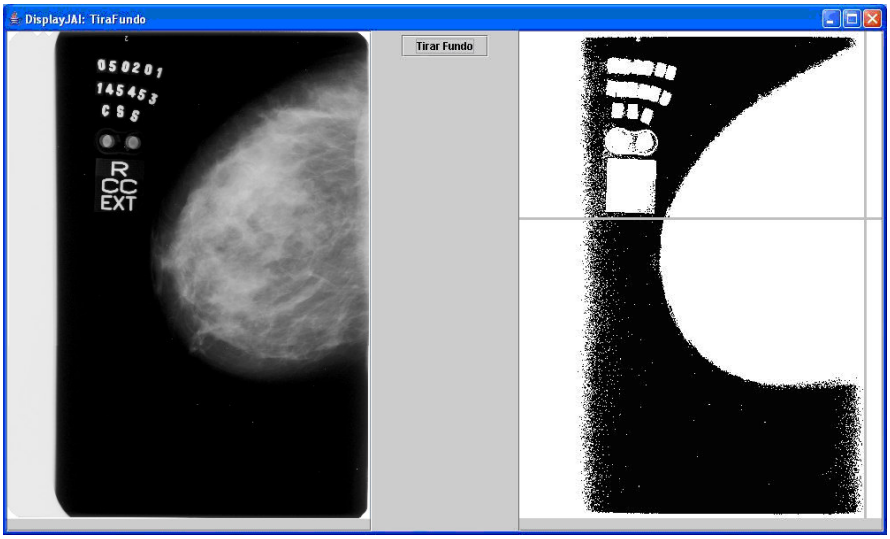


Figura 3 – Exemplo do centro de uma imagem mamográfica

De posse do ponto central, composto pela linha central e pela última coluna encontrada anteriormente, são lançados raios de 0,5 em 0,5 graus, procurando-se ao longo da linha traçada, os *pixel*s pretos, uma vez que a mama está branca e assim que um *pixel* preto é encontrado, este pertence à borda da mama. Os pontos são armazenados em dois vetores: um guardando a posição da coordenada *x*, e outro da coordenada *y*. Para traçar o contorno da mama, é necessário ligar os pontos encontrados pelas linhas calculadas a partir da equação reduzida da reta (11). Considerando-se dois pontos, $P=(x_p,y_p)$ e $Q=(x_q,y_q)$, pertencentes à reta, tem-se que $m = \frac{(y_q - y_p)}{(x_q - x_p)}$ e $b = y_p - m \cdot y_p$.

$$\boxed{y = mx + b} \tag{11}$$

Por exemplo, a partir dos pontos P(2,5) e Q(7,15), aplicados sobre a Equação (11), tem-se: $m = \frac{(15-5)}{(7-2)} = 2$ e $b = 5 - 2 \cdot 2 = 1$. Portanto, a equação reduzida da reta para este exemplo é dada por $y = 2x + 1$.

A Figura 4 ilustra o resultado da aplicação do algoritmo para a eliminação do fundo de uma imagem mamográfica.

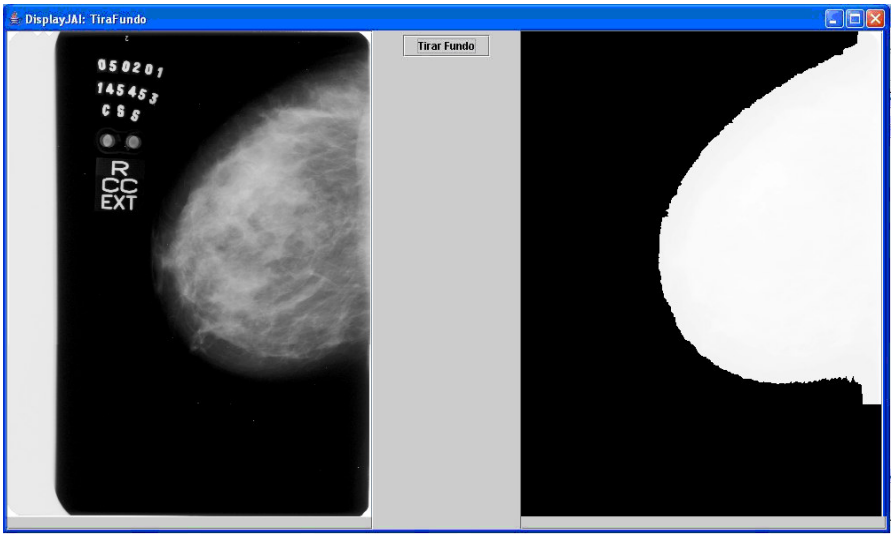


Figura 4 – Exemplo da aplicação do algoritmo para eliminação do fundo em uma imagem mamográfica

2.4 Modelo de dados

Para a implementação do sistema foram criadas três tabelas: IMAGEM, que armazena um código e o caminho até a imagem partindo-se do diretório onde é executado o programa; CARACTERISTICA, que armazena um código e o nome referente à característica e IMAGEM_CARACTER, que faz o relacionamento entre as duas anteriores, armazenando o código da imagem e o código da característica e também o valor associado àquela característica da referida imagem. Portanto, a tabela IMAGEM armazenada todas as imagens pertencentes ao banco, a tabela CARACTERISTICA todas as características e a tabela IMAGEM_CARACT guarda o valor de cada característica em cada uma das imagens do banco, valor este calculado quando é feita a primeira busca utilizando a referida característica. O Diagrama Entidade-Relacionamento (DER) das tabelas apresentadas se encontra na Figura 5.

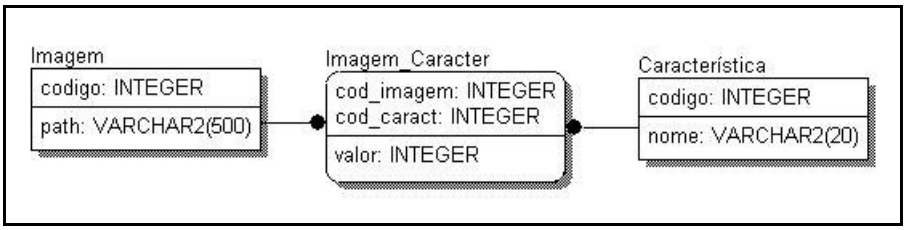


Figura 5 – Diagrama Entidade Relacionamento das tabelas do sistema

2.5 Esquema genérico para CBIR

Para a implementação do sistema, foi desenvolvida uma estrutura genérica para CBIR, de forma que futuramente as classes implementadas possam ser aplicadas para outros tipos de imagens, uma vez que permite a inserção de novos extratores de características no sistema. Este esquema pode ser observado no Anexo A desta monografia.

Para este esquema genérico apresentado na Figura 6, foram criadas quatro classes que servirão de base para o sistema:

- `ParameterBlock`, cujos objetos da classe funcionam como um vetor de elementos que servirão de parâmetro para os extratores de características;
- `AbstractExtractor`, classe abstrata que serve de base para a criação dos extratores de características. Padroniza a implementação de um construtor, um método para fazer a comparação de duas imagens (`compare`), outro para calcular o valor da característica (`computeValue`) e também permite ao programador definir os parâmetros para o extrator (`setParameters`);
- `SimilarityFunction`, classe responsável por permitir a combinação de diversos extratores para a obtenção do grau de similaridade entre duas imagens;
- `CharVector`, classe que cria o vetor de características, podendo estabelecer o nome e o valor das características (`setValue`), recuperar o valor de uma determinada

característica (`getValue`), calcular a similaridade entre dois vetores de características (`computeSimilarity` e `computeStrictySimilarity`) ou, ainda, retornar o vetor de características atual (`getVector`).

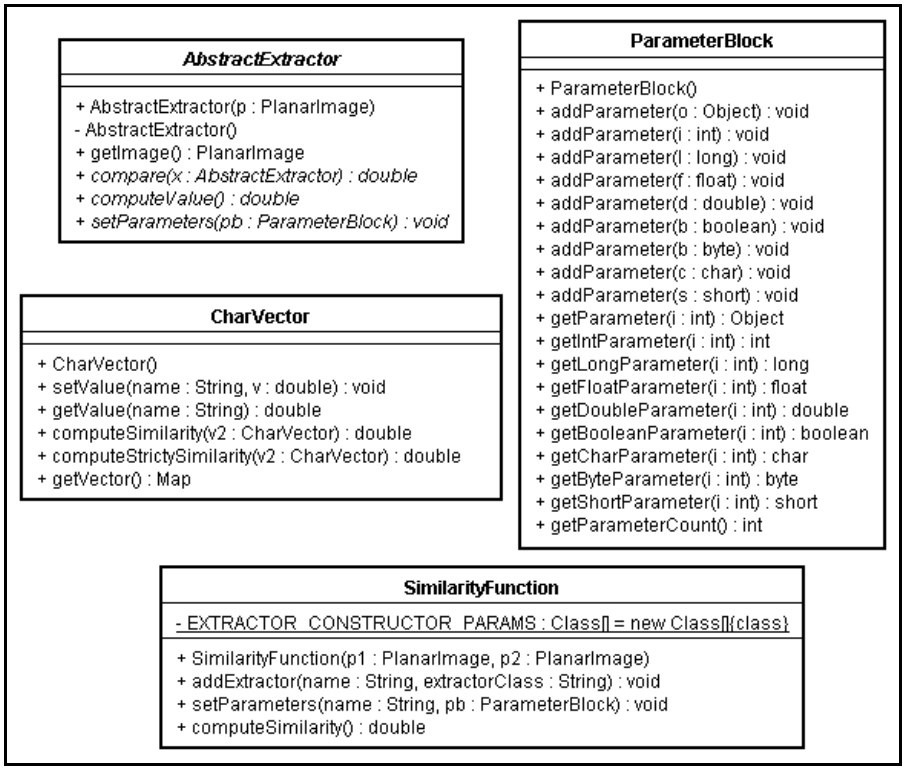


Figura 6 – Diagrama de classes do esquema genérico para CBIR utilizado

2.6 Extração de características das imagens mamográficas

Nesta seção são apresentados os extratores implementados, seguindo o esquema apresentado na seção anterior. Os códigos referentes à implementação encontram-se nos apêndices desta monografia. Foram implementados os extratores apresentados na Tabela 2.

Tabela 2 – Extratores implementados

Extrator	Descrição
Área	Soma dos *pixels* da área pertencente à mama dividida pela quantidade total de *pixels* da imagem.
Densidade	Média dos valores dos *pixels* da imagem dividida pelo maior

	valor de cinza possível (no caso 65536).
Forma	Divisão da quantidade de *pixels* do eixo *y* pela quantidade de *pixels* no eixo *x*. Esta quantidade de *pixels* é encontrada a partir do centro da mama.
Entropia	Representa o grau de desordem dos *pixels* da imagem.
Segundo Momento Angular	Extrai o nível de homogeneidade da imagem.
Contraste	Identifica as variações locais das imagens.

2.6.1 Extrator de Área

O extrator `ExtraiArea.java` tem um construtor implementado pela classe estendida `AbstractExtrator` que recebe uma imagem como parâmetro. Este extrator possui um método chamado `compare`, ilustrado na Figura 7, que recebe um outro extrator como parâmetro, utilizado para a comparação de duas imagens. Obtém o valor da característica para cada extrator (o atual e o passado por parâmetro) e faz a comparação dos valores, tirando a diferença dos dois valores. Se essa diferença for menor ou igual ao limite passado para a classe, as imagens são consideradas iguais, caso contrário, são consideradas diferentes.

```
public double compare(AbstractExtractor x) {
    if ( ! (x instanceof ExtraiArea) )
    {
        throw new IllegalArgumentException("ExtraiArea extractor required.");
    }
    ExtraiArea area = (ExtraiArea) x;
    double area1 = computeValue();
    double area2 = area.computeValue();
    double dif = area1 > area2 ? area1 - area2 : area2 - area1;
    if (dif > thr)
        return 0;
    else
        return 1;
}
```

Figura 7 – Trecho de código que implementa o método `compare` do extrator `ExtraiArea`

Outro método é o `computeValue`, ilustrado na Figura 8, que calcula a área da imagem passada ao construtor, abrindo-a e fazendo a soma dos *pixel*s não pretos da imagem, sendo que os *pixel*s pretos representam o fundo da mesma.

```
public double computeValue() {
    int[] im1 = getData();
    int area = 0;
    int total = 0;
    for( int i = 0; i < im1.length; i++) {
        if( im1[i] != 0 )
            area++;
        total++;
    }
    return (double) area / total;
}
```

Figura 8 – Trecho de código que implementa o método `computeValue` do extrator `ExtraiArea`

2.6.2 Extrator de Densidade

Assim como o `ExtraiArea.java`, o extrator `ExtraiDensidade.java` também tem seu construtor recebendo uma imagem como parâmetro definido pela classe estendida `AbstractExtrator`. Também implementa o método `compare` que é o responsável pela comparação da densidade da imagem passada ao construtor e da imagem do extrator passado como parâmetro para o método, cujo funcionamento é semelhante ao do `ExtraiArea.java`, calculando a densidade das duas imagens e analisando a diferença entre as densidades.

Para se obter o valor da densidade da imagem, é implementado o método `computeValue`, que calcula a densidade somando os valores dos *pixel*s da área pertencente efetivamente à mama, e dividindo esta soma pela quantidade total de *pixel*s da região, obtendo a média dos níveis de cinza da imagem. Para alcançar um valor normalizado, foi dividida a

média pelo maior valor de *pixel* possível (65536). Assim a densidade de qualquer imagem do banco sempre esta entre 0 e 1. O método `computeValue` está ilustrado na Figura 9.

```
public double computeValue() {
    int[] im = getData();
    long soma = 0, quant = 0;
    for (int i = 0; i < im.length; i++) {
        if (im[i] != 0) {
            quant++;
            soma += im[i];
        }
    }
    return (double) ( soma / quant ) / 65536;
}
```

Figura 9 – Trecho de código que implementa o método `computeValue` do extrator de densidade

2.6.3 Extrator de Forma

O extrator `ExtraiForma.java` também foi implementado da mesma maneira que os anteriores, no qual o método `computeValue` retorna a medida resultante da divisão da quantidade total de *pixel*s no eixo *y* pela quantidade total de *pixel*s no eixo *x*, todas considerando apenas a região pertencente à mama, como pode ser observado na Figura 10.

```
public double computeValue()
{
    PlanarImage im = getImage();
    int width = im.getWidth();
    int height = im.getHeight();
    SampleModel sm = im.getSampleModel();
    int nbands = sm.getNumBands();
    Raster inputRaster = im.getData();
    WritableRaster outputRaster = inputRaster.createCompatibleWritableRaster();
    int[] pixels = new int[nbands*width*height];
    ColorModel cm = im.getColorModel();
    TiledImage tiledImage = new TiledImage(0,0,width,height,0,0,sm,cm);
    inputRaster.getPixels(0,0,width,height,pixels);
    int offset, y1 = 0, y2 = 0, x1 = 0, x2 = 0, dy = 0, dx = 0, prim = height / 3;
    for (int h=0; h<height; h++) {
        int w=0;
        offset = h*width*nbands+w*nbands;
        for (int band=0; band<nbands; band++)
            if (pixels[offset+band] != 0)
                dy++;
    }
    if (dy == 0)
        for (int h=0; h<height; h++) {
            int w=width -1;
            offset = h*width*nbands+w*nbands;
            for (int band=0; band<nbands; band++)
                if (pixels[offset+band] != 0)
                    dy++;
        }
    int h = dy / 2;
    for (int w=0; w<width; w++) {
        offset = h*width*nbands+w*nbands;
        for (int band=0; band<nbands; band++)
            if (pixels[offset+band] != 0)
                dx++;
    }
    return (double)dy / dx;
}
```

Figura 10 – Trecho de código que implementa o método `computeValue` do extrator de forma

2.6.4 Características de textura obtidas a partir da matriz de co-ocorrência

As características a seguir têm o objetivo de fornecer uma maneira de medir a textura de uma imagem. Foram implementadas segundo as fórmulas apresentadas em Ferrero *et al.* (2006), obtidas a partir da matriz de co-ocorrência, implementada na classe `MatrizCoOcorrencia.java`. Segundo Marques *et al.* (2002) e Ferrero *et al.* (2006), a

matriz de co-ocorrência corresponde à freqüência de um nível de cinza na imagem ***A*** considerando uma distância em uma direção.

Ferrero *et al.* (2006) e Marques et al. (2002) explicam que a posição p(i,j) indica a freqüência de ocorrência de um particular par de níveis de cinza i e j, obtido a partir de uma distância e de um ângulo (direção). Considerando uma distância igual a um e um ângulo de 0°, a posição (0,1) da matriz de co-ocorrência é incrementada em uma unidade, caso o valor do elemento à esquerda ou à direita de um pixel de valor 0 for igual a 1, assim como ilustrado na Figura 11 (FERRERO et al., 2006).

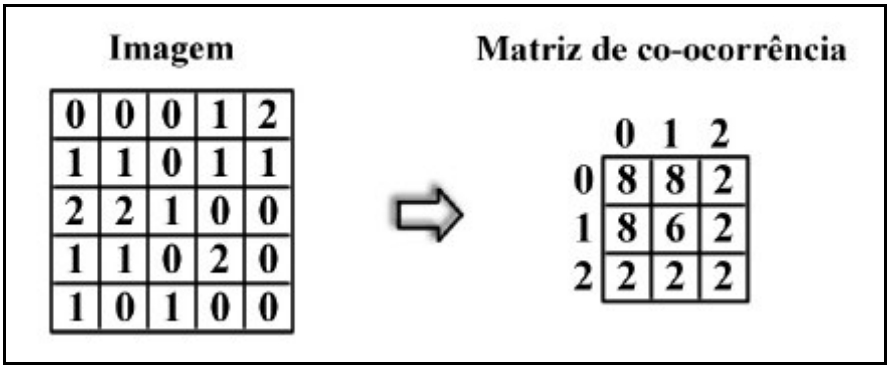


Figura 11 – Exemplo de imagem com três níveis de cinza e sua respectiva matriz de co-ocorrência para distância um e ângulo 0° (FERRERO *et al.*, 2006)

Para a normalização da matriz, cada posição é dividida pela soma de todas as posições (FERRERO *et al.*, 2006 e Marques *et al.*, 2002). No exemplo da Figura 11, cada uma das posições da matriz seria dividida por 40, que é a soma de todas as posições.

O problema encontrado para a implementação desta matriz é que ela é quadrada e de ordem igual ao número de níveis de cinza da imagem. Como as imagens utilizadas trabalham com 16 *bits* de resolução de contraste (65536 níveis de cinza), torna-se inviável computacionalmente a construção da referida matriz com a dimensão ideal. Tentando solucionar este problema, foi encontrado o intervalo de níveis de cinza realmente utilizado nas imagens, constatando-se um máximo de 3600 níveis (de 61935 até 65535). Portanto, para obter-se a referida matriz foi primeiramente retirado do valor do *pixel* 61935 (valor mínimo utilizado) e depois dividido o valor por 8, obtendo uma matriz quadrada de ordem 450,

solucionando-se, assim, o problema da dimensão da matriz. O código fonte da implementação da matriz encontra-se no Apêndice E desta monografia.

Na classe `MatrizCoOcorrencia.java` implementada foram criados dois métodos públicos para a obtenção da matriz: um passando por parâmetro apenas a distância e o ângulo e outro passando também o sinal do ângulo em questão. Estes métodos são `getMatriz(int dist, int an)` e `getMatriz(int dist, int an, String sinal)`.

O extrator `ExtraiEntropia.java` calcula a entropia da região segmentada segundo a fórmula apresentada em Ferrero *et al.* (2006), apresentada na equação (12), onde *p(i,j)* representa o elemento *(i,j)* normalizado, assim como nas equações 13 e 14.

$$Entropia = -\sum_{i}\sum_{j} p(i,j)\cdot \log(p(i,j)) \tag{12}$$

A partir das fórmulas apresentadas em Ferrero *et al.* (2006), também foram implementados os extratores `ExtraiSMA.java` e `ExtraiContraste.java`, que fazem a extração do segundo momento angular e contraste, respectivamente. As fórmulas de tais características estão apresentadas nas equações (13) e (14), onde *Ng* é a quantidade de níveis de cinza da imagem.

$$SegundoMomentoAngular = \sum_{i}\sum_{j} p(i,j)^2 \tag{13}$$

$$Contraste = \sum_{n=0}^{Ng-1} n^2 \left\{ \sum_{i}\sum_{j} p(i,j), se\ |i-j| = n \right\} \tag{14}$$

2.7 Recuperação das imagens

Para a recuperação das imagens, primeiramente foi verificado no banco de dados se o valor da característica selecionada para a imagem em questão já foi calculado ou não, caso já tivesse sido, este valor era recuperado, caso contrário, o valor era calculado e inserido no banco de dados. De posse destes valores, foi criado o vetor de característica da imagem de referência, e depois foi criado, para cada imagem do banco de dados, também seu respectivo vetor de característica, calculando a similaridade entre este vetor e o da imagem de referência. Para a criação e comparação dos vetores foi utilizada a classe `CharVector.java`, por meio dos métodos `computeSimilarity` e `computeStrictySimilarity`, que retornam a distância Euclidiana entre os dois vetores, considerando todas as características. A única diferença entre estes dois métodos é que o primeiro ignora caso em um dos vetores tenha uma característica que não esteja presente no outro vetor, e o segundo método apresentaria um erro para a mesma situação. Os valores retornados são armazenados em um vetor numérico, onde cada uma das posições representa a similaridade de uma das imagens com a imagem de referência. Foi criado este vetor para facilitar a recuperação, bastando ordenar o mesmo, e recuperar as *n* imagens com a menor distância Euclidiana, ou seja, com a maior similaridade, onde *n* é o número de imagens que se deseja retornar.

2.8 Implementação da interface do sistema

Na interface do sistema, ilustrada na Figura 12, o usuário pode selecionar quais características serão utilizadas para a recuperação e também qual será a imagem de referência.

Para isto foram criadas caixas de seleção de cada uma das características do banco (parte A da figura), e uma lista com todas as imagens do banco para a seleção da imagem de referência. Quando a imagem é selecionada, sua visualização é automaticamente disponibilizada abaixo da lista de seleção (parte B da figura). Também é permitido ao usuário selecionar quantas imagens deseja recuperar. Quando selecionado o botão `Procurar`, é feita a recuperação das imagens mais similares à imagem de consulta segundo as características selecionadas (parte C da figura).

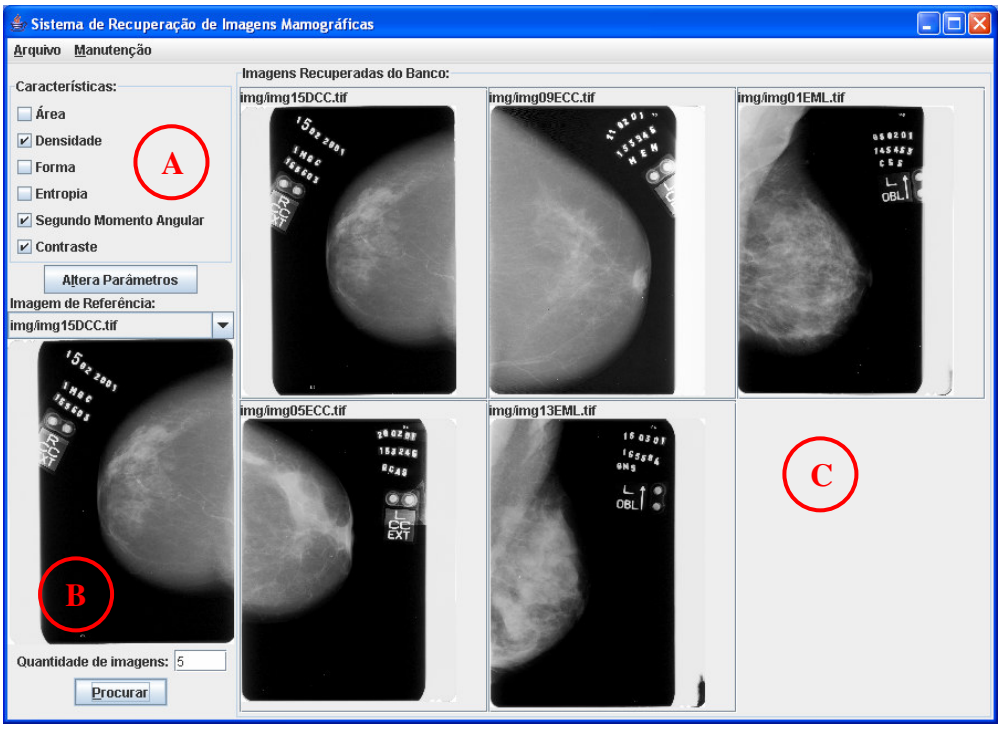


Figura 12 – Exemplo da Interface do sistema

Por meio do menu Arquivo, é permitida ao usuário a criação de uma nova consulta, que reabre o banco e atualiza as características presentes na tabela de características, assim como a lista de imagens também é atualizada com as imagens presentes no banco.

No menu Manutenção, o usuário pode escolher a tabela sobre a qual deseja alterar ou consultar os dados, Imagem ou Característica. Quando selecionada a opção Imagem do menu, é exibida a interface de manutenção das imagens, com as imagens já cadastradas e seus respectivos códigos e caminhos. Quando o usuário deseja inserir uma nova imagem ou alterar

uma já existente, basta digitar seu código e caminho (*path*) e selecionar o botão `Save`. Caso o código já exista no banco, é atualizado o caminho. Caso contrário, a nova imagem é inserida. Quando o usuário deseja excluir uma imagem do banco, basta digitar o seu código no campo referente ao código e selecionar o botão `Delete`. A interface de manutenção de características tem seu funcionamento semelhante à manutenção de imagens, diferenciando-se apenas de que na imagem é armazenado o caminho até a imagem e na manutenção de característica é armazenado o nome da mesma. Na Figura 13 são ilustradas as interfaces de manutenção das tabelas IMAGEM e CARACTERISTICA do banco de dados.

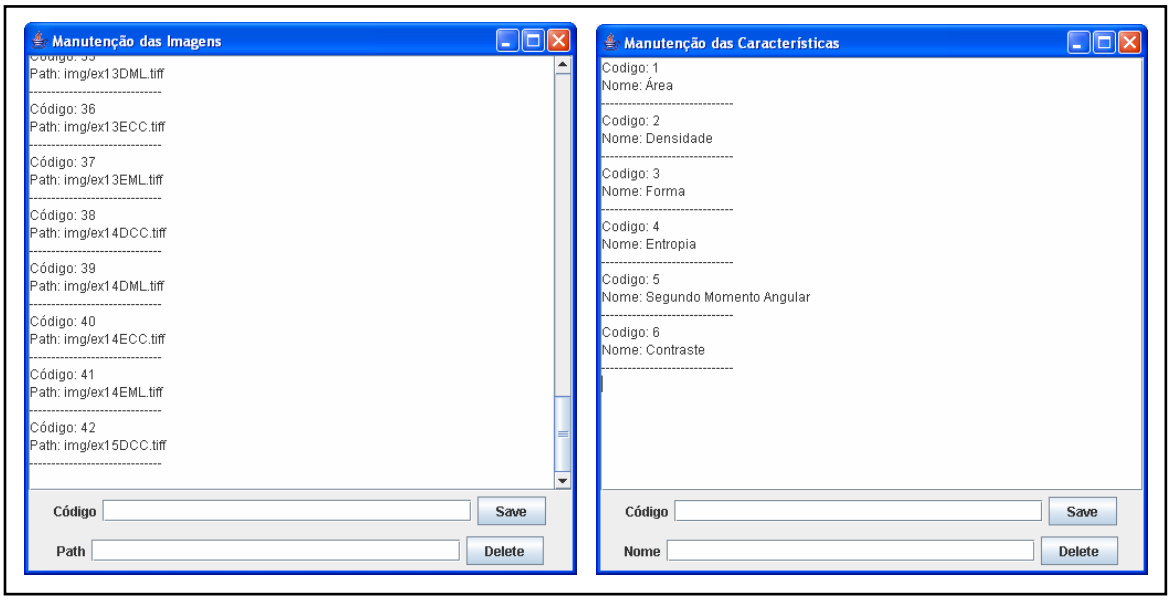


Figura 13 – Exemplo das interfaces de manutenção do banco

Como mostrado, o sistema implementado tem três características de textura, que para o seu cálculo é necessária a criação de uma matriz chamada de matriz de co-ocorrência. Para a obtenção desta matriz é necessário saber qual a distância e o grau, onde tais parâmetros influenciam na obtenção da referida matriz. Por isto, foi criada mais uma interface responsável por determinar tais parâmetros. O padrão do sistema é a distância de um *pixel* e ângulo de 0°. Na interface ilustrada na Figura 14, o usuário pode escolher tais informações apenas para as características de textura selecionadas.

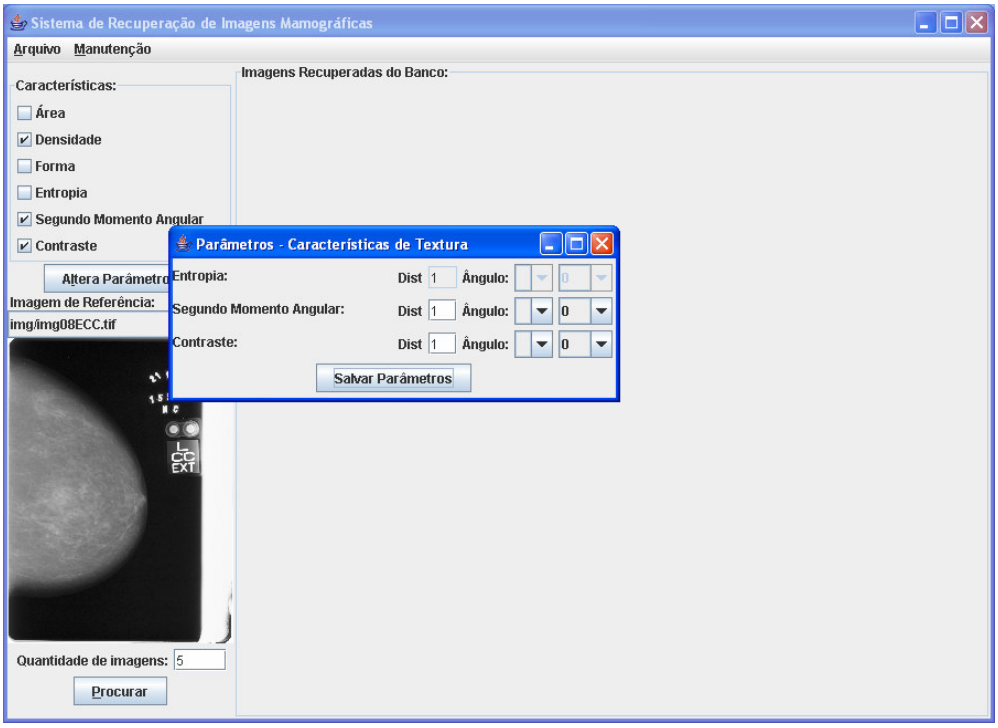


Figura 14 – Exemplo da interface para selecionar os parâmetros das características de textura

No Apêndice I desta monografia encontram-se todos os códigos fontes implementados para a criação das interfaces.

2.9 Considerações Finais

Neste capítulo foi apresenta toda a metodologia utilizada para a implementação do sistema. O próximo capítulo apresenta os testes e discussões aplicados sobre o sistema implementado conforme descrito neste capítulo.

CAPÍTULO 3 – RESULTADOS E DISCUSSÕES

Este capítulo apresenta e discute os testes e resultados obtidos, com a implementação do sistema proposto, assim como o desempenho do mesmo.

3.1 Conjuntos de teste criados

Para os testes deste sistema, foram criados três conjuntos de imagens a saber:

- Conjunto com 42 imagens fornecidas pelo LAPIMO da EESC/USP provenientes de diversos hospitais, citados no item 2.2 desta monografia. Estas imagens foram digitalizadas com 16 *bits* de resolução de contraste (65536 níveis de cinza). Este conjunto é composto por imagens de visão crânio-caudal e médio-lateral, que estão exemplificadas na Figura 15.

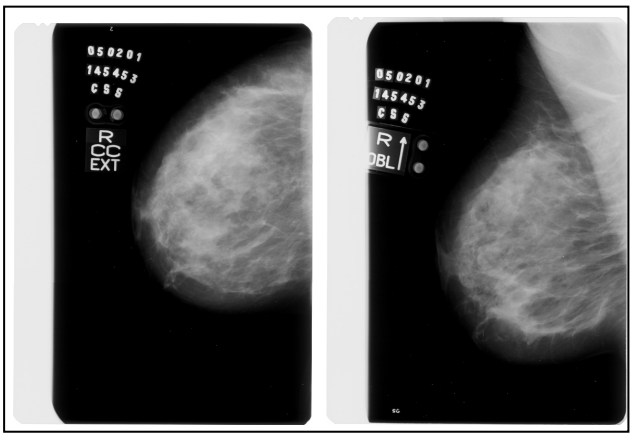


Figura 15 – Exemplo de imagens nas visões crânio-caudal e médio-lateral, respectivamente

- Conjunto com 20 imagens, sendo quatro imagens reais e de mesma paciente provenientes do primeiro conjunto, onde duas delas são da mama direita nas visões crânio-caudal e médio-lateral, e as outras duas da mama esquerda em

ambas visões. As outras 16 imagens foram obtidas a partir da variação de brilho das imagens citadas. Para cada imagem foram criadas outras quatro imagens com estas variações, sendo duas imagens mais claras (com aumento de 60 e 1000 níveis de cinza em cada pixel, ilustadas na Figura 16b e Figura 16c, respectivamente) e outras duas imagens mais escuras (com diminuição de 60 e 1000 níveis de cinza em cada pixel, ilustadas na Figura 16d e Figura 16e, respectivamente), como pode ser visualizado no exemplo da Figura 16 . A finalidade deste conjunto é permitir a verificação da eficiência das características em relação à variação do brilho das imagens.

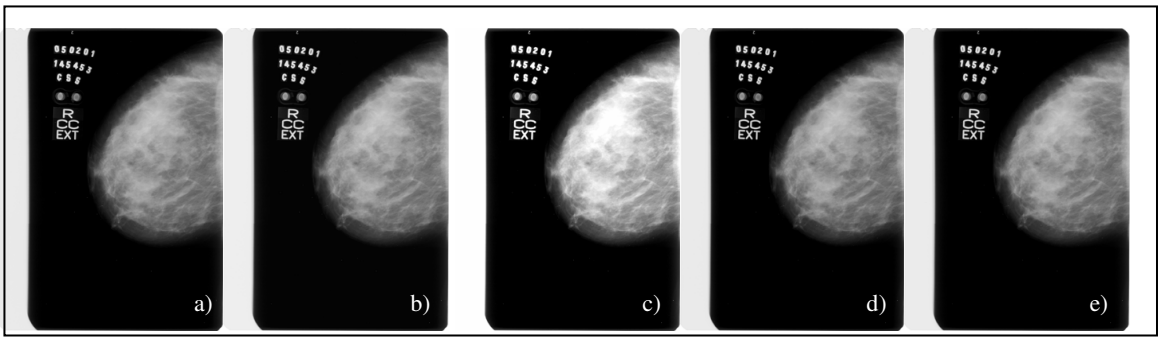


Figura 16 – Exemplo de uma imagem do segundo conjunto com suas variações: (a) imagem original; (b) imagem mais clara com 60 níveis de cinza; (c) imagem mais clara com 1000 níveis de cinza; (d) imagem mais escura com 60 níveis de cinza; (e) imagem mais escura com 1000 níveis de cinza.

- Conjunto com 16 imagens, contendo quatro imagens originais do primeiro conjunto mais três variações de cada imagem (como demonstrado na Figura 17a). Na primeira variação foi criada uma mancha (de mesmo tamanho e cor) em uma determinada posição da imagem, essa mancha foi inserida em todas as imagens do conjunto, procurando posicioná-las, aproximadamente, no mesmo local (um exemplo é apresentado na Figura 17b). Na segunda variação foi criada outra mancha que foi posicionada em locais diferentes das imagens do conjunto, mas com o mesmo tamanho e cor, assim como a anterior (conforme ilustrado na Figura 17c). Por último foi criada uma terceira mancha inserida em posições

diferenciadas nas imagens do conjunto, assumindo tamanhos diferentes em cada imagem, mas permanecendo com a mesma cor (como mostrado na Figura 17d). Este conjunto foi criado com o objetivo de verificar a eficiência das características em relação à recuperação de imagens com achados mamográficos semelhantes (nódulos, por exemplo), sendo útil para o teste das características de textura.

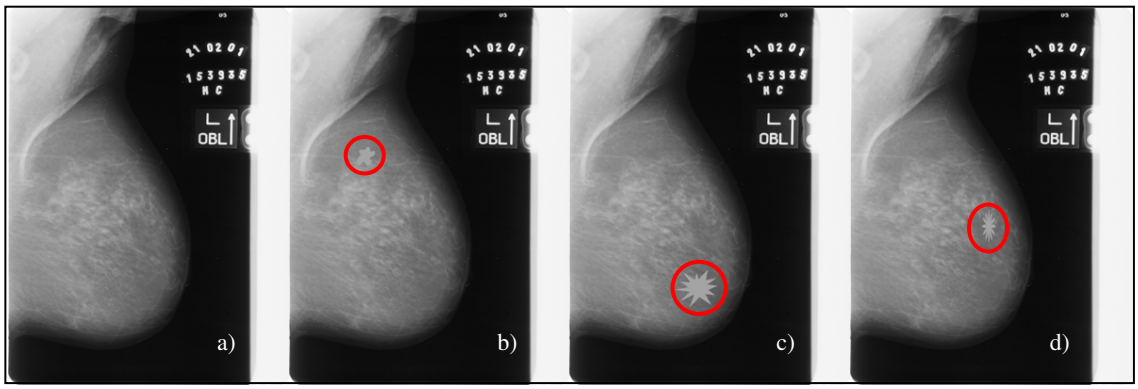


Figura 17 – Exemplo de uma imagem do terceiro conjunto com suas variações: (a) imagem original; (b) imagem com uma mancha de mesmo tamanho, cor e posicionamento; (c) imagem com outra mancha de mesmo tamanho e cor e posicionamento diferenciado; (d) imagem com uma mancha de mesma cor e tamanho e posicionamento variados;

Durante a execução dos testes foi detectado um erro na etapa de segmentação de uma das imagens do segundo conjunto proposto. A imagem em questão foi modificada, tendo sido realizada a soma de 1000 níveis de cinza em cada *pixel*. O erro ocorreu na identificação do lado da mama, uma vez que tendo a imagem seus valores clareados, os *pixels* foram identificados como o fundo da imagem. Na Figura 18 encontra-se a ilustração da imagem original e da imagem modificada antes e após a aplicação do algoritmo de eliminação de fundo.

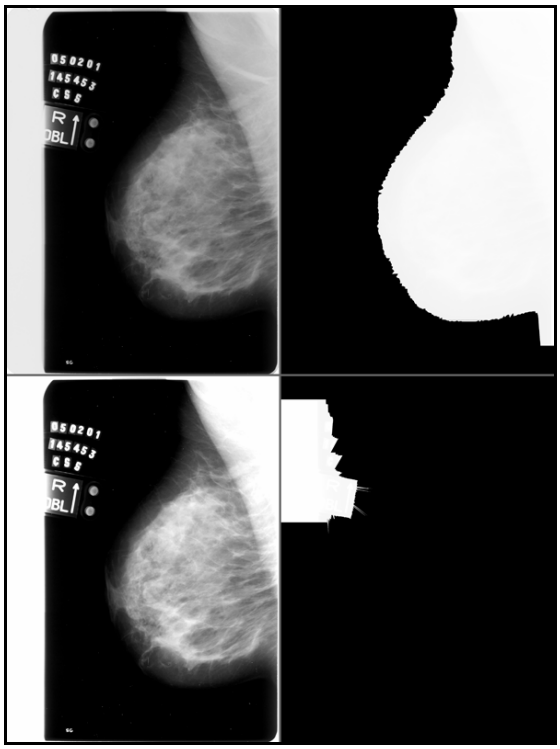


Figura 18 – Imagem original e clareada antes e após aplicação do algoritmo de eliminação de fundo

3.2 Precisão e Revocação

As medidas de precisão e revocação, também conhecidas por *precision-recall*, são utilizadas para avaliar a performance na recuperação de informação (BAEZA_YATES *et al.*, 1999) assim como de sistemas de recuperação de imagens (YAMAMOTO *et al.*, 1999). Estas medidas estão sendo largamente utilizadas na literatura para avaliar sistemas CBIR, como no caso de Marques *et al.* (2006), André *et al.* (2004), El-Naqa *et al.* (2004), Ji *et al.* (2003), Bueno (2002), El-Naqa *et al.* (2002) e Traina *et al.* (2002).

Considerando um exemplo onde A é a quantidade de imagens relevantes na base de dados, e B a quantidade de imagens do conjunto resposta de uma determinada busca de imagens por conteúdo, tem-se que:

- precisão representa a porcentagem de imagens relevantes que foram recuperadas. Matematicamente, $precisão = \frac{A \cap B}{B}$;
- revocação representa a porcentagem de imagens recuperadas que são relevantes. Matematicamente, $revocação = \frac{A \cap B}{A}$ (BUENO, 2002 e TRAINA *et al.*, 2002).

Para melhor compreensão, suponha a utilização de um determinado banco de dados de imagens para uma busca por conteúdo, onde um conjunto *A* possua as imagens relevantes e um conjunto *B* as imagens recuperadas, sendo $A = \{i_2, i_4, i_5, i_7, i_9\}$ e $B = \{i_5, i_6, i_7, i_1, i_3, i_8, i_9, i_{10}\}$. Considere a primeira imagem recuperada i_5. Esta imagem corresponde a 20% de todas as imagens relevantes, portanto tem uma revocação de 20%. Além disso, ela tem uma precisão de 100% (uma entre uma imagem recuperada é relevante), podendo-se dizer que esta imagem tem uma precisão de 100% em 20% de revocação. Considere agora a segunda imagem recuperada que é relevante, i_7. Esta imagem tem uma precisão de 66,67% (duas entre três imagens recuperadas são relevantes) em 40% de revocação (foram recuperadas duas imagens relevantes de um total de cinco imagens relevantes). Continuando com a análise descrita, têm-se todos os pontos necessários para se a curva de precisão *versus* revocação (PxR). Caso ao final do conjunto *B* não tenham sido recuperadas todas as imagens consideradas relevantes, o próximo nível de revocação será igual à zero.

O sistema implementado foi avaliado utilizando as curvas de precisão e revocação, que foram calculadas para as consultas de cada característica individualmente e algumas características agrupadas, que serão mostradas nas próximas seções deste capítulo.

Os testes para a avaliação do sistema foram criados sobre o segundo e terceiro conjuntos. O primeiro conjunto foi utilizado para avaliar inicialmente o retorno da recuperação a fim de verificar se os extratores estavam funcionando de forma correta. No entanto, para a construção das curvas de precisão *versus* revocação não havia uma maneira de definir quais as imagens relevantes. Os outros conjuntos possuíam quatro imagens vindas originalmente do primeiro conjunto e mais algumas imagens resultantes de variações nas quatro imagens iniciais. Portanto, para os testes de avaliação foram consideradas relevantes as imagens referentes à imagem original com as variações de cada conjunto, sendo esperadas a recuperação de 5 imagens para o segundo conjunto (a original e suas 4 variações) e 4 imagens para o terceiro conjunto (a original e as outras 3 representando suas respectivas variações).

3.3 Avaliação das Características Individuais

Nesta seção serão apresentadas as avaliações das características individuais do sistema em todos os conjuntos de teste criados.

3.3.1 Área

A área é uma medida que retorna o tamanho da mama na imagem. Tendo sido observado que as imagens possuíam o mesmo tamanho aproximado, em média 2048 *pixels* por 2750 *pixels*, calculou-se a área da mama de forma proporcional, dividindo-se a área encontrada pelo tamanho total da imagem.

Foi executada uma busca pela área no primeiro conjunto apenas para a verificação do desempenho da mesma em um conjunto com imagens reais, mas este desempenho não teve como ser medido, pois não havia como se avaliar se as imagens retornadas poderiam ser consideradas relevantes ou não. Um exemplo desta recuperação pela área no primeiro conjunto pode ser visualizado na Figura 19a.

Como esperado, os testes realizados no segundo conjunto mostraram a área como uma medida satisfatória por recuperar as imagens esperadas, uma vez que retornou as imagens iguais mais claras e mais escuras, não recuperando apenas a imagem que apresentou falha na fase de segmentação, conforme discutido anteriormente, conforme ilustrado na Figura 19b.

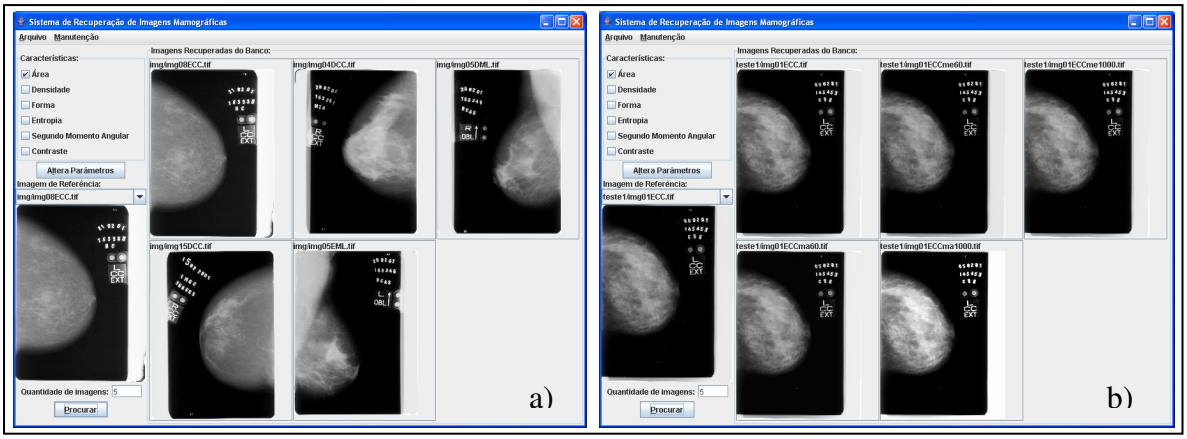


Figura 19 – Interfaces da recuperação de cinco imagens pela característica “área” em dois conjuntos: (a) primeiros conjunto; (b) segundo conjunto

Na Figura 20 é apresentado o gráfico de precisão e revocação da característica “área” para o segundo conjunto de imagens, por meio do qual pode-se perceber que a curva da consulta b não recuperou todas as imagens esperadas, sendo esta a curva da imagem que apresentou erro na etapa de segmentação. A curva das consulta a, c, d refere-se à curva das três outras imagens utilizadas como referência.

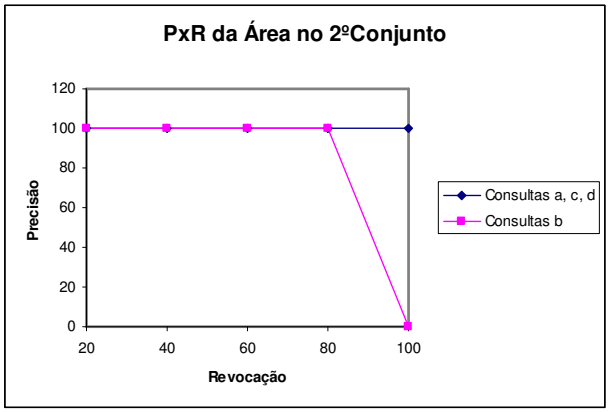


Figura 20 – Precisão e revocação da característica “área” para o segundo conjunto de imagens.

O gráfico do terceiro conjunto, apresentado na Figura 21, mostra apenas uma curva, que corresponde à curva de todas as consultas realizadas para o teste. Como pode-se perceber as imagens recuperadas eram relevantes, levando à conclusão de que a característica “área” teve um ótimo desempenho para os testes executados.

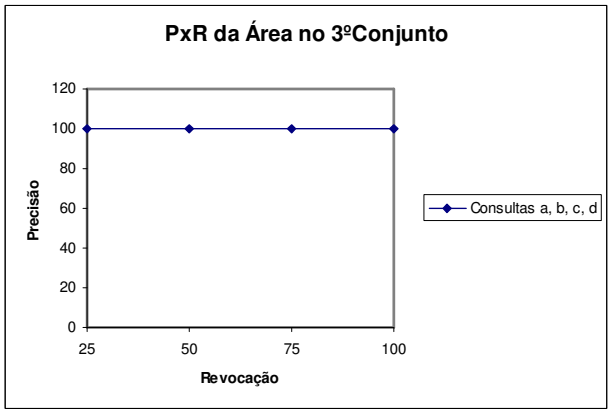


Figura 21 – Precisão e revocação da característica “área” para o terceiro conjunto

3.3.2 Densidade

A densidade é uma característica capaz de retornar aquelas imagens com valores semelhantes de níveis de cinza. Nos testes realizados observou-se que as imagens de visão

médio-lateral apresentam o músculo peitoral, possuindo naturalmente uma densidade maior que as imagens da mesma paciente em visão crânio-caudal, o que interfere, portanto, na obtenção da densidade total da mama. Logo, nos testes aplicados no primeiro conjunto, quando fornecida uma imagem de mama pouco densa (mais escura) na visão médio-lateral, foram recuperadas imagens de mamas com áreas mais densas de visão crânio-caudal conforme ilustrado na Figura 22. Como a densidade é uma característica que depende do valor de cada *pixel* da imagem, no segundo conjunto esta medida é afetada, pois as imagens deste conjunto sofreram alterações nos valores de seus *pixels*, nem sempre sendo recuperadas as mesmas imagens com brilhos diferentes. No terceiro conjunto de imagens foram recuperadas as imagens esperadas, mesmo que o foco de tal conjunto não era o teste desta característica específica.

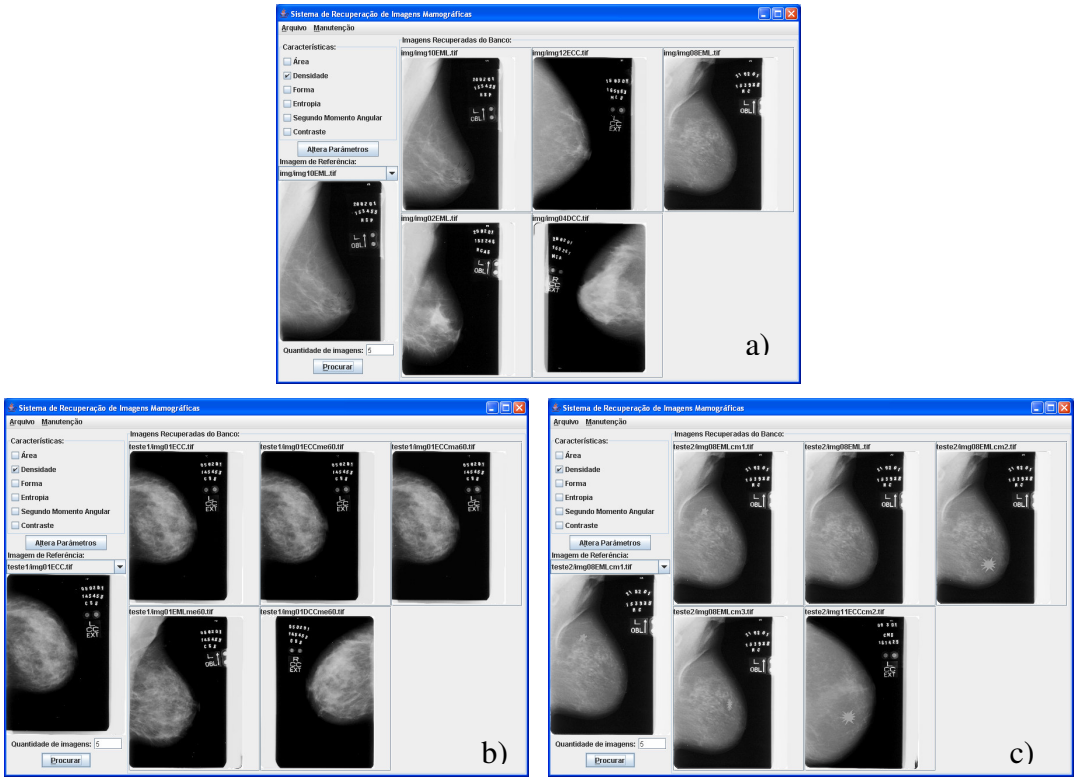


Figura 22 – Interfaces da recuperação de cinco imagens pela característica “densidade” nos três conjuntos: (a) primeiros conjunto; (b) segundo conjunto; (b) terceiro conjunto

Como já dito anteriormente, não foram geradas as curvas de PxR para o primeiro conjunto pela dificuldade em determinar quais seriam as imagens relevantes para cada

consulta a ser realizada. O gráfico de precisão e revocação da característica “densidade” do segundo conjunto é apresentado na Figura 23. Este gráfico tem quatro curvas, onde cada uma delas corresponde à curva de uma das quatro consultas realizadas. Por meio dos gráficos percebe-se que para este conjunto a densidade não teve um desempenho satisfatório, recuperando todas as imagens esperadas em apenas uma das quatro consultas realizadas.

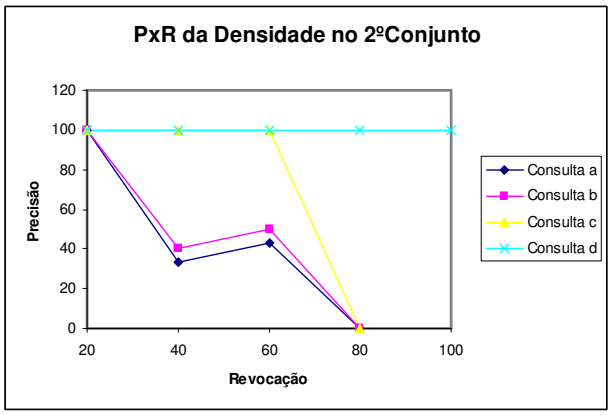


Figura 23 – Precisão e revocação da característica “densidade” para o segundo conjunto

O gráfico de PxR da característica “densidade” aplicado sobre o terceiro conjunto mostra apenas uma curva, que corresponde à curva de todas as consultas executadas com cada uma das imagens utilizadas para o teste. Nota-se que todas as imagens consideradas relevantes foram recuperadas, apresentando um desempenho plenamente satisfatório. Este gráfico pode ser visualizado na Figura 24.

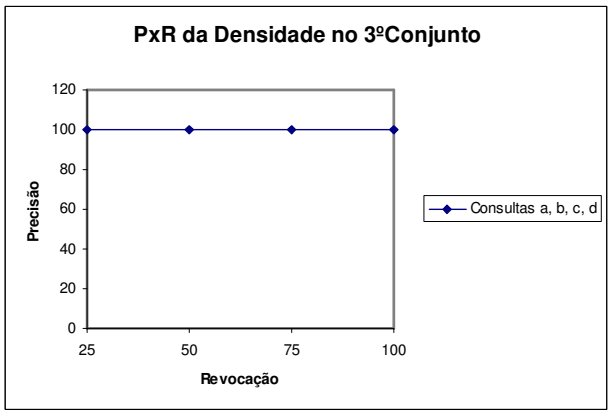


Figura 24 – Precisão e revocação da característica “densidade” para o terceiro conjunto

3.3.3 Forma

A forma é uma medida que obtém o grau de alongamento da mama, ou seja, o quão alongada a mama é. Quanto maior o valor da forma (divisão da distância no eixo *y* pela distância no eixo *x* a partir do centro da mama) mais alongada é a mama. Quanto menor o valor desta característica, mais arredondada é a mama. No primeiro conjunto de imagens, a forma mostrou-se uma característica capaz de retornar imagens de mesmas dimensões, como demonstrado na Figura 25a. Já no segundo conjunto foi capaz de recuperar imagens de mesma visão, como ilustrado na Figura 25b e Figura 25c.

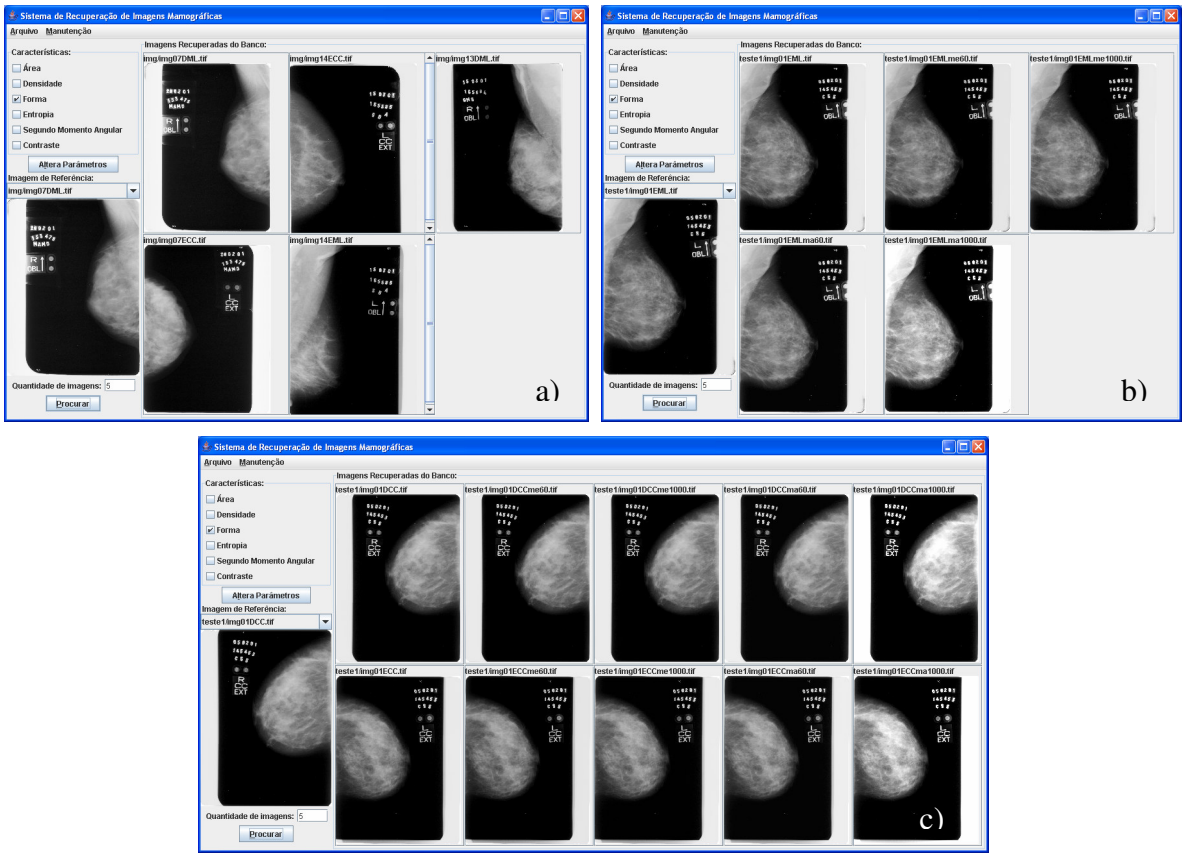


Figura 25 – Interfaces da recuperação de imagens pela característica “densidade” em dois conjuntos: (a) 5 imagens recuperadas do primeiro conjunto; (b) 5 imagens recuperadas do segundo conjunto; (b) 10 imagens recuperadas do segundo conjunto

Na Figura 26 é apresentado o gráfico de precisão e revocação da característica "forma" para o segundo conjunto, onde existem duas curvas, uma curva representado o resultado de três consultas com imagens distintas e outra curva representado o resultado da consulta que utilizou a imagem que apresentou erro na etapa de segmentação.

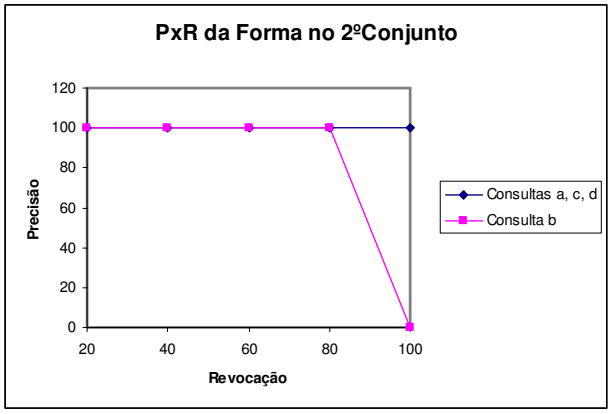


Figura 26 – Precisão e revocação da característica "forma" para o segundo conjunto

O gráfico para o terceiro conjunto, Figura 27, possui apenas uma curva representando as quatro consultas realizadas e apresentou um desempenho plenamente satisfatório, recuperando todas as imagens consideradas relevantes.

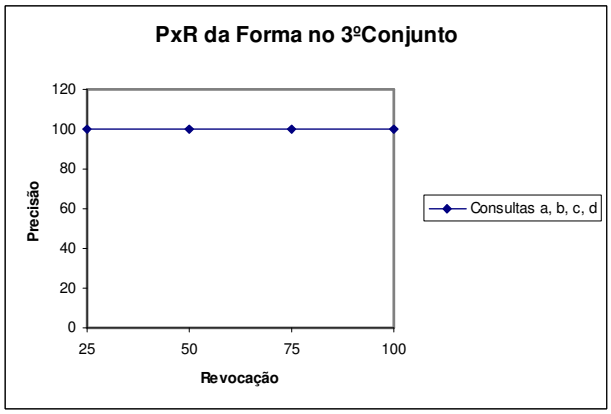


Figura 27 – Precisão e revocação da característica "forma" para o terceiro conjunto

3.3.4 Entropia

A entropia é uma característica de textura capaz de recuperar as imagens com os mesmos graus de desordem dos *pixels*, como foi observado no primeiro conjunto de imagens, Figura 28a, as imagens recuperadas apresentando texturas parecidas, com tons semelhantes e agrupamentos similares.

No segundo conjunto de imagens, a característica apresentou um comportamento esperado, recuperando as imagens correspondentes à imagem de consulta com as modificações (mais claras ou mais escuras) ou as imagens de mesma paciente e visão, diferenciando apenas o lado da mama, uma vez que estas modificações não apresentam grandes diferenças de textura, ou seja, de desordem dos *pixels*, como pode ser observado na Figura 28b.

Na Figura 28c pode-se observar que no terceiro conjunto de teste, foram recuperadas todas as imagens correspondentes a imagem de referências com as modificações inseridas (manchas de vários tamanhos e em várias posições).

Pode-se concluir que esta característica é interessante para recuperar imagens médicas semelhantes, mesmo que estas tenham sido adquiridas com características diferentes (níveis de corrente e tensão do equipamento médico, por exemplo), o que constitui um aspecto interessante para avaliar imagens médicas, principalmente em sistemas de auxílio ao diagnóstico, independente do equipamento utilizado para a aquisição.

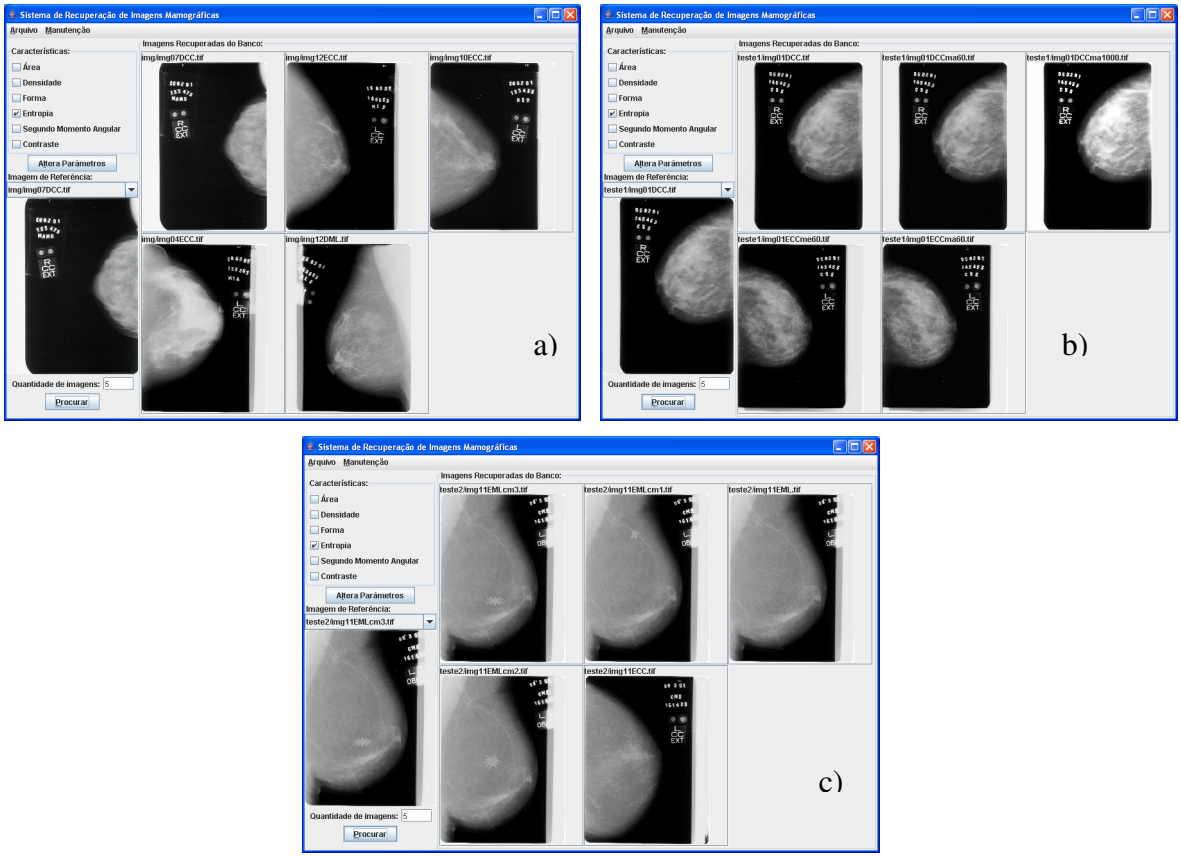


Figura 28 – Interfaces da recuperação de cinco imagens pela característica “entropia” nos três conjuntos: (a) primeiros conjunto; (b) segundo conjunto; (b) terceiro conjunto

Na Figura 29 é apresentado o gráfico de PxR da característica “entropia” para o segundo conjunto, apresentando quatro curvas, onde cada uma representa uma consulta feita com imagens distintas. Como pode ser observado, neste caso a entropia não apresentou um desempenho satisfatório, recuperando inicialmente as imagens esperadas, porém não recuperando todas as imagens relevantes.

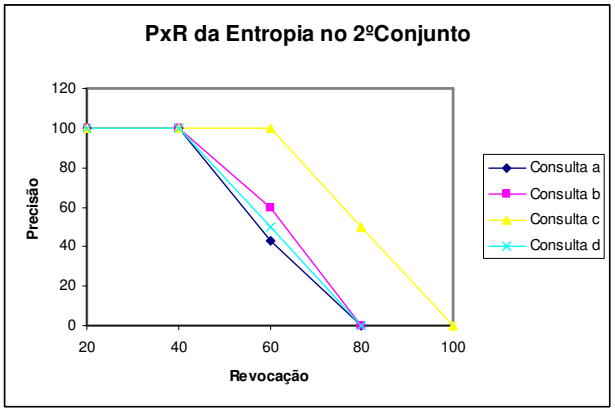


Figura 29 – Precisão e revocação da característica “entropia” para o segundo conjunto

O gráfico para o terceiro conjunto (Figura 30) mostra que para as quatro consultas, todas as imagens consideradas foram recuperadas em ordem, apresentando um ótimo desempenho.

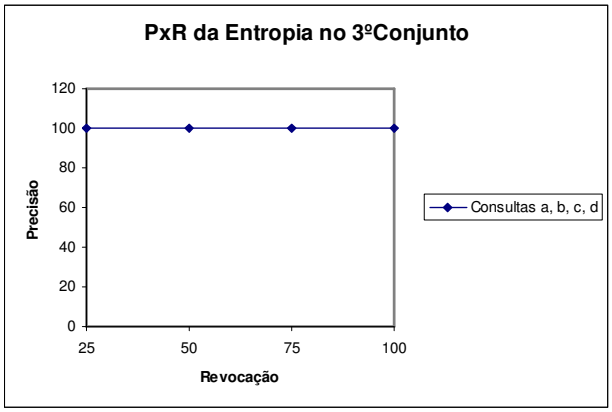


Figura 30 – Precisão e revocação da característica “entropia” para o terceiro conjunto

3.3.5 Segundo Momento Angular

O Segundo Momento Angular recuperou imagens com o mesmo nível de homogeneidade, conforme esperado para todos os conjuntos de testes. Observou-se que o mesmo grau de homogeneidade não quer dizer imagens com o mesmo grau de densidade. Por exemplo, pode ser recuperada uma imagem homogênea e muito densa a partir de uma imagem de referência pouco densa, mas também homogênea. Na Figura 31 podem ser observadas as interfaces com os resultados da recuperação das imagens utilizando os três conjuntos.

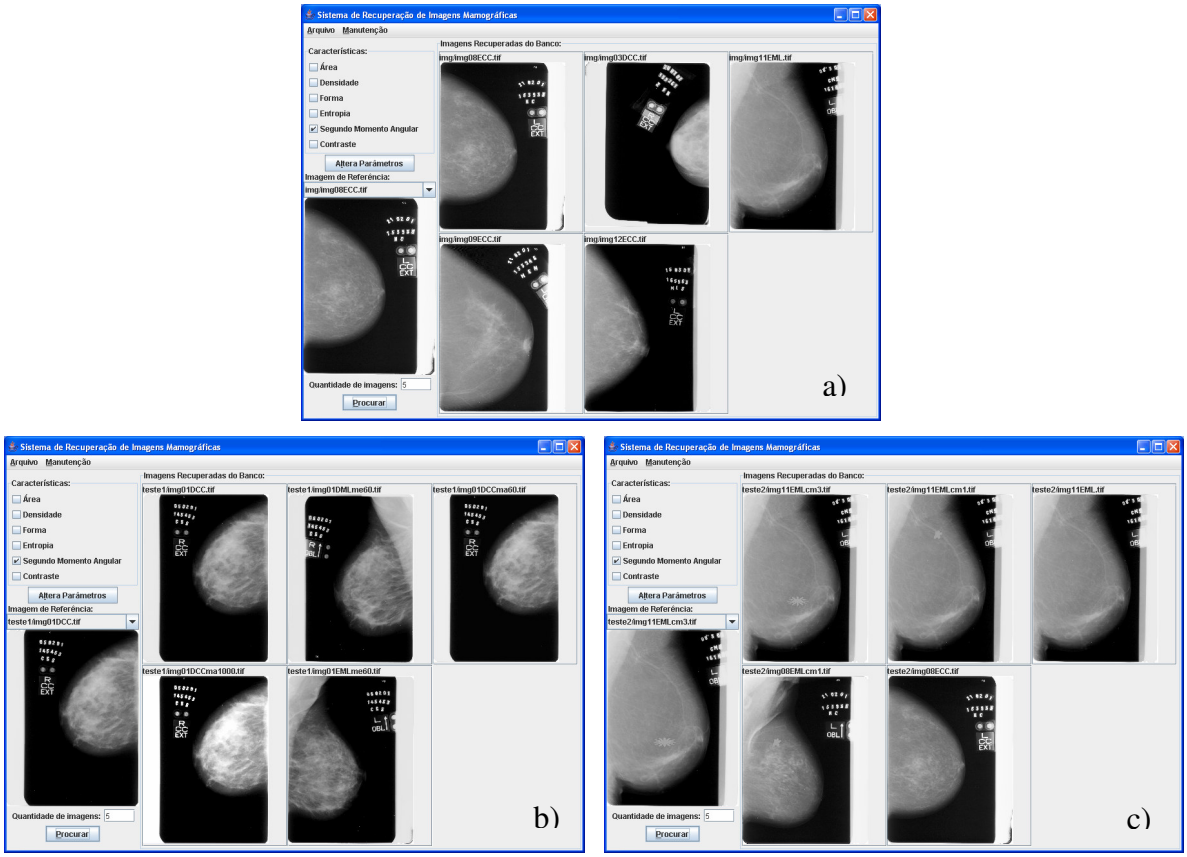


Figura 31 – Interfaces da recuperação de cinco imagens pela característica “segundo momento angular” nos três conjuntos: (a) primeiros conjunto; (b) segundo conjunto; (b) terceiro conjunto

Na Figura 32 é apresentado o gráfico de PxR da característica “segundo momento angular” para o segundo conjunto, onde apenas uma consulta apresentou um desempenho mediano (consulta d), as outras consultas apresentaram desempenhos inferiores e em nenhum dos casos, todas as imagens consideradas relevantes foram recuperadas.

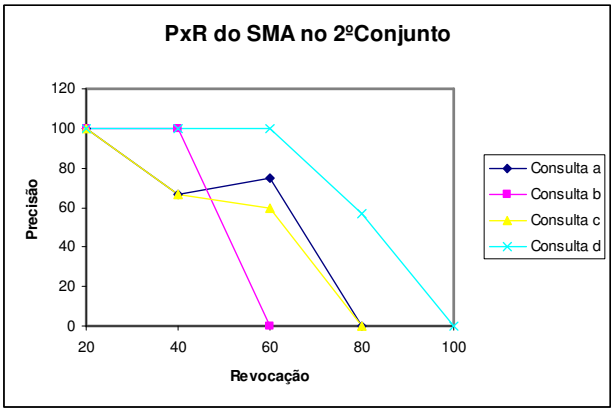


Figura 32 – Precisão e revocação da característica “segundo momento angular” para o segundo conjunto

O gráfico do terceiro conjunto, apresentado na Figura 33 tem quatro curvas, cada uma representando uma consulta realizada. Pode-se notar neste conjunto de teste o segundo momento angular apresentou um melhor desempenho do que no conjunto anterior, recuperando em três das quatro consultas todas as imagens consideradas relevantes.

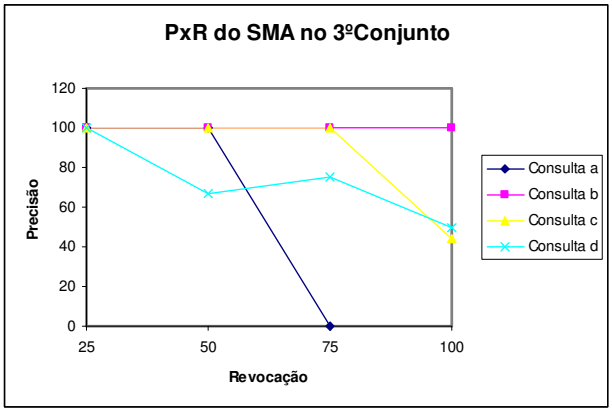


Figura 33 – Precisão e revocação da característica "segundo momento angular" para o terceiro conjunto

3.3.6 Contraste

O contraste identifica imagens com variações locais semelhantes, como pode ser verificado na Figura 34a, lembrando que os testes no primeiro conjunto foram feitos para se verificar as imagens retornadas em consultas com imagens reais, não sendo utilizados para a verificação do desempenho do sistema, devido à dificuldade de se determinar as imagens relevantes para este conjunto.

No segundo conjunto, Figura 34b, pode-se observar que esta característica apresentou um desempenho satisfatório, não retornando a imagem com maior brilho, pois esta teve uma maior interferência no contraste da imagem, provavelmente devido à saturação da escala de níveis de cinza. No terceiro conjunto, Figura 34c, foram recuperadas as imagens com as variações corretamente.

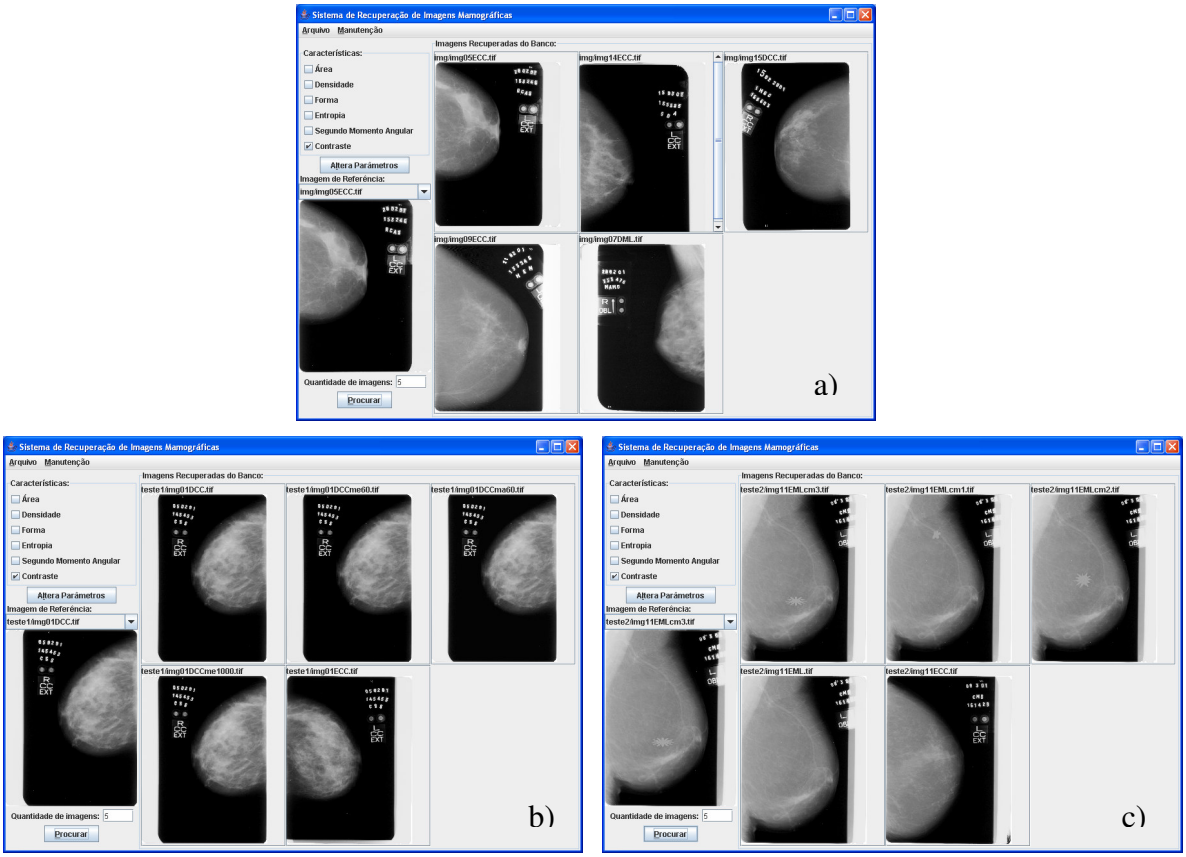


Figura 34 – Interfaces da recuperação de cinco imagens pela característica “contraste” nos três conjuntos: (a) primeiros conjunto; (b) segundo conjunto; (b) terceiro conjunto

Na Figura 35 é mostrado o gráfico de PxR da característica “contraste” para o segundo conjunto, contendo quatro curvas referentes às consultas realizadas. Percebe-se que o contraste não apresentou um bom desempenho, não recuperando em nenhuma das consultas realizadas todas as imagens consideradas relevantes e também possuindo baixos níveis de precisão.

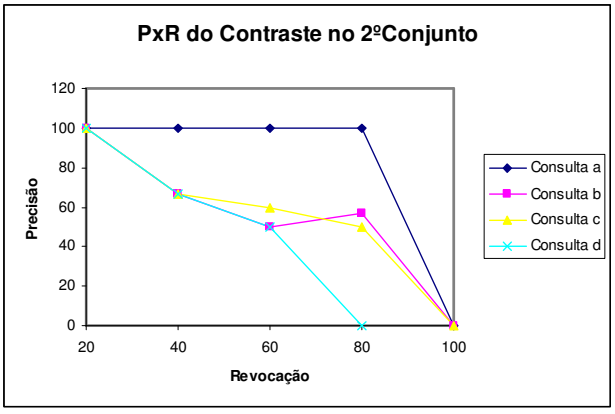


Figura 35 – Precisão e revocação da característica “contraste” para o segundo conjunto

O gráfico do terceiro conjunto, Figura 36, também tem quatro curvas, representando as quatro consultas. Este gráfico apresenta um desempenho melhor do que o do segundo conjunto, recuperando em três consultas todas as imagens consideradas relevantes, porém os valores de precisão ainda são muito baixos, não apresentando, portanto, um desempenho ótimo.

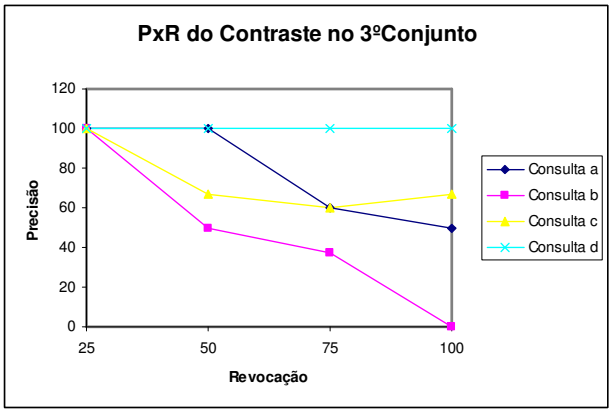


Figura 36 – Precisão e revocação da característica “contraste” para o terceiro conjunto

3.4 Avaliação das Características Agrupadas

Para avaliação das características agrupadas do sistema proposto foram estabelecidos sete agrupamentos. As características implementadas foram separadas nas características de textura (entropia, segundo momento angular e contraste) e demais características (área, densidade e forma). Em seguida, as características foram agrupadas duas a duas, totalizando três agrupamentos das características de textura e três agrupamentos das características que não eram de textura. O último agrupamento refere-se ao teste de todas as características juntas.

3.4.1 Área, Densidade e Forma

A Figura 37 ilustra o gráfico de PxR do agrupamento das características de área e densidade para o segundo conjunto. Esse gráfico possui três curvas, onde a primeira representa as curvas da consulta a e da consulta c, e as outras duas representam as curvas das demais consultas. Por este gráfico percebe-se que o agrupamento destas características apresentou um desempenho satisfatório, não recuperando todas as imagens relevantes em apenas uma consulta.

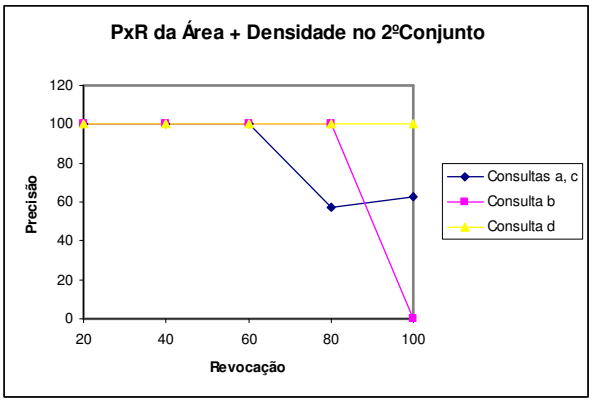


Figura 37 – Precisão e revocação das características de área e densidade agrupadas para o segundo conjunto

O gráfico da PxR das características área e densidade agrupadas para o terceiro conjunto está representado na Figura 38, onde há apenas uma curva que representa as quatro consultas realizadas, apresentando um desempenho plenamente satisfatório.

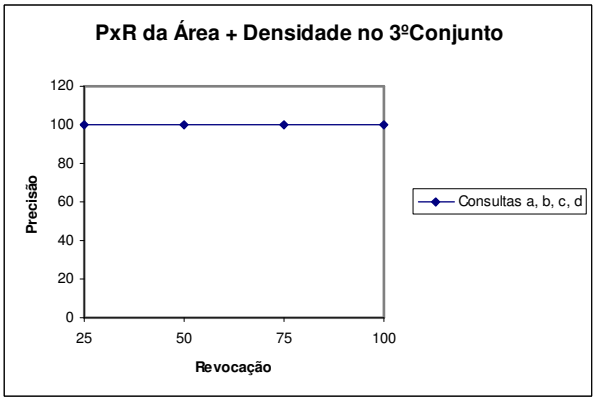


Figura 38 – Precisão e revocação das características de área e densidade agrupadas para o terceiro conjunto

Os agrupamentos de característica densidade + forma e área + forma apresentaram o mesmo comportamento em ambos os conjuntos de teste avaliados. Conforme ilustrado na Figura 39, o gráfico de PxR do referido agrupamento mostra que as imagens recuperadas eram relevantes, com exceção da imagem que apresentou erro na etapa de segmentação, levando à conclusão de que estas características agrupadas apresentaram um desempenho satisfatório.

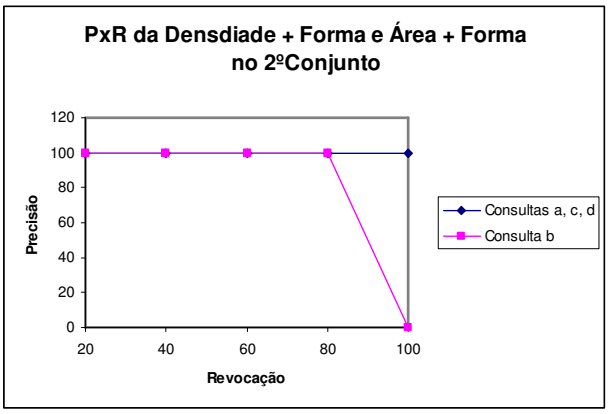


Figura 39 – Precisão e revocação das características de área + forma e densidade + forma agrupadas para o segundo conjunto

Na Figura 40 está sendo ilustrado o gráfico de PxR dos agrupamentos de características densidade + forma e área + forma para o terceiro conjunto, que apresentaram um desempenho plenamente satisfatório assim como a gráfico do agrupamento área + densidade para o mesmo conjunto de imagens.

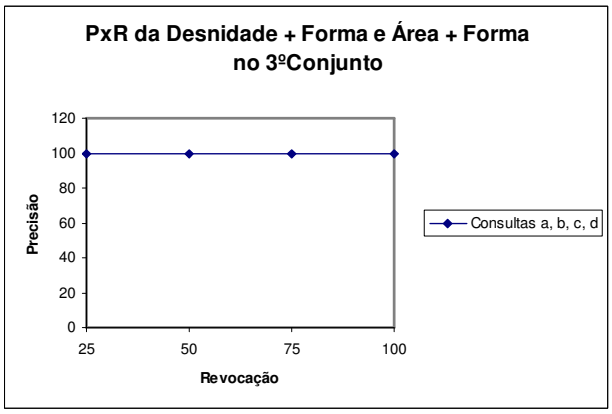


Figura 40 – Precisão e revocação das características de área + forma e densidade + forma agrupadas para o terceiro conjunto

Como as três características aqui agrupadas apresentaram a mesma curva para terceiro conjunto, os agrupamentos delas também apresentaram esta mesma curva.

Para o segundo conjunto, a área e a forma também tiveram as mesmas curvas, portanto o agrupamento área + forma apresentou esta mesma curva. A única característica isolada que apresentou curvas diferentes foi a densidade, mas mesmo assim, o agrupamento densidade + forma apresentou a mesma curva da área e forma, isso se dá pelo fato de a forma ter uma maior influencia sobre a densidade, sobressaindo-se sobre a da densidade.

3.4.2 Características de textura

Conforme ilustrado na Figura 41, o gráfico de PxR do agrupamento das características de entropia e segundo momento angular mostra que no segundo conjunto nenhuma das consultas realizadas apresentaram todas as imagens relevantes esperadas, levando à conclusão de que estas características apresentaram um baixo desempenho para os testes criados. Pode-se observar que as curvas apresentadas neste agrupamento são idênticas as apresentadas pela característica entropia, levando à conclusão de que esta característica tem maior influência quando comparada ao segundo momento angular.

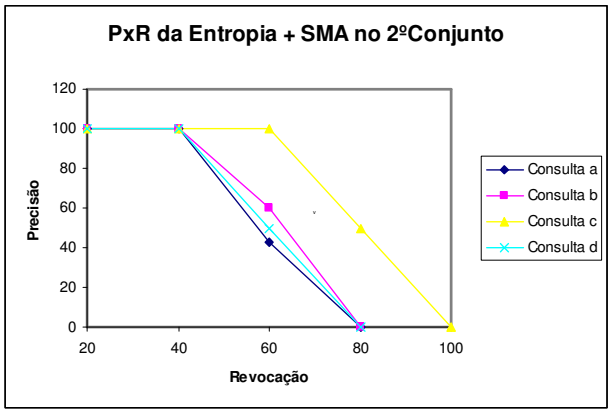


Figura 41 – Precisão e revocação das características entropia e segundo momento angular agrupadas para o segundo conjunto

Assim como no gráfico do segundo conjunto, o gráfico do terceiro conjunto para o agrupamento das características de entropia e segundo momento angular (Figura 42) também é igual ao apresentado pela entropia, confirmando que esta característica tem maior influência sobre o segundo momento angular, sobressaindo-se sempre.

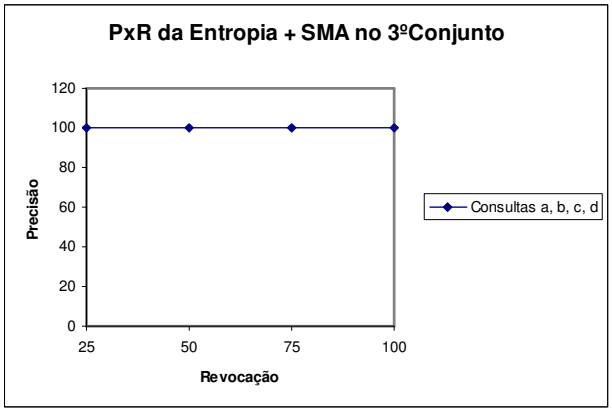


Figura 42 – Precisão e revocação das características entropia e segundo momento angular agrupadas para o terceiro conjunto

Os gráficos apresentados nas Figura 43 e Figura 44 mostram que considerando o agrupamento das características segundo momento angular e contraste, foram recuperadas mais imagens relevantes no terceiro conjunto do que no segundo, porém o segundo conjunto apresenta maiores níveis de precisão, decaindo apenas por não ter recuperado todas as imagens consideradas relevantes.

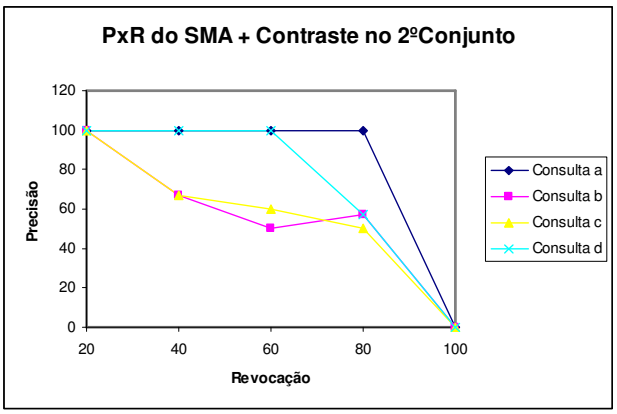


Figura 43 – Precisão e revocação das características segundo momento angular e contraste agrupadas para o segundo conjunto

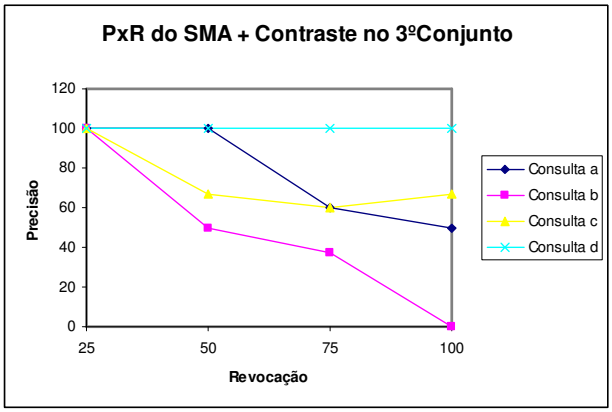


Figura 44 – Precisão e revocação das características segundo momento angular e contraste agrupadas para o terceiro conjunto

Conforme ilustrado nas Figura 45 e Figura 46, os gráficos de precisão e revocação do agrupamento das características de entropia e contraste mostram que o terceiro conjunto recuperou em todas as consultas realizadas todas as imagens relevantes e o segundo conjunto não recuperou todas as imagens em nenhuma das consultas, apresentando, portanto um melhor desempenho no terceiro conjunto do que no primeiro.

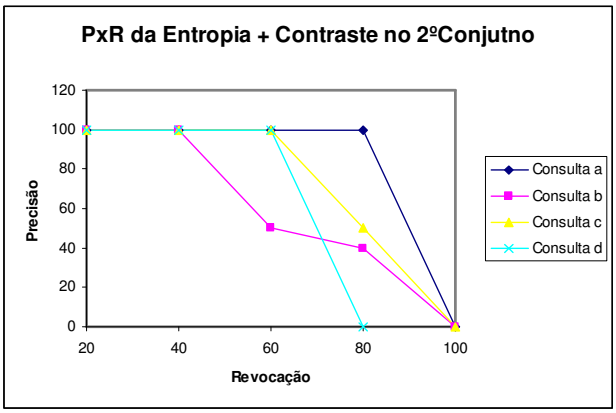


Figura 45 – Precisão e revocação das características entropia e contraste agrupadas para o segundo conjunto

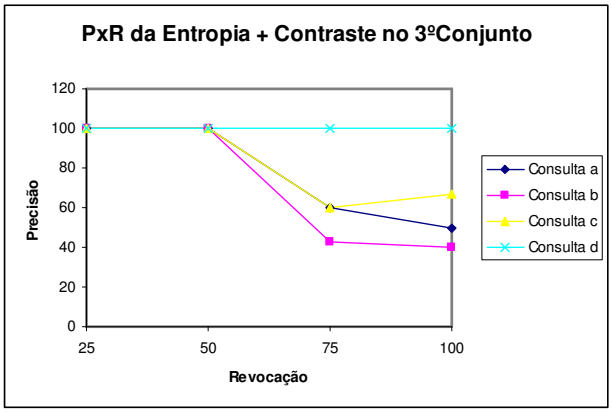


Figura 46 – Precisão e revocação das características entropia e contraste agrupadas para o terceiro conjunto

3.4.3 Desempenho com todas as características

Conforme ilustrado na Figura 47, o gráfico de precisão e revocação do agrupamento de todas as características para o segundo conjunto mostra que o desempenho foi satisfatório com altos níveis de precisão e não recuperando todas as imagens relevantes em apenas uma das consultas.

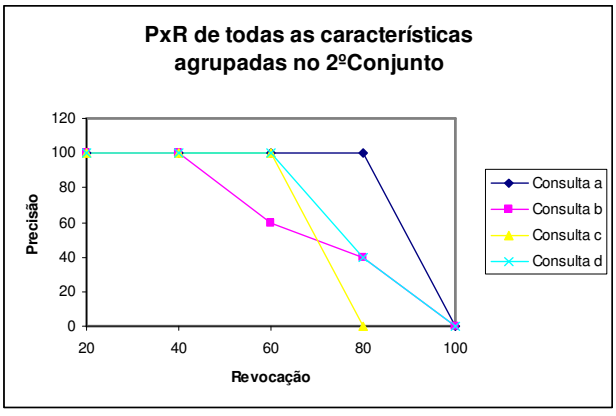


Figura 47 – Precisão e revocação te todas as características agrupadas para o segundo conjunto

No terceiro conjunto, o desempenho de todas as características agrupadas apresentou níveis de precisão mais baixos do que no segundo conjunto, porém em todas as conjuntas foram recuperadas todas as imagens consideradas relevantes.

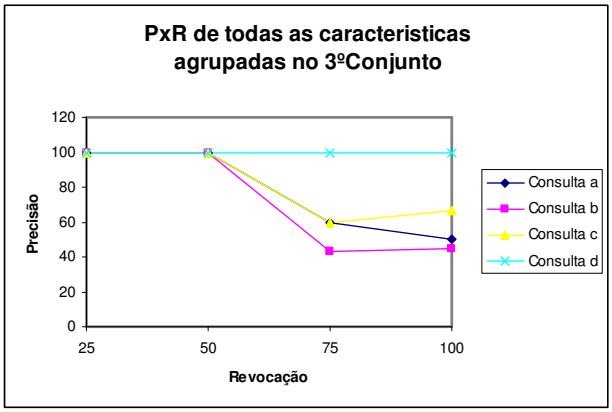


Figura 48 – Precisão e revocação te todas as características agrupadas para o terceiro conjunto

3.5 Média de Precisão e Revocação para todos os testes

Na Tabela 3 estão relacionadas as características e agrupamentos de características, juntamente com seus respectivos valores de precisão média para o segundo e terceiro conjuntos e revocação média para ambos os conjuntos. A discussão dos valores e os gráficos são apresentados nas seções de cada característica.

Tabela 3 – Precisão e Revocação médias (em porcentagem %)

Característica(s)	Precisão Média 2ºConjunto	Revocação Média 2ºConjunto	Precisão Média 3ºConjunto	Revocação Média 3ºConjunto	Precisão Média Total	Revocação Média Total
Área	95,00	57,89	100,00	62,50	97,50	60,20
Densidade	68,60	47,14	100,00	62,50	84,30	54,82
Forma	95,00	57,89	100,00	62,50	97,50	60,20
Entropia	64,87	43,08	100,00	62,50	82,44	52,79
SMA	64,09	41,67	82,41	58,93	73,25	50,30
Contraste	61,43	48,00	74,43	60,00	67,93	54,00
Área + Densidade	86,96	57,89	100,00	62,50	93,48	60,20
Densidade + Forma	95,00	57,89	100,00	62,50	97,50	60,20
Área + Forma	95,00	57,89	100,00	62,50	97,50	60,20
Entropia + SMA	64,87	43,08	100,00	62,50	82,44	52,79
SMA + Contraste	65,38	50,00	74,43	60,00	69,90	55,00
Entropia + Contraste	70,53	48,00	82,47	62,50	76,50	55,25
Todas as Características	70,53	48,00	82,75	62,50	76,64	55,25

A partir dos dados apresentados na Tabela 3, conclui-se que as características individuais que apresentaram um melhor desempenho para os testes criados foram a área e a forma, e as características agrupadas com melhor desempenho foram Área + Forma e Densidade + Forma, onde tiveram uma precisão média de 95% em 57,89% de revocação média para o segundo conjunto e precisão média 100% em 62,5% de revocação média para o terceiro conjunto.

Também é possível analisar que os piores desempenhos entre as características individuais foram apresentados pelo contraste, com precisão média de 61,43% em 48% de revocação média para o segundo conjunto e precisão média de 74,43% em 60% de revocação média para o terceiro conjunto, seguido do segundo momento angular com precisão média de 64,09% em 41,67% de revocação média para o segundo conjunto e precisão média de 82,41% em 58,93% de revocação média para o terceiro conjunto. Os piores desempenhos das características agrupadas foram apresentados pela Entropia + SMA no segundo conjunto, com precisão média de 64,87% em 43,08% de revocação média e pelo SMA + Contraste no terceiro conjunto, com precisão média de 74,43% em 60% de revocação média.

3.6 Comparação de tempo para a recuperação em cada conjunto

Como meio de se avaliar o sistema, foi medido o tempo para a recuperação das imagens em cada uma das características implementadas e criados dois tipos de gráficos. O primeiro mostra o tempo para a recuperação de cada característica em cada conjunto, possibilitando a análise do tempo das características nos três conjuntos, e o segundo tipo de gráfico foi criado para cada um dos conjuntos mostrando o percentual de tempo de recuperação de cada característica.

Os testes apresentados foram feitos em um computador *Pentium 4*, 3.0 *GigaHertz*, com *512 MegaBytes* de memória RAM e 80 *GigaBytes* de disco rígido com sistema operacional Windows XP. Para cada tipo de gráfico explicado acima, foram executados os testes duas vezes, observando-se a mesma seqüência de procedimentos. As consultas realizadas buscavam as cinco imagens mais próximas a uma mesma imagem de referência.

A Figura 49 apresenta os gráficos criados para a demonstração do tempo para a recuperação de cada um dos conjuntos criados, discriminando o tempo de cada característica. Pode-se observar por estes gráficos que o tempo de recuperação não seguiu paralelamente ao longo das características. Em algumas características como a densidade e a forma, houve um menor tempo para a recuperação das imagens nos conjuntos com menor quantidade de imagens do que no conjunto com maior quantidade, devido ao fato de possuírem um processamento mais complexo do que as demais.

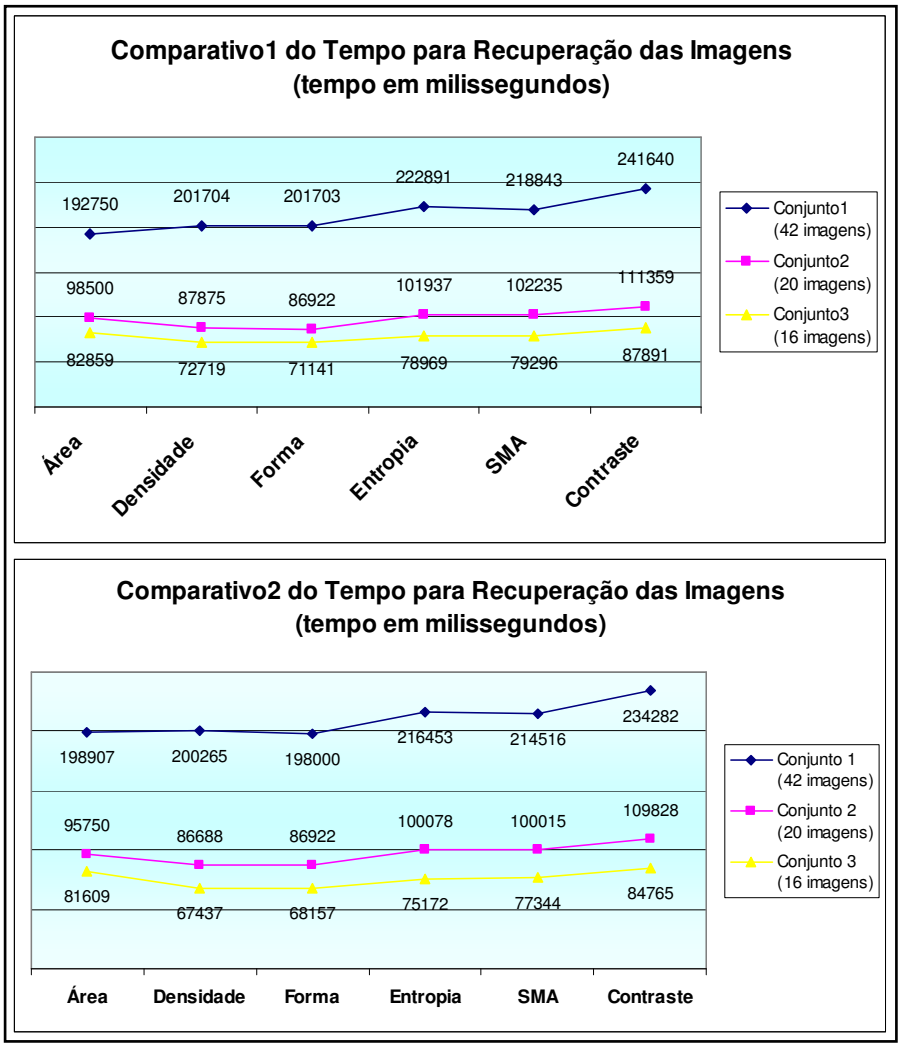


Figura 49 – Gráficos comparativos dos tempos para a recuperação das imagens

Para uma melhor análise do tempo gasto para a recuperação de cada característica em cada um dos conjuntos de imagens, foram criados os gráficos da Figura 50 que demonstram a porcentagem de tempo de cada característica para cada um dos conjuntos, em cada uma das vezes que o tempo foi marcado, ficando do lado esquerdo os gráficos da primeira marcação e do lado direito da segunda.

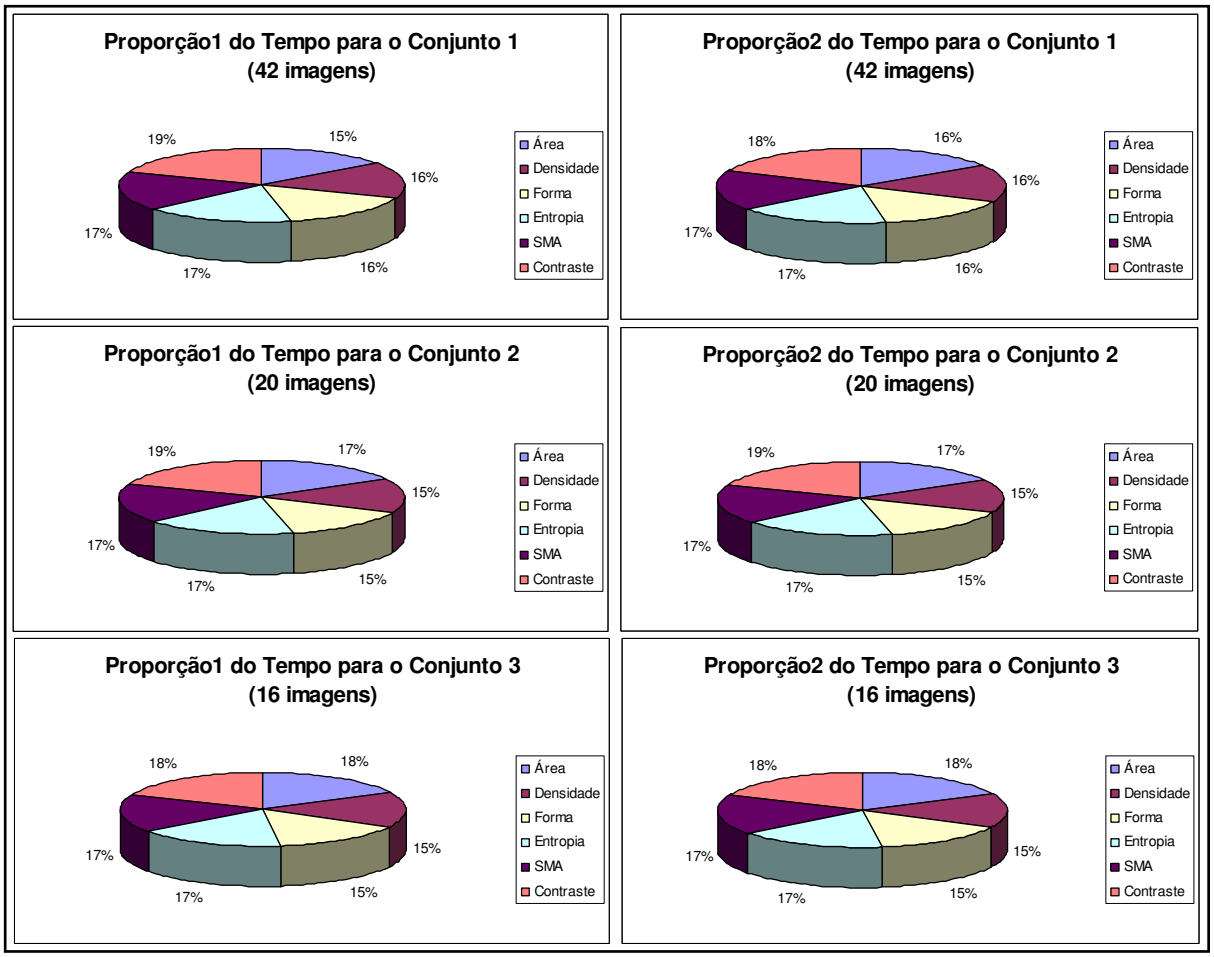


Figura 50 – Gráficos que demonstram a proporção do tempo gasto para cada característica em cada conjunto de teste

Por meio dos gráficos da Figura 50, pode-se perceber que a área foi calculada mais rapidamente no primeiro conjunto, onde havia mais imagens e mais lentamente no terceiro conjunto, onde havia menos imagens, concluindo, portanto, que a mesma apresenta uma performance mais rápida em conjuntos de dados maiores. Tanto a densidade como a forma se mostraram mais rápidas para o segundo e terceiro conjunto do que para o primeiro, levando à conclusão de que é mais rápida em conjuntos de dados menores. As demais características não apresentaram grandes diferenças no tempo de cálculo entre os conjuntos de imagens.

CONCLUSÕES

O objetivo deste trabalho foi a implementação de um sistema de recuperação de imagens baseada em conteúdo para a aplicação em um banco de imagens mamográficas, utilizado originalmente para apoiar o desenvolvimento e os testes de sistemas CAD. Para atingir este objetivo, primeiramente foi realizada uma revisão bibliográfica a fim de encontrar na literatura sistemas CBIR já implementados de aplicação na área de imagens médicas, bem como as características e os critérios de similaridade utilizados por estes sistemas. Após a fase de levantamento da bibliografia existente sobre o assunto, foi implementado um sistema para atingir ao objetivo almejado.

Primeiramente foi desenvolvido um algoritmo para a eliminação do fundo das imagens mamográficas, uma vez que o sistema deve considerar sempre a imagem sem o fundo para a realização dos cálculos necessários. Logo após, foi proposto um esquema genérico para CBIR, que também implementa uma função de similaridade, a partir do qual foram criados os extratores de características e a recuperação das imagens. Também foi elaborado um banco de dados para armazenar os dados necessários.

Durante a construção deste sistema foram analisados os resultados parciais que foram apresentados em SANTOS *et al.* (2006a) e SANTOS *et al.* (2006b).

Os extratores implementados mostraram-se satisfatórios para os testes realizando, apresentando um bom desempenho na recuperação. O único detalhe é que eles são um pouco lentos para calcular na primeira vez, uma vez que são calculados uma única vez, na primeira consulta, nas demais os valores são apenas recuperados do banco. As imagens contribuem para a lentidão dos cálculos, pois seus cálculos são muito grandes.

Os extratores das características área e densidade foram os que apresentaram melhor desempenho e os extratores das características de contraste e segundo momento angular foram os que apresentaram pior desempenho para os testes realizados.

Além do aprendizado proporcionado, podem ser citadas como contribuições deste trabalho: a análise profunda do funcionamento de um sistema CBIR, bem como a explicação de cada uma das etapas necessárias para a construção de tal sistema; o desenvolvimento de um esquema genérico para CBIR que pode ser usado para implementar extratores e recuperação de imagens para objetos de qualquer natureza e, obviamente, a análise das características implementadas especificamente para recuperar imagens mamográficas.

Apesar de implementar apenas uma função de similaridade, considerada a mais simples (distância Euclidiana), este trabalho apresenta várias funções possíveis, assim como várias características que podem ser implementadas com suas respectivas fontes bibliográficas. Desta forma, a continuidade natural deste trabalho é a implementação e testes de novas características e novas funções de similaridade.

Trabalhos futuros

Conforme apresentado no capítulo anterior, na fase de teste do sistema demonstrado foram detectadas algumas falhas a serem corrigidas futuramente, como o algoritmo de segmentação das imagens, que apresentou erro na tentativa de descobrir o lado da mama.

Outro ponto que se pretende analisar futuramente é a obtenção das características de textura fornecendo parâmetros diferenciados para o cálculo da matriz de co-ocorrência, uma vez que no sistema atual, foram testadas as características com parâmetros fixos. O sistema,

no entanto, já está preparado para tais testes, que não foram executados por não haver tempo hábil para esta ação.

Também deverão ser implementadas novas características a fim de tentar encontrar aquelas com melhor desempenho para o contexto apresentado (imagens mamográficas).

Outro ponto que se pretende explorar é uma análise mais aprofundada das curvas de precisão e revocação a fim de que essas possam, efetivamente, auxiliarem na avaliação dos resultados obtidos.

Além disso, como citado, o esquema genérico implementado, assim como os extratores desenvolvidos, pode ser aplicado em imagens de natureza diversas a fim de avaliar a sua utilidade para construção de sistemas de CBIR.

REFERÊNCIAS

ANDRÉ, T. C. S. S.; MARQUES, P. M. A.; RODRIGUES, J. A. H.; RANGAYYAN, R. M. **Sistema de Recuperação de Imagens Baseada em Conteúdo Usando Mapas de Kohonen e Técnicas de Correlação Cruzada**. In: IX CBIS, Congresso Brasileiro de Informática em Saúde, Ribeirão Preto, novembro de 2004.

APACHE Derby, Apache Derby. Disponível em: http://db.apache.org/derby/. Acesso em novembro, 2006.

ARAUJO, M. R. B.; TRAINA JR., C; TRAINA A.; MARQUES, P. M. A. **Recuperação de Exames em Sistemas de Informação Hospitalar com Suporte a Busca de Imagens Baseada em Conteúdo**. In: VIII Congresso Brasileiro de Informática em Saúde - CBIS'2002, Natal - RN, de 29 de setembro a 02 de outubro de 2002.

BAEZA-YATES, R.; RIBEIRO-NETO, B. **Modern information retrieval.** New York, ed. Addison-Wesley. 513p. 1999.

BENATTI, R. H.; SCHIABEL, H.; NUNES, F. L. S. **A mammographic images database for computer-aided diagnosis schemes**. In: World Congress on Medical Physics and Biomedical Engineering, 2003, Sidney. Proceedings of World Congress on Medical Physics and Biomedical Engineering, 2003. v. 1.

BUENO, J. M. **Suporte à Recuperação de Imagens Médicas Baseada em Conteúdo através de Histogramas Métricos**. 2001. 148 f. Dissertação (Doutorado em Ciência da Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e Computação (ICMC), Universidade de São Paulo (USP), São Carlos, 2001.

BUENO, J. M.; TRAINA, A. J. M.; TRAINA JR, C.; MARQUES, P. M. A. **cbPACS: PACS com Suporte à Recuperação de Imagens Médicas Baseada em Conteúdo**. In: VIII Congresso Brasileiro de Informática em Saúde – CBIS, Natal, RN-Brazil, 2002.

CORBOY, A.; TSANG, W.; RAICU, D.; FURST, J. **Texture-Based Image Retrieval for Computerized Tomography Databases.** In: The 18th IEEE International Symposium on Computer-Based Medical Systems (CBMS'05), June, 2005.

DAY-CARE, Instituto de Informações e Suporte em Oncologia. Disponível em http://www.daycare.com.br/perguntas/perg49.asp. Acesso em março, 2006.

DENGLER, J.; BEHRENS, S.; DESAGA, J. F. **Segmentation of microcalcifications in mammograms**. In: IEEE Transactions on Medical Imaging, v.12, n.4, p.634-642, 1993.

EL-NAQA, I.; YANG Y.; GALATSANOS, N. P; NISHIKAWA, R. M.; WERNICK, M. N. **A similarity learning approach to content based image retrieval: application to digital mammography**. In: IEEE Trans. on Medical Imaging, vol. 23, pp.1233-1244, 2004.

EL NAQA, I.; YANG, Y.; GALATSANOS N. P.; WERNICK, M. N. **Content-Based Image Retrieval for digital Mammography.** In: IEEE International Conference on Image Processing, Rochester, NY, September 2002.

FERRERO, C. A.; LEE, H. D.; CHUNG, W. F.; COY, C. S. R.; FAGUNDES, J. J.; GÓES, J. R. N. **Seleção de Características Baseadas em Textura para a Identificação de Anormalidades em Imagens de Colonoscopia**. In: X Congresso Brasileiro de Informática em Saúde – CBIS'2006, Florianópolis, SC-Brazil, 2006.

GATO, H. E. R.; NUNES, F. L. S., SCHIABEL, H. **Uma proposta de Recuperação de Imagens Mamográficas Baseada em Conteúdo**. In: IX Congresso Brasileiro de Informática em Saúde, 2004, Ribeirão Preto. Anais do Congresso Brasileiro de Informática em Saúde, 2004. v. 1.

GIGER, M. L. **Computer-aided diagnosis of breast lesions in medical images**. In: Computing in Science & Engineering, v.2, n.5, p. 39-45, 2000.

GONZALEZ, R.C.; WOODS, R.E. **Processamento de Imagens Digitais**. Ed Edgard Blücher LTDA. São Paulo, 2000.

GUDIVADA V. N.; RAGHAVAN V. V. **Content-Based Image Retrieval Systems**, IEEE Computer-Special Issue, Vol. 28, pp. 18--62, September 1995.

HIRATA, M.K. **Centro de Massa**. Disponível em: http://www.ifi.unicamp.br/~lunazzi/F530_F590_F690_F809_F895/F809/F809_sem1_2005/MiguelK-Rigitano_RF.pdf. Acesso em maio, 2006.

INCA, Instituto Nacional de Câncer. **Estimativa da Incidência de Câncer para 2006 no Brasil e nas cinco Regiões.** Disponível em: http://www.inca.gov.br/conteudo_view.asp?id=1793. Acesso em dezembro, 2005.

JI, X.; KATO, Z.; HUANG, Z. **Non-Photorealistic Rendering and Content-Based Image Retrieval**. In Proceedings of Pacific Conference on Computer Graphics and Applications, Canmore, Canada, pages 153--162, IEEE, October 2003.

KINOSHITA, S. K.; PEREIRA JR, R. R.; HONDA, M. O.; RODRIGUES, J. A. H., MARQUERS, P.M.A. **Recuperação Baseada em Conteúdo de Imagens Mamográficas: Atributos Visuais de Forma, Espectrais no domínio de Radon e Granulometria**. In: IX Congresso Brasileiro de Informática em Saúde, 2004, Ribeirão Preto. CBIS2004, 2004.

LEHMANN, T. M.; GÜLG, M. O.; THIES, C.; FISCHER, B.; KEYSERS, D.; KOHNEN, M.; SCHUBERT H.; WEIN, B. B. **Content-based image retrieval in medical applications for picture archiving and communication systems.** Proceedings of IS&T/SPIE Medical Imaging 2003: PACS and Integrated Medical Systems, Vol. SPIE 5033, 5033, February 2003.

LEHMANN, T. M.; GÜLG, M. O.; THIES, C.; PLODOWSKI, B.; KEYSERS, D.; OTT, B.; SCHUBERT, H. IRMA – **Content-Based Image Retrieval in Medical Applications**. In: Proceedings of the 14th World Congress on Medical Informatics (MedInfo 2004). IOS Press, Amsterdam, 2004, pp. 842-848

MARQUES, J.; TRAINA, A. J. M. **Realimentação de Relevância: Integração do Conhecimento do especialsta com a Recuperação de Imagens por Conteúdo**. In: VI Workshop de Informática Médica - WIM'2006 (junto ao V Simpósio Brasileiro de Qualidade de Software - SBQS), 2006, Vitória. Anais do VI Workshop de Informática Médica WIM'2006. Vitória. 2006. v.1. p. 83-93.

MARQUES, P. M. A.; HONDA, M. H.; RODRIGUES, J. A. H.; SANTOS, R. R.; TRAINA, A.J.M.; TRAINA JR, C.; BUENO, J.M. **Recuperação de Imagem Baseada em Conteúdo: Uso de Atributos de Textura para Caracterização de Microcalcificações Mamográficas**. Revista Brasileira de Radiologia, vol. 35, pp. 93-98, 2002.

MOSHFEGHI, M.; SAIZ, C.; YU, H.; **Content-based retrieval of medical images with relative entropy.** Proc. of SPIE Medical Imaging 2004, Picture Archiving and Communication Systems (PACS) and Imaging Informatics, vol. 5371, 2004, pp. 31-42

NUNES, F. L. S.; SCHIABEL, H.; GOES, C. E. **Using free technology to store and retrieve mammographic images by Internet**. In: 5th International Workshop on Image Analysis, 2004, Lisboa, Proceedings of 5th International Workshop on Image Analysis, 2004a.

NUNES, F. L. S.; SCHIABEL, H.; OLIVEIRA JR, J. A.; GOES, C. E., BENATTI, R. H. **Disponibilização de imagens mamográficas via Internet: uma forma de contribuir para a integração regional de pesquisas em diagnóstico auxiliado por computador**. In: Anais do XXIV Congresso da Sociedade Brasileira de Computação – Seminário Integrado de Hardware e Software, Salvador, julho 2004b.

NUNES, F. L. S. **Investigações em processamento de imagens mamográficas para auxílio ao diagnóstico de mamas densas**. Tese Doutorado. Instituto de Física de São Carlos, 2001.

SANTOS, R. D. C. **Java Advanced Imaging API: A Tutorial**. Revista de Informática Teórica e Aplicada, Rio Grande do Sul, v. 11, p. 93-123, 2004.

SANTOS, A. P. O.; NUNES, F. L. S. **Recuperação de Imagens Mamográficas Baseada em Conteúdo**. In: XIV Congresso de Iniciação Científica da UFSCar, São Carlos, outubro 2006a.

SANTOS, A. P. O.; NUNES, F. L. S.; DELAMARO, M. E. **Recuperação de Imagens Mamográficas Baseada em Conteúdo**. In: 14º Simpósio Internacional de Iniciação Científica da USP, São Paulo, novembro 2006b.

SUCKLING J.; PARKER J.; DANCE D.; ASTLEY S.; HUTT I.; BOGGIS C.; RICKETTS I.; STAMATAKIS E.; CERNEAZ N.; KOK S.; TAYLOR P.; BETAL D.; SAVAGE J. **The mammographic images analysis society digital mammogram database.** Exerpta Medica. International Congress Series 1069: pp. 375-378, 1994.

SUN Microsystems, **JAI (Java Advanced Imaging) Application Programming Interface document home page**. Disponível em: http://java.sun.com/products/java-media/jai/forDevelopers/jai-apidocs/index.html. Acesso em novembro, 2006a.

SUN Microsystems, **JAVA document home page.** Disponível em: http://java.sun.com/j2se/1.5.0/docs/api/. Acesso em novembro, 2006b.

TRAINA, A. J. M.; TRAINA JR., C; BUENO, J. M.; MARQUES, P. M. A. **The Metric Histogram: A New and Efficient Approach for Content-based Image Retrieval.**

Proceedings of the Sixth IFPI Working Conference on Visual Database Systems (VDB6), Brisbane, Austrália, Kluwer Academic Press, p.297-311, 2002.

TRAINA JR, C.; TRAINA, A. J. M.; SEEGER, C.; FALOUTSOS, C. **Slim-Trees: High Performance Metric Trees Minimizing Overlap Between Nodes.** In Intl. Conf. On Extending Database Technology. Konstanz, Germany, 2000.

YAMAMOTO, H.; IWASA, H.; YOKOYA, N. TAKEMURA, H. **Content-based Similarity Retrieval of Images Based on Spacial Coor Distributions**. In: 10º Intl. Conference on Image Analysis and Processing, 1999.

APÊNDICE A – CÓDIGO FONTE DA CLASSE TIRAFUNDO.JAVA

```
/**
 * Esta classe recebe como parametro uma String que representa o caminho
 * para a imagem que terá o seu fundo retirado e armazena a nova imagem
 * sem o fundo em um novo arquivo chamado sfundo.tiff no diretório
 * corrente.
 */

package segmentacao;

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.awt.image.renderable.ParameterBlock;
import java.awt.geom.Point2D;
import java.lang.*;
import java.lang.Math.*;
import javax.media.jai.*;
import javax.swing.*;

public class TiraFundo extends JFrame
{
    /**
     * Contrutor da classe que abre a imagem passada por parâmtro e
     * elimina o fundo da imagem, salvando-a sem fundo no mesmo
     * diretório com o nome de sfundo.tiff
     */
    public TiraFundo(String mamo) {
        PlanarImage mamograma = JAI.create("fileload",mamo);
        int width = mamograma.getWidth();
        int height = mamograma.getHeight();
        SampleModel sm = mamograma.getSampleModel();
        int nbands = sm.getNumBands();
        Raster inputRaster = mamograma.getData();
        WritableRaster outputRaster =
                    inputRaster.createCompatibleWritableRaster();
        int[] pixels = new int[nbands*width*height];
        ColorModel cm = mamograma.getColorModel();
        TiledImage tiledImage = new
                    TiledImage(0,0,width,height,0,0,sm,cm);
        inputRaster.getPixels(0,0,width,height,pixels);
        int offset, flagFim = 0, thr = 62300;

        long somaPri=0, valorPri=0, somaSeg=0, valorSeg=0,
             somaPen=0, valorPen=0, somaUlt=0, valorUlt=0,
            PixelInicial = 0, PixelFinal = 0, xCentro, yCentro;
        // Guarda o valor de um terço da imagem.
        int h1_3 = (int) height/3;
         long hist[] = new long[65536];

        /**
         * Trecho que faz o histograma e que calcula a média dos
         * valores dos pixels das colunas iniciais (10ª e 11ª) e das
         * finais correspondentes da imagem.
         */
        for (int h=0; h<height; h++)
            for (int w=0; w<width; w++) {
```

```
      offset = h*width*nbands+w*nbands;
      for (int band=0; band<nbands; band++) {
         if (w == 10) {
            somaPri = somaPri + 1;
            valorPri = valorPri + pixels[offset+band];
         }
         else if (w == 11) {
            somaSeg = somaSeg + 1;
            valorSeg = valorSeg + pixels[offset+band];
         }
         else if (w == width-12) {
            somaPen = somaPen + 1;
            valorPen = valorPen + pixels[offset+band];
         }
         else if (w == width-11) {
            somaUlt = somaUlt + 1;
            valorUlt = valorUlt + pixels[offset+band];
         };
         // histograma
         hist[pixels[offset+band]]++;
      }
   }

/**
 * Trecho que encontra o valor do threshold (maior valor do
 * histograma)
 */
for (int i = 0; i < 65536; i++)
   if (hist[i] > hist[thr]) {
      thr = i;
   }

// Trecho que faz o thresholding (limiarização) da imagem
for (int h=0; h<height; h++)
   for (int w=0; w<width; w++) {
      offset = h*width*nbands+w*nbands;
      for (int band=0; band<nbands; band++)
         if ( pixels[offset+band] > thr )
            pixels[offset+band] = 65535;
         else
            pixels[offset+band] = 0;
   }

/**
 * Calcula a diferença das médias de valor de cinza das
 * primeiras e das últimas colunas para determinar o lado da
 * mama.
 */
long dif1 = Math.max((long)valorPri/somaPri, (long)valorSeg/somaSeg)
      - Math.min((long)valorPri/somaPri, (long)valorSeg/somaSeg);
long dif2 = Math.max((long)valorPen/somaPen, (long)valorUlt/somaUlt)
      - Math.min((long)valorPen/somaPen, (long)valorUlt/somaUlt);

/**
 * Se as últimas colunas tem uma maior diferença, então a mama
 * esta do lado direito e deve-se encontrar o ponto central a
 * partir da direita.
 */
if (dif2 > dif1) {
   /**
    * percorre a última coluna procurando o centro da mama,
```

```
     * caso não encontre, vai para a penúltima e assim por
     * diante, até que seja encontrado pelo menos um pixel
     * preto.
     */
    int w = width;
    do {
       w = w -1;
       for (int h=0; h<height; h++) {
          offset = h*width*nbands+w*nbands;
          for (int band=0; band<nbands; band++)
             if (pixels[offset+band] == 0) {
                flagFim = 1;
                if ( h < h1_3 ) {
                   flagFim = 0;
                   PixelInicial = h;
                }
                else
                   PixelFinal = h;
             }
       }
    } while (flagFim == 0);
    xCentro = w;
    yCentro = ( (PixelFinal - PixelInicial) /2 ) + PixelInicial;
}
/*
 * Caso as primeiras colunas tenham uma maior diferença, então
 * a mama está do lado esquerdo e deve-se procurar o ponto
 * central a partir da esquerda.
 */
else {
    //percorre a primeira coluna procurando o centro da mama
    int w = 0;
    do {
       w = w + 1;
       for (int h=0; h<height; h++) {
          offset = h*width*nbands+w*nbands;
          for (int band=0; band<nbands; band++)
             if (pixels[offset+band] == 0) {
                flagFim = 1;
                if ( h < h1_3 ) {
                   flagFim = 0;
                   PixelInicial = h;
                }
                else
                   PixelFinal = h;
             }
       }
    } while (flagFim == 0);
    xCentro = w;
    yCentro = ( (PixelFinal - PixelInicial) /2 ) + PixelInicial;
}

/*
 * Lançamento de raios a partir do ponto central definido acima
 * a fim de encontrar a borda da mama.
 */
long vetorX[] = new long[362];
long vetorY[] = new long[362];
int x, y;
double Angulo, AnguloRadiano, Xaux, Yaux;
int posVetor = 0;
```

```
long menorY = height, maiorY = 0;
// Se a mama esta do lado direito
if (dif2 > dif1) {
   // Vai indo de 0.5 em 0.5 graus, de 90° a 270°
   for ( Angulo = 90.0; Angulo < 271; Angulo += 0.5 ) {
      AnguloRadiano = Math.toRadians(Angulo);
      flagFim = 0; // ainda não chegou ao fim
      x = (int)xCentro;
      y = (int)yCentro;
      Xaux = (double)xCentro;
      Yaux = (double)yCentro;
      while ((flagFim == 0) && ( x < width ) && ( x > 0 ) &&
             ( y > PixelInicial) && ( y < PixelFinal )) {
         Xaux = Xaux + Math.cos(AnguloRadiano);
         x = (int)Xaux;
         Yaux = Yaux - Math.sin(AnguloRadiano);
         y = (int)Yaux;
         offset = y*width*nbands+x*nbands;
         for (int band=0; band<nbands; band++)
            if (pixels[offset+band] == 0)
               flagFim = 1;
      }
      vetorX[posVetor] = x;
      vetorY[posVetor] = y;
      if (y > maiorY)
         maiorY = y;
      if ( y < menorY)
         menorY = y;

      posVetor++;
      Xaux = x;
      Yaux = y;
   }
}
// Se a mama está do lado esquerdo
else {
   // Vai indo de 0.5 em 0.5 graus, de 90° a 270°
   for ( Angulo = 90.0; Angulo > -89.0; Angulo -= 0.5 ) {
      if ( Angulo < 0 )
         AnguloRadiano = Math.toRadians(360+Angulo);
      else
         AnguloRadiano = Math.toRadians(Angulo);
      flagFim = 0; // ainda não chegou ao fim
      x = (int)xCentro;
      y = (int)yCentro;
      Xaux = (double)xCentro;
      Yaux = (double)yCentro;
      while ((flagFim == 0) && ( x < width ) && ( x > 0 ) &&
             ( y > PixelInicial) && ( y < PixelFinal )) {
         Xaux = Xaux + Math.cos(AnguloRadiano);
         x = (int)Xaux;
         Yaux = Yaux - Math.sin(AnguloRadiano);
         y = (int)Yaux;
         offset = y*width*nbands+x*nbands;
         for (int band=0; band<nbands; band++)
            if (pixels[offset+band] == 0)
               flagFim = 1;
      }
      vetorX[posVetor] = x;
      vetorY[posVetor] = y;
      if (y > maiorY)
```

```
            maiorY = y;
         if ( y < menorY)
            menorY = y;
         posVetor++;
         Xaux = x;
         Yaux = y;
      }
   }

   int flagSobre = 0;
   long posMuda = height;
   int h;


   /**
    * Encontrando e traçando linhas dos pontos encontrados
    * e dos intemediários para delimitar a borda da mama.
    */
   for (int j=0; j<posVetor-1; j++){
      int vetYj = (int)vetorY[j];
      int vetYj1 = (int)vetorY[j+1];
      /**
       * tirando a média dos dois pontos para ver se estão a
       * uma distância maior que 5% do tamanho da imagem, pois
       * se estiverem, é um sinal que o ponto encontrado pode
       * não pertencer a mama, uma vez que está muito distante
       * do anterior (distante da borda)
       */
      int dist = vetYj > vetYj1 ? vetYj - vetYj1 : vetYj1 - vetYj;
      if (dist > height*0.05) {
         vetorY[j+1] = vetYj > vetYj1 ? vetorY[j+1]+(dist/2) :
                                    vetorY[j+1]-(dist/2);
         if (vetYj1 == maiorY)
            maiorY = (int)vetorY[j+1];
         else if(vetYj1 == menorY)
            menorY = (int)vetorY[j+1];
         vetYj1 = (int)vetorY[j+1];
      }
      /**
       * Este trecho analisa se o ponto encontrado está
       * sobrepondo ou não uma outra parte já encontrada, e se
       * estiver é feito o tratamento necessário
       * para não se perder nenhuma parte da imagem.
       */
      if ((flagSobre == 0) && (vetYj > vetYj1)) {
         flagSobre = 1;
         posMuda = vetYj;
      }
      else if ((flagSobre == 1) && (vetYj < vetYj1))
         flagSobre = 2;
      else if ((flagSobre == 2) && (vetYj > vetYj1))
         flagSobre = 1;
      else if ((flagSobre == 2) && (vetYj > posMuda))
         flagSobre = 0;
      // Tratamento se os pontos entiverem em ordem crescente
      for (int auxY = vetYj; auxY < vetYj1+1; auxY++) {
         int auxX;
         if (vetorX[j] == vetorX[j+1])
            auxX = (int)vetorX[j];
         else {
            double m = (double)(vetorY[j+1] - vetorY[j]) /
```

```
                            (vetorX[j+1] - vetorX[j]);
            double b = vetorY[j] - m*vetorX[j];
            auxX = ( auxY - (int)b );
            auxX /= m;
        }
        h = auxY;
        for (int w=0; w<width; w++) {
            offset = h*width*nbands+w*nbands;
            for (int band=0; band<nbands; band++) {
                // se mama está do lado direito
                if (dif2 > dif1)
                    if (flagSobre == 0)
                        if (w < auxX)
                            pixels[offset+band] = 0;
                        else
                            pixels[offset+band] = 65535;
                    else {// se flag == 2
                        if (w >= auxX)
                            pixels[offset+band] = 65535;
                        if ((auxY > posMuda) && (w < auxX))
                            pixels[offset+band] = 0;
                    }
                else // mama está do lado esquerdo
                    if (flagSobre == 0)
                        if (w > auxX)
                            pixels[offset+band] = 0;
                        else
                            pixels[offset+band] = 65535;
                    else {// se flag == 2
                        if (w <= auxX)
                            pixels[offset+band] = 65535;
                        if ((auxY > posMuda) && (w > auxX))
                            pixels[offset+band] = 0;
                    }
            }
        }
    }
    // Tratamento se os pontos entiverem em ordem decrescente
    for (int auxY = vetYj; auxY > vetYj1-1; auxY--) {
        int auxX;
        if (vetorX[j] == vetorX[j+1])
            auxX = (int)vetorX[j];
        else {
            double m = (double)(vetorY[j+1] - vetorY[j]) /
                            (vetorX[j+1] - vetorX[j]);
            double b = vetorY[j] - m*vetorX[j];
            auxX = ( auxY - (int)b );
            auxX /= m;
        }
        h = auxY;
        for (int w=0; w<width; w++) {
            offset = h*width*nbands+w*nbands;
            for (int band=0; band<nbands; band++)
                // se mama está do lado direito
                if (dif2 > dif1) {
                    if (w >= auxX)
                        pixels[offset+band] = 0;
                }
                else
                    if (w <= auxX)
                        pixels[offset+band] = 0;
```

```
        }
    }
}

/*
 * Todas as linhas que estiverem abaixo do menorY encontrado e
 * que estiveram acima do maiorY encontrado ficam inteiras
 * pretas.
 */
for (h=0; h<height; h++)
    for (int w=0; w<width; w++) {
        offset = h*width*nbands+w*nbands;
        for (int band=0; band<nbands; band++)
            if ((h < menorY) || (h > maiorY))
                pixels[offset+band] = 0;
    }

/**
 * Criação das variáveis necessárias para duas novas imagens,
 * um que será a imagem original, e outra a imagem resultante
 * (sem o fundo)
 */
SampleModel sm1 = mamograma.getSampleModel();
int nbands1 = sm1.getNumBands();
Raster inputRaster1 = mamograma.getData();
WritableRaster outputRaster1 =
    inputRaster1.createCompatibleWritableRaster();
int[] pixels1 = new int[nbands1*width*height];
ColorModel cm1 = mamograma.getColorModel();
TiledImage original = new TiledImage(0,0,width,height,0,0,sm,cm);
inputRaster1.getPixels(0,0,width,height,pixels1);

SampleModel sm2 = mamograma.getSampleModel();
int nbands2 = sm2.getNumBands();
Raster inputRaster2 = mamograma.getData();
WritableRaster outputRaster2 =
    inputRaster2.createCompatibleWritableRaster();
int[] pixels2 = new int[nbands2*width*height];
int offset2;

/**
 * Neste trecho é avaliado se o pixel na imagem que foi
 * limiarizada e que está com o fundo preto e a mama branca é
 * preto, se for, a imagem resultante recebe 0 (preto), caso
 * contrário, a mesma recebe o valor da imagem original.
 */
for (h=0; h<height; h++)
    for (int w=0; w<width; w++) {
        offset = h*width*nbands+w*nbands;
        offset2 = h*width*nbands2+w*nbands2;
        for (int band=0; band<nbands; band++)
            if (pixels[offset+band] == 0)
                pixels2[offset2+band] = 0;
            else
                pixels2[offset2+band] = pixels1[offset+band];
    }

outputRaster.setPixels(0,0,width,height,pixels2);
tiledImage.setData(outputRaster);

// Grava a imagem resultante com o nome sfundo.tiff
```

```
        JAI.create("filestore",tiledImage,"sfundo.tiff","TIFF");
    }
}
```

APÊNDICE B – CÓDIGO FONTE DO EXTRATOR EXTRAIAREA.JAVA

```
/**
 * Está é uma classe que implementa um extrator de área.
 * Esta classe extende a classe AbstractExtrator, que serve como
 * base para a criação dos extratores de característica deste sistema.
 */
package extractorimpl;

import java.awt.Rectangle;
import java.awt.image.Raster;

import javax.media.jai.PlanarImage;

import extractor.AbstractExtractor;
import extractor.ParameterBlock;

public class ExtraiArea extends AbstractExtractor {

    // variação permitida para que pixels sejam considerados iguais
    private int thr;

    /** Cria um objeto associado a uma imagem.
     *
     * @param p - imagem a ser utilizada.
     */
    public ExtraiArea(PlanarImage p)
    {
        super(p);
    }

    /**
     * Compara a área da imagem atual e daquela associada ao extrator
     * passado por parâmetro.
     * Se a diferença entre as duas áreas for menor ou igual ao threshold
     * definido, elas são considerados iguais.
     * O resultado retornado é o número 1 se as áreas são iguais,
     * considerando-se o threshold, ou 0 no caso das áreas serem
     * consideradas diferentes.
     */
    public double compare(AbstractExtractor x) {
        if ( ! (x instanceof ExtraiArea) )
        {
            throw new IllegalArgumentException
                                    ("ExtraiArea extractor required.");
        }
        ExtraiArea area = (ExtraiArea) x;
        double area1 = computeValue();
        double area2 = area.computeValue();
        double dif = area1 > area2 ? area1 - area2 : area2 - area1;

        if(dif > thr) /* se a diferença das áreas é maior que o limite thr*/
            return 0;  /* as área são diferentes */
        else         /* se a diferença das área é menor ou igual ao limite */
            return 1;  /* as áreas são iguais */
```

```
    }

    /**
     * Calcula o valor da área para a imagem, que nada mais é do
     * que a soma dos pixels da imagem, no caso, apenas os pixels
     * pertencetes a área segmentada (aquele pixels que não tem o
     * valor zero). Para se obter um valor normalizado, dividiu-se
     * a área pelo tamanho da imagem total (considerando o
     * fundo da mesma).
     */
    public double computeValue() {
        int[] im1 = getData();
        int area = 0;
        int total = 0;
        for( int i = 0; i < im1.length; i++) {
            if( im1[i] != 0 )
                area++;
            total++;
        }
        return (double) area / total;
    }

    /**
     * Obtem, na forma de um array, os pixels da região definida para a
     * comparação.
     *
     * @return - vetor de inteiro com os pixels a serem comparados,
     * que representam a região da imagem definida.
     */
    public int[] getData()
    {
        PlanarImage im = getImage();
        Raster r = im.getData( new
                Rectangle( 0,0,im.getWidth(),im.getHeight() ) );
        return r.getPixels(0,0,r.getWidth(), r.getHeight(), (int[]) null);
    }

    /**
     * Define os parâmetros a serem utilizados por esse objeto.
     *
     */
    public void setParameters(ParameterBlock pb) {
        thr = pb.getIntParameter(2);
        thr = thr < 0 ? 0 : thr;
    }

}
```

APÊNDICE C – CÓDIGO FONTE DO EXTRATOR EXTRAIDENSIDADE.JAVA

```
/**
 * Está é uma classe que implementa um extrator de densidade.
 * Esta classe extende a classe AbstractExtrator, que serve como
 * base para a criação dos extratores de característica deste sistema.
 */
package extractorimpl;

import java.awt.Rectangle;
import java.awt.image.Raster;

import javax.media.jai.PlanarImage;

import extractor.AbstractExtractor;
import extractor.ParameterBlock;

public class ExtraiDensidade extends AbstractExtractor {

    // variação permitida para que pixels sejam considerados iguais
    private int thr;

    /** Cria um objeto associado a uma imagem.
     *
     * @param p - imagem a ser utilizada.
     */
    public ExtraiDensidade(PlanarImage p)
    {
        super(p);
    }

    /**
     * Compara a densidade da imagem atual e daquela associada ao extrator
     * passado por parâmetro.
     * Se a diferença entre as duas densidades for menor ou igual ao
     * threshold definido, elas são considerados iguais.
     * O resultado retornado é o número 1 se as desnsidades são iguais,
     * considerando-se o threshold, ou 0 no caso das densidades serem
     * consideradas diferentes.
     */
    public double compare(AbstractExtractor x) {
        if ( ! (x instanceof ExtraiDensidade) )
        {
            throw new IllegalArgumentException
                                ("ExtraiDensidade extractor required.");
        }
        ExtraiDensidade dens = (ExtraiDensidade) x;
        double dens1 = computeValue();
        double dens2 = dens.computeValue();
        double dif = dens1 > dens2 ? dens1 - dens2: dens2 - dens1;

        if (dif > thr) // se a diferença das densidades é maior que o limite
            return 0;  // as densidades são diferentes
```

```
        else          // caso contrário
            return 1;  // as densidade são iguais
    }

    /**
     * Calcula o valor da densidade para a imagem, que nada mais é do
     * que a soma dos pixels da imagem, dividido pelo número total de
     * pixels da mesma. Para se obter um valor normalizado, dividiu-se
     * a densidade obtida pelo nível de cinza máximo possível para as
     * imagens utilizadas, no caso, 65536.
     */
    public double computeValue() {
        int[] im = getData();
        long soma = 0, quant = 0;
        for (int i = 0; i < im.length; i++) {
            if (im[i] != 0) {
                quant++;
                soma += im[i];
            }
        }
        return (double) ( soma / quant ) / 65536;
    }

    /** Obtem, na forma de um array, os pixels da região definida para a
     * comparação.
     *
     * @return - vetor de inteiro com os pixels a serem comparados,
     * que representam a região da imagem definida.
     */
    public int[] getData()
    {
        PlanarImage im = getImage();
        Raster r = im.getData( new
                              Rectangle( 0,0,im.getWidth(),im.getHeight() ) );
        return r.getPixels(0,0,r.getWidth(), r.getHeight(), (int[]) null);
    }

    /**
     * Define os parâmetros a serem utilizados por esse objeto.
     *
     */
    public void setParameters(ParameterBlock pb) {
        thr = pb.getIntParameter(2);
        thr = thr < 0 ? 0 : thr;
    }

}
```

APÊNDICE D – CÓDIGO FONTE DO EXTRATOR EXTRAIFORMA.JAVA

```
/**
 * Está é uma classe que implementa um extrator de forma da mama.
 * Esta classe extende a classe AbstractExtrator, que serve como
 * base para a criação dos extratores de característica deste sistema.
 */
package extractorimpl;

import java.awt.Rectangle;
import java.awt.image.*;

import javax.media.jai.*;
import com.sun.media.jai.widget.DisplayJAI;

import extractor.AbstractExtractor;
import extractor.ParameterBlock;

public class ExtraiForma extends AbstractExtractor {

    // variação permitida para que pixels sejam considerados iguais
    private int thr;

    /** Cria um objeto associado a uma imagem.
     *
     * @param p - imagem a ser utilizada.
     */
    public ExtraiForma(PlanarImage p)
    {
        super(p);
    }

    /**
     * Compara a densidade da imagem atual e daquela associada ao extrator
     * passado por parâmetro.
     * Se a diferença entre as duas densidades for menor ou igual ao
     * threshold definido, elas são considerados iguais.
     * O resultado retornado é o número 1 se as desnsidades são iguais,
     * considerando-se o threshold, ou 0 no caso das densidades serem
     * consideradas diferentes.
     */
    public double compare(AbstractExtractor x) {
        if ( ! (x instanceof ExtraiForma) )
        {
            throw new IllegalArgumentException
                                    ("ExtraiForma extractor required.");
        }
        ExtraiForma forma = (ExtraiForma) x;
        double f1 = computeValue(), f2 = forma.computeValue();
        double dif = f1 > f2 ? f1 - f2 : f2 - f1;
        if (dif > (double)thr) /* se a diferença é maior que o limite thr*/
            return 0;  /* as formas são diferentes */
        else          /* caso contrário */
            return 1;  /* as formas são iguais */
    }
```

```
    /**
     * Obtém o valor do grau de elongação da mama dividindo-se a distância
     * entre os dois extremos da mama (distância vertical) pela distância
     * entre a parede toráxica e a borda da mama o centro da mama
     * (distância horizontal).
     *
     * @return - valor double que representa a medida de razão do diâmetro
     * da imagem
     */
    public double computeValue()
    {
        PlanarImage im = getImage();
        int width = im.getWidth();
        int height = im.getHeight();
        SampleModel sm = im.getSampleModel();
        int nbands = sm.getNumBands();
        Raster inputRaster = im.getData();
        WritableRaster outputRaster =
                            inputRaster.createCompatibleWritableRaster();
        int[] pixels = new int[nbands*width*height];
        ColorModel cm = im.getColorModel();
        TiledImage tiledImage = new TiledImage(0,0,width,height,0,0,sm,cm);
        inputRaster.getPixels(0,0,width,height,pixels);
        int offset, y1 = 0, y2 = 0, x1 = 0, x2 = 0;
        int dy = 0, dx = 0, prim = height / 3;
        for (int h=0; h<height; h++) {
            int w=0;
            offset = h*width*nbands+w*nbands;
            for (int band=0; band<nbands; band++)
                if (pixels[offset+band] != 0)
                    dy++;
        }
        if (dy == 0)
            for (int h=0; h<height; h++) {
                int w=width -1;
                offset = h*width*nbands+w*nbands;
                for (int band=0; band<nbands; band++)
                    if (pixels[offset+band] != 0)
                        dy++;
            }
        int h = dy / 2;
        for (int w=0; w<width; w++) {
            offset = h*width*nbands+w*nbands;
            for (int band=0; band<nbands; band++)
                if (pixels[offset+band] != 0)
                    dx++;
        }
        return (double)dy / dx;
    }

    /**
     * Define os parâmetros a serem utilizados por esse objeto.
     *
     */
    public void setParameters(ParameterBlock pb) {
        thr = pb.getIntParameter(2);
        thr = thr < 0 ? 0 : thr;
    }

}
```

APÊNDICE E – CÓDIGO FONTE DA CLASSE MATRIZCOOCORRENCIA.JAVA

```
/**
 * Está é classe obtem a matriz de co-ocorrência de uma imagem.
 * Esta matriz é essencial para a extração de atributos de textura.
 */
package extractorimpl;

import java.awt.image.*;
import javax.media.jai.*;
import com.sun.media.jai.widget.DisplayJAI;

public class MatrizCoOcorrencia {
    private PlanarImage image;
    private int distancia;
    private int angle;
    private String sinal = "";
    private double matriz[][] = new double[451][451];
    private int pixelAtual;

    private int width;
    private int height;
    private SampleModel sm;
    private int nbands;
    private int[] pixels;
    private int offset;

    /**
     * Contrutor da classe que recebe uma imagem para que
     * dela seja obtida a matriz de co-ocorrência.
     */
    public MatrizCoOcorrencia( PlanarImage im ) {
        this.image = im;
        /* Instancia a matriz com zero em todas as posições!!! */
        for( int linha = 0; linha < 451; linha ++ )
            for( int coluna = 0; coluna < 451; coluna ++ )
                matriz[linha][coluna] = 0;
        width = image.getWidth();
        height = image.getHeight();
        sm = image.getSampleModel();
        nbands = sm.getNumBands();
        pixels = new int[nbands*width*height];
    }

    /**
     * Este método calcula a matriz de coocorrencia da imagem passada ao
     * construtor
     * Parametros:
     *  - dist: distância utilizada para o cálculo da matriz (distancia em
     *          pixels)
     *  - ang: ângulo para o calculo da matriz. Este ângulo pode ser 0, 45,
     *         90 ou 135.
     *         Este método considera todas as direções do angulo passado.
     */
    public double[][] getMatriz( int dist, int an ) {
```

```
    this.distancia = dist;
    this.angle = an;
    Raster inputRaster = image.getData();
    WritableRaster outputRaster =
                          inputRaster.createCompatibleWritableRaster();
    ColorModel cm = image.getColorModel();
    TiledImage tiledImage = new TiledImage(0,0,width,height,0,0,sm,cm);
    inputRaster.getPixels(0,0,width,height,pixels);
    for (int h=0; h<height; h++)
        for (int w=0; w<width; w++) {
            offset = h*width*nbands+w*nbands;
            for (int band=0; band<nbands; band++) {
                pixelAtual = pixels[offset+band]; // valor do pixel
                getNeighbours( h, w );
            }
        }
    // calculando a soma de todos os pares da matriz criada
    int soma = 0;
    for( int linha = 0; linha < 451; linha ++ )
        for( int coluna = 0; coluna < 451; coluna ++ )
            soma += matriz[linha][coluna];
    // calculando os valores normalizados
    for( int linha = 0; linha < 451; linha ++ )
        for( int coluna = 0; coluna < 451; coluna ++ )
            matriz[linha][coluna] /= soma;
    return matriz;
}

/* Este método calcula a matriz de coocorrencia da imagem passada ao
 * construtor
 * Parametros:
 *  - dist: distância utilizada para o cálculo da matriz (distancia em
 *          pixels)
 *  - ang: ângulo para o calculo da matriz. Este ângulo pode ser 0, 45,
 *          90 ou 135
 *  - sinal: indica a direção do angulo que será calculado. Este valor
 *           pode ser "+" ou "-".
 *           se o sinal for "+", será considerada a direção sobre o
 *           angulo positivo.
 *           se o sinal for "-", será considerada a direção sobre o
 *           angulo negativo.
 */
public double[][] getMatriz( int dist, int an, String sinal ) {
    this.angle = an;
    this.distancia = dist;
    this.sinal = sinal;
    if( (!sinal.equals("+")) || (!sinal.equals("-")) ) {
        System.out.println( "O sinal deve ser \"+\" ou \"-\"!!!" );
    }
    Raster inputRaster = image.getData();
    WritableRaster outputRaster =
                           inputRaster.createCompatibleWritableRaster();
    ColorModel cm = image.getColorModel();
    TiledImage tiledImage = new TiledImage(0,0,width,height,0,0,sm,cm);
    inputRaster.getPixels(0,0,width,height,pixels);
    for (int h=0; h<height; h++) {
        for (int w=0; w<width; w++) {
            offset = h*width*nbands+w*nbands;
            for (int band=0; band<nbands; band++) {
                pixelAtual = pixels[offset+band]; // valor do pixel
                getNeighbours( h, w );
```

```
            }
        }
    }
    // calculando a soma de todos os pares da matriz criada
    int soma = 0;
    for( int linha = 0; linha < 451; linha ++ )
        for( int coluna = 0; coluna < 451; coluna ++ )
            soma += matriz[linha][coluna];
    // calculando os valores normalizados
    for( int linha = 0; linha < 451; linha ++ )
        for( int coluna = 0; coluna < 451; coluna ++ )
            matriz[linha][coluna] /= soma;
    return matriz;
}

/**
 * Este método incrementa a matriz de co-ocorrencia com as informações
 * referentes aos seus vizinhos.
 */
private void getNeighbours( int h, int w ) {
    int linha;
    int coluna;
    if( pixelAtual != 0 )
        coluna = ( int ) ( pixelAtual - 61935 )/8;
    else
        coluna = -1;
    if( sinal.equals("") ) {
        switch( angle ) {
            case 0: for( int distAux=1; distAux<distancia+1; distAux++ ) {
                        if( ( w + distAux ) < width ) {
                            offset = h*width*nbands+( w + distAux )*nbands;
                            for (int band=0; band<nbands; band++)
                                if ((pixels[offset+band]!=0)&&(coluna!=-1)) {
                                    linha= (int)(pixels[offset+band]-61935)/8;
                                    matriz[linha][coluna] += 1;
                                }
                        }
                        if( ( w - distAux ) >= 0 ) {
                            offset = h*width*nbands+( w - distAux )*nbands;
                            for (int band=0; band<nbands; band++)
                                if((pixels[offset+band]!=0)&&(coluna!=-1)) {
                                    linha= (int)(pixels[offset+band]-61935)/8;
                                    matriz[linha][coluna] += 1;
                                }
                        }
                    }
                    break;
            case 45: for(int distAux=1; distAux<distancia+1; distAux++ ) {
                        if( ((h+distAux)<height)||((w+distAux)<width) ) {
                            offset=(h+distAux)*width*nbands+
                                                    (w+distAux)*nbands;
                            for (int band=0; band<nbands; band++)
                                if ((pixels[offset+band]!=0)&&(coluna!=-1)) {
                                    linha= (int)(pixels[offset+band]-61935)/8;
                                    matriz[linha][coluna] += 1;
                                }
                        }
                        if( ( (h-distAux)>0 ) || ( (w-distAux) > 0 ) ) {
                            offset=(h-distAux)*width*nbands+
                                                    (w-distAux)*nbands;
                            for (int band=0; band<nbands; band++)
```

```
                        if ((pixels[offset+band]!=0)&&(coluna!=-1)) {
                            linha=  (int)(pixels[offset+band]-61935)/8;
                            matriz[linha][coluna] += 1;
                        }
                    }
                }
                break;
        case 90: for (int distAux=1; distAux<distancia+1; distAux++) {
                    if( ( h + distAux ) < height ) {
                        offset = (h+distAux)*width*nbands+w*nbands;
                        for (int band=0; band<nbands; band++)
                            if ((pixels[offset+band]!=0)&&(coluna!=-1)) {
                                linha=  (int)(pixels[offset+band]-61935)/8;
                                matriz[linha][coluna] += 1;
                            }
                    }
                    if( ( h - distAux ) >= 0 ) {
                        offset = ( h - distAux )*width*nbands+w*nbands;
                        for (int band=0; band<nbands; band++)
                            if ((pixels[offset+band]!=0)&&(coluna!=-1)) {
                                linha= (int)(pixels[offset+band]-61935)/8;
                                matriz[linha][coluna] += 1;
                            }
                    }
                }
                break;
        case 135: for(int distAux=1; distAux<distancia+1; distAux++){
                    if( ( (h+distAux)<height ) || ( (w-distAux)>0 ) ) {
                        offset = (h+distAux)*width*nbands+
                                                (w-distAux)*nbands;
                        for (int band=0; band<nbands; band++)
                            if((pixels[offset+band]!=0)&&(coluna!=-1)){
                                linha= (int)(pixels[offset+band]-61935)/8;
                                matriz[linha][coluna] += 1;
                            }
                    }
                    if( ( (h-distAux)>0) || ( (w+distAux) < width ) ) {
                        offset= (h-distAux)*width*nbands+
                                                (w+distAux)*nbands;
                        for (int band=0; band<nbands; band++)
                            if( (pixels[offset+band]!=0)&&(coluna!=-1) ){
                                linha= (int)(pixels[offset+band]-61935)/8;
                                matriz[linha][coluna] += 1;
                            }
                    }
                 }
                 break;
        default: System.out.println
                    ( "O angulo deve ser igual a 0, 45, 90 ou 135!!!" );
               break;
    } // end switch
} // end if
if( sinal.equals("+") ) {
    switch( angle ) {
        case 0: for( int distAux=1; distAux<distancia+1; distAux++ ) {
                    if( (w+distAux) < width ) {
                        offset = h*width*nbands+(w+distAux)*nbands;
                        for (int band=0; band<nbands; band++)
                            if((pixels[offset+band]!=0) && (coluna!=-1)){
                                linha= (int)(pixels[offset+band]-61935)/8;
                                matriz[linha][coluna] += 1;
```

```
                        }
                    }
                }
                break;
        case 45: for(int distAux=1; distAux<distancia+1; distAux++ ) {
                    if( ((h+distAux)<height) || ((w+distAux)<width) ) {
                        offset = (h+distAux)*width*nbands+
                                                    (w+distAux)*nbands;
                        for (int band=0; band<nbands; band++)
                            if( (pixels[offset+band]!=0)&&(coluna!=-1) ){
                                linha= (int)(pixels[offset+band]-61935)/8;
                                matriz[linha][coluna] += 1;
                            }
                    }
                }
                break;
        case 90: for(int distAux=1; distAux<distancia+1; distAux++) {
                    if( (h+distAux) < height ) {
                        offset = (h+distAux)*width*nbands+w*nbands;
                        for (int band=0; band<nbands; band++)
                            if( (pixels[offset+band]!=0)&&(coluna!=-1) ){
                                linha= (int)(pixels[offset+band]-61935)/8;
                                matriz[linha][coluna] += 1;
                            }
                    }
                }
                break;
        case 135: for(int distAux=1; distAux<distancia+1; distAux++) {
                    if( ((h+distAux)<height) || ((w-distAux)>0) ) {
                        offset = (h+distAux)*width*nbands+
                                                    (w-distAux)*nbands;
                        for (int band=0; band<nbands; band++)
                            if( (pixels[offset+band]!=0)&&(coluna!=-1) ){
                                linha= (int)(pixels[offset+band]-61935)/8;
                                matriz[linha][coluna] += 1;
                            }
                    }
                }
                break;
        default: System.out.println
                    ( "O angulo deve ser igual a 0, 45, 90 ou 135!!!" );
                break;
    } // end switch
} // end if
if( sinal.equals("-") ) {
    switch( angle ) {
        case 0: for(int distAux=1; distAux<distancia+1; distAux++) {
                    if( ( w - distAux ) >= 0 ) {
                        offset = h*width*nbands+( w - distAux )*nbands;
                        for (int band=0; band<nbands; band++)
                            if( (pixels[offset+band]!=0)&&(coluna!=-1) ){
                                linha= (int)(pixels[offset+band]-61935)/8;
                                matriz[linha][coluna] += 1;
                            }
                    }
                }
                break;
        case 45: for(int distAux=1; distAux<distancia+1; distAux++) {
                    if( ((h-distAux)>0) || ( (w+distAux)<width ) ) {
                        offset= (h-distAux)*width*nbands+
                                                    (w+distAux)*nbands;
```

```
                        for (int band=0; band<nbands; band++)
                            if( (pixels[offset+band]!=0)&&(coluna!=-1) ){
                                linha= (int)(pixels[offset+band]-61935)/8;
                                matriz[linha][coluna] += 1;
                            }
                      }
                    }
                    break;
            case 90: for(int distAux=1; distAux<distancia+1; distAux++ ) {
                      if( (h-distAux) >= 0 ) {
                        offset = (h-distAux)*width*nbands+w*nbands;
                        for (int band=0; band<nbands; band++)
                            if( (pixels[offset+band]!=0)&&(coluna!=-1) ){
                                linha= (int)(pixels[offset+band]-61935)/8;
                                matriz[linha][coluna] += 1;
                            }
                      }
                    }
                    break;
            case 135: for(int distAux=1; distAux<distancia+1; distAux++) {
                      if( ( ( h-distAux)>0 ) || ( (w-distAux) >0 ) ) {
                        offset = (h-distAux)*width*nbands+
                                                    (w-distAux)*nbands;
                        for (int band=0; band<nbands; band++)
                            if( (pixels[offset+band]!=0)&&(coluna!=-1) ){
                                linha= (int)(pixels[offset+band]-61935)/8;
                                matriz[linha][coluna] += 1;
                            }
                      }
                     }
                     break;
            default: System.out.println
                      ( "O angulo deve ser igual a 0, 45, 90 ou 135!!!" );
                   break;
          } // end switch
        } // end if
    }
}
```

APÊNDICE F – CÓDIGO FONTE DO EXTRATOR EXTRAIENTROPIA.JAVA

```
/**
 * Está é uma classe que implementa um extrator da característica
 * de textura entropia.
 * Esta classe extende a classe AbstractExtrator, que serve como
 * base para a criação dos extratores de característica deste sistema.
 */
package extractorimpl;

import java.lang.Math;
import java.awt.Rectangle;
import java.awt.image.Raster;
import java.awt.image.SampleModel;

import javax.media.jai.PlanarImage;
import javax.media.jai.iterator.RectIter;
import javax.media.jai.iterator.RectIterFactory;

import extractor.AbstractExtractor;
import extractor.ParameterBlock;
import extractorimpl.MatrizCoOcorrencia;

public class ExtraiEntropia extends AbstractExtractor {

    /** variação permitida para que pixels sejam considerados iguais */
    private int thr;
    // Parâmetros para o calculo da Matriz de co-ocorrencia
    private int distancia = 1;
    private int angulo = 0;
    private String sinal = "";

    /** Cria um objeto associado a uma imagem.
     *
     * @param p - imagem a ser utilizada.
     */
    public ExtraiEntropia(PlanarImage p, int ang, int dist, String sinal)
    {
        super(p);
        distancia = dist;
        angulo = ang;
        this.sinal = sinal;
    }

    /**
     * Compara a entropia das duas imagens. Se a diferença entre as duas
     * entropias for menor ou igual ao threshold definido, eles são
     * considerados iguais.
     * O resultado retornado é o número 1 se as áreas são iguais,
     * considerando-se o threshold, ou 0 no caso das imagens serem
     * consideradas diferentes.
     */

    public double compare(AbstractExtractor x) {
        if ( ! (x instanceof ExtraiEntropia) )
        {
            throw new IllegalArgumentException
```

```
                                        ("ExtraiEntropia extractor required.");
        }
        ExtraiEntropia entropia = (ExtraiEntropia) x;
        double entrop1 = computeValue(), entrop2 = entropia.computeValue(),
                         dif;
        dif = entrop1 > entrop2 ? entrop1 - entrop2 : entrop2 - entrop1;
        if (dif > thr)
            return 0;
        else
            return 1;
    }

    public void setDistancia( int dist ) {
        this.distancia = dist;
    }

    public void setAngle( int angle ) {
        this.angulo = angle;
    }

    public void setSinal( String sinal ) {
        this.sinal = sinal;
    }

    public double computeValue() {
        double[][] matriz = getData();
        double entropia = 0;
        for( int linha = 0; linha < 451; linha++ )
            for( int coluna = 0; coluna < 451; coluna++ )
                if( matriz[linha][coluna] != 0 )
                    entropia += matriz[linha][coluna] *
                                (Math.log(matriz[linha][coluna])/Math.log(2));
        return 0 - entropia;
    }

    /** Obtem, na forma de uma matriz, os valores da matriz de coocorrencia
     * da imagem para que seja feito o cálculo da entropia comparação.
     *
     * @return - vetor de inteiro com os pixels a serem comparados,
     * que representam a região da imagem definida.
     */
    public double[][] getData()
    {
        MatrizCoOcorrencia matrizCO = new MatrizCoOcorrencia( getImage() );
        double[][] matriz = new double[451][451];
        if( sinal.equals( "" ) )
            matriz = matrizCO.getMatriz( distancia, angulo );
        else
            matriz = matrizCO.getMatriz( distancia, angulo, sinal );
        return matriz;
    }

    /**
     * Define os parâmetros a serem utilizados por esse objeto.
     *
     */
    public void setParameters(ParameterBlock pb) {
        thr = pb.getIntParameter(2);
        thr = thr < 0 ? 0 : thr;
    }
}
```

APÊNDICE G – CÓDIGO FONTE DO EXTRATOR EXTRAISMA.JAVA

```
/**
 * Está é uma classe que implementa um extrator da característica
 * de textura Segundo Momento Angular(SMA).
 * Esta classe extende a classe AbstractExtrator, que serve como
 * base para a criação dos extratores de característica deste sistema.
 */
package extractorimpl;

import java.awt.Rectangle;
import java.awt.image.Raster;
import java.awt.image.SampleModel;

import javax.media.jai.PlanarImage;
import javax.media.jai.iterator.RectIter;
import javax.media.jai.iterator.RectIterFactory;

import extractor.AbstractExtractor;
import extractor.ParameterBlock;
import extractorimpl.MatrizCoOcorrencia;

public class ExtraiSMA extends AbstractExtractor {

    /** variação permitida para que pixels sejam considerados iguais */
    private int thr;
    // Parâmetros para o calculo da Matriz de co-ocorrencia
    private int distancia = 1;
    private int angulo = 0;
    private String sinal = "";

    /** Cria um objeto associado a uma imagem.
     *
     * @param p - imagem a ser utilizada.
     */
    public ExtraiSMA(PlanarImage p, int ang, int dist, String sinal)
    {
        super(p);
        distancia = dist;
        angulo = ang;
        this.sinal = sinal;
    }

    /**
     * Compara o valor do SMA das duas imagens. Se a diferenç afor menor
     * ou igual ao threshold definido, as imagens são considerados iguais.
     * O resultado retornado é o número 1 se os valores são iguais,
     * considerando-se o threshold, ou 0 no caso dos valores serem
     * consideradas diferentes.
     */

    public double compare(AbstractExtractor x) {
        if ( ! (x instanceof ExtraiSMA) )
        {
            throw new IllegalArgumentException
                                        ("ExtraiSMA extractor required.");
        }
        ExtraiSMA sma = (ExtraiSMA) x;
```

```
        double sma1 = computeValue(), sma2 = sma.computeValue(), dif;

        dif = sma1 > sma2 ? sma1 - sma2 : sma2 - sma1;

        if (dif > thr)
            return 0;
        else
            return 1;
    }

    public void setDistancia( int dist ) {
        this.distancia = dist;
    }

    public void setAngle( int angle ) {
        this.angulo = angle;
    }

    public void setSinal( String sinal ) {
        this.sinal = sinal;
    }

    public double computeValue() {
        double[][] matriz = getData();
        double sma = 0;
        for( int linha = 0; linha < 450; linha++ )
            for( int coluna = 0; coluna < 450; coluna++ )
                sma += matriz[linha][coluna] * matriz[linha][coluna];
        return sma;
    }

    /** Obtem, na forma de uma matriz, os valores da matriz de coocorrencia
     * da imagem para que seja feito o cálculo da entropia comparação.
     *
     * @return - vetor de inteiro com os pixels a serem comparados,
     * que representam a região da imagem definida.
     */
    public double[][] getData()
    {
        MatrizCoOcorrencia matrizCO = new MatrizCoOcorrencia( getImage() );
        double[][] matriz = new double[450][450];
        if( sinal.equals( "" ) )
            matriz = matrizCO.getMatriz( distancia, angulo );
        else
            matriz = matrizCO.getMatriz( distancia, angulo, sinal );
        return matriz;
    }

    /**
     * Define os parâmetros a serem utilizados por esse objeto.
     *
     */
    public void setParameters(ParameterBlock pb) {
        thr = pb.getIntParameter(2);
        thr = thr < 0 ? 0 : thr;
    }
}
```

APÊNDICE H - CÓDIGO FONTE DO EXTRATOR EXTRAICONTRASTE.JAVA

```
/**
 * Está é uma classe que implementa um extrator da característica
 * de textura Contraste.
 * Esta classe extende a classe AbstractExtrator, que serve como
 * base para a criação dos extratores de característica deste sistema.
 */
package extractorimpl;

import java.lang.Math;
import java.awt.Rectangle;
import java.awt.image.Raster;
import java.awt.image.SampleModel;

import javax.media.jai.PlanarImage;
import javax.media.jai.iterator.RectIter;
import javax.media.jai.iterator.RectIterFactory;

import extractor.AbstractExtractor;
import extractor.ParameterBlock;
import extractorimpl.MatrizCoOcorrencia;

public class ExtraiContraste extends AbstractExtractor {

    /** variação permitida para que pixels sejam considerados iguais */
    private int thr;
    // Parâmetros para o calculo da Matriz de co-ocorrencia
    private int distancia = 1;
    private int angulo = 0;
    private String sinal = "";

    /** Cria um objeto associado a uma imagem.
     *
     * @param p - imagem a ser utilizada.
     */
    public ExtraiContraste(PlanarImage p, int ang, int dist, String sinal)
    {
        super(p);
        distancia = dist;
        angulo = ang;
        this.sinal = sinal;
    }

    /**
     * Compara o contraste das duas imagens. Se a diferença for menor
     * ou igual ao threshold definido, eles são considerados iguais.
     * O resultado retornado é o número 1 quando iguais,
     * considerando-se o threshold, ou 0 no caso dos valores serem
     * consideradas diferentes.
     */

    public double compare(AbstractExtractor x) {
        if ( ! (x instanceof ExtraiSMA) )
        {
```

```
            throw new IllegalArgumentException
                                ("ExtraiContraste extractor required.");
        }
        ExtraiContraste contr = (ExtraiContraste) x;
        double contr1 = computeValue(), contr2 = contr.computeValue(), dif;

        dif = contr1 > contr2 ? contr1 - contr2 : contr2 - contr1;
        if (dif > thr)
            return 0;
        else
            return 1;
    }

    public void setDistancia( int dist ) {
        this.distancia = dist;
    }

    public void setAngle( int angle ) {
        this.angulo = angle;
    }

    public void setSinal( String sinal ) {
        this.sinal = sinal;
    }

    public double computeValue() {
        double[][] matriz = getData();
        double contr = 0;
        double calcAux;
        for( int n = 0; n < 450; n++ ) {
            calcAux = 0;
            for( int linha = 0; linha < 450; linha++ )
                for( int coluna = 0; coluna < 450; coluna++ )
                    if( Math.abs( linha - coluna ) == n )
                        calcAux += matriz[linha][coluna];
            contr += n * n * calcAux;
        }
        return contr;
    }

    /** Obtem, na forma de uma matriz, os valores da matriz de coocorrencia
     * da imagem para que seja feito o cálculo da entropia comparação.
     *
     * @return - vetor de inteiro com os pixels a serem comparados,
     * que representam a região da imagem definida.
     */
    public double[][] getData()
    {
        MatrizCoOcorrencia matrizCO = new MatrizCoOcorrencia( getImage() );
        double[][] matriz = new double[450][450];
        if( sinal.equals( "" ) )
            matriz = matrizCO.getMatriz( distancia, angulo );
        else
            matriz = matrizCO.getMatriz( distancia, angulo, sinal );
        return matriz;
    }

    /**
     * Define os parâmetros a serem utilizados por esse objeto.
     *
     */
```

```
    public void setParameters(ParameterBlock pb) {
        thr = pb.getIntParameter(2);
        thr = thr < 0 ? 0 : thr;
    }

}
```

APÊNDICE I – CÓDIGOS FONTE DA IMPLEMENTAÇÃO DAS INTERFACES

```
/**
 * Código fonte da classe TelaImagem.java que implementa uma
 * tela para a manutenção das imagens presentes no banco.
 */
package interfaces;

import dbmanager.ConnectionManager;
import dbmanager.ImageManager;

import java.awt.*;
import java.awt.event.*;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.swing.*;

public class TelaImagem extends JFrame implements ActionListener {
   private JTextArea displayArea;
   private Container container;
   private JTextField codigo;
   private JTextField path;

   public TelaImagem() {

      super( "Manutenção das Imagens" );

      JLabel lcod = new JLabel( "Código" );
      JLabel lpath = new JLabel( "Path" );
      JButton save = new JButton( "Save" );
      JButton delete = new JButton( "Delete" );
      codigo = new JTextField();
      path = new JTextField();

      container = getContentPane();
      container.setLayout( new BorderLayout() );

      displayArea = new JTextArea();
      container.add( new JScrollPane(displayArea), BorderLayout.CENTER );

      JPanel painel1 = new JPanel( new GridLayout( 2, 1 ) );

      save.addActionListener( this );
      delete.addActionListener( this );

      JPanel codPanel = new JPanel( new FlowLayout( FlowLayout.CENTER ) );
      codPanel.add( lcod );
      codigo.setColumns( 30 );
      codPanel.add( codigo );
      codPanel.add( save );
      JPanel pathPanel = new JPanel( new FlowLayout(FlowLayout.CENTER) );
      pathPanel.add( lpath );
      path.setColumns( 30 );
      pathPanel.add( path );
```

```
    pathPanel.add( delete );

    painel1.add( codPanel );
    painel1.add( pathPanel );

    container.add( painel1, BorderLayout.SOUTH );

    // inicia conexão banco
    ConnectionManager.ConnectDB();

    displayArea.setText( "" );
    // consulta a base de dados
    ImageManager aux = new ImageManager();
    ResultSet rs = aux.findall();
    try{
       while( rs.next() )
       {
          ImageManager temp = ImageManager.SetObject(rs);
          String dados = temp.mostraDados();
          displayArea.append( dados );
          displayArea.append("------------------------------\n");
       }
    }
    catch( SQLException e ) {
       displayArea.append( "Erro: " + e.getErrorCode() + " msg: " +
                           e.getMessage() );
    }
    ConnectionManager.CloseDB();


    setSize( 500, 500 );
    setLocation( 200, 100 );
    setVisible( true );
}

public void actionPerformed ( ActionEvent event ) {

    // inicia conexão banco
    ConnectionManager.ConnectDB();

    int codigoIm = Integer.valueOf( codigo.getText() ).intValue();
    String pathIm = path.getText();
    ImageManager im = new ImageManager( codigoIm, pathIm );

    if ( event.getActionCommand() == "Save" ) {
       im.save();
    }
    else if (event.getActionCommand() == "Delete" ) {
       im.delete();
    }

    // exibe todo o conteúdo da tabela depois da alteração
    //(inserção/ alteração/ exclusão)
    displayArea.setText( "" );
    ImageManager aux = new ImageManager();
    ResultSet rs = aux.findall();
    try{
       while( rs.next() )
       {
          ImageManager temp = ImageManager.SetObject(rs);
          String dados = temp.mostraDados();
```

```
               displayArea.append( dados );
               displayArea.append("------------------------------\n");
            }
         }
         catch( SQLException e ) {
            displayArea.append( "Erro: " + e.getErrorCode() + " msg: " +
                                e.getMessage());
         }
         ConnectionManager.CloseDB();

      }
   }
```

```
/**
 * Código fonte da classe TelaCaract.java que implementa uma
 * tela para a manutenção das características presentes no banco.
 */
package interfaces;

import dbmanager.ConnectionManager;
import dbmanager.CaractManager;

import java.awt.*;
import java.awt.event.*;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.swing.*;

public class TelaCaract extends JFrame implements ActionListener {
   private JTextArea displayArea;
   private Container container;
   private JTextField codigo;
   private JTextField nome;

   public TelaCaract() {

      super( "Manutenção das Características" );

      JLabel lcod = new JLabel( "Código" );
      JLabel lnome = new JLabel( "Nome" );
      JButton save = new JButton( "Save" );
      JButton delete = new JButton( "Delete" );
      codigo = new JTextField();
      nome = new JTextField();

      container = getContentPane();
      container.setLayout( new BorderLayout() );

      displayArea = new JTextArea();
      container.add( new JScrollPane(displayArea), BorderLayout.CENTER );

      JPanel painel1 = new JPanel( new GridLayout( 2, 3 ) );

      save.addActionListener( this );
      delete.addActionListener( this );

      JPanel codPanel = new JPanel( new FlowLayout(FlowLayout.CENTER) );
```

```
    codPanel.add( lcod );
    codigo.setColumns( 30 );
    codPanel.add( codigo );
    codPanel.add( save );
    JPanel nomePanel = new JPanel( new FlowLayout(FlowLayout.CENTER) );
    nomePanel.add( lnome );
    nome.setColumns( 30 );
    nomePanel.add( nome );
    nomePanel.add( delete );

    painel1.add( codPanel );
    painel1.add( nomePanel );

    container.add( painel1, BorderLayout.SOUTH );

    // inicia conexão banco
    ConnectionManager.ConnectDB();

    displayArea.setText( "" );
    // consulta a base de dados
    CaractManager aux = new CaractManager();
    ResultSet rs = aux.findall();
    try{
        while( rs.next() )
        {
            CaractManager temp = CaractManager.SetObject(rs);
            String dados = temp.mostraDados();
            displayArea.append( dados );
            displayArea.append("------------------------------\n");
        }
    }
    catch( SQLException e ) {
        displayArea.append( "Erro: " + e.getErrorCode() + " msg: " +
                                e.getMessage());
    }
    ConnectionManager.CloseDB();


    setSize( 500, 500 );
    setLocation( 100, 100 );
    setVisible( true );
}

public void actionPerformed ( ActionEvent event ) {

    // inicia conexão banco
    ConnectionManager.ConnectDB();

    int codigoC = Integer.valueOf( codigo.getText() ).intValue();
    String nomeC = nome.getText();
    CaractManager carcat = new CaractManager( codigoC, nomeC );

    if ( event.getActionCommand() == "Save" ) {
        carcat.save();
    }
    else if (event.getActionCommand() == "Delete" ) {
        carcat.delete();
    }

    // exibe todo o conteúdo da tabela depois da alteração
    //(inserção/ alteração/ exclusão)
```

```
        displayArea.setText( "" );
        CaractManager aux = new CaractManager();
        ResultSet rs = aux.findall();
        try{
           while( rs.next() )
           {
              CaractManager temp = CaractManager.SetObject(rs);
              String dados = temp.mostraDados();
              displayArea.append( dados );
              displayArea.append("------------------------------\n");
           }
        }
        catch( SQLException e ) {
           displayArea.append( "Erro: " + e.getErrorCode() + " msg: " +
                               e.getMessage());
        }
        ConnectionManager.CloseDB();

    }
}



/**
 * Código fonte da classe PainelParam.java que implementa uma
 * um JPanel para a definição dos parâmetros da matriz de
 * co-ocorrência, servindo de apoio para a tela implementada
 * na classe ParamMCO.java
 */
package interfaces;

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

public class PainelParam {
    public String angulos[] = { "0", "45", "90", "135" };
    public String sinais[] = { "", "+", "-" };
    public JComboBox angulo, sinal;
    public JTextField distancia;

    public JPanel mostrar(boolean editable) {
        JPanel painel = new JPanel();

        distancia = new JTextField( "1", 2 );

        angulo = new JComboBox( angulos );
        sinal = new JComboBox( sinais );

        JLabel dist = new JLabel("Dist");
        JLabel ang = new JLabel("Ângulo:");

        distancia.setEditable( editable );
        sinal.setEnabled( editable );
        angulo.setEnabled( editable );

        painel.add( dist );
        painel.add( distancia );
        painel.add( ang );
        painel.add( sinal );
```

```
        painel.add( angulo );

        return painel;
    }
}


/**
 * Código fonte da classe ParamMCO.java que implementa uma
 * tela para a definição dos parâmetros para a construção da
 * matriz de co-ocorrência das características de textura.
 */
package interfaces;

import java.awt.*;
import java.awt.event.*;

import javax.swing.*;

public class ParamMCO extends JFrame{
    private int anguloEntr=0, anguloSMA=0, anguloContr=0, distanciaEntr=1,
                distanciaSMA=1, distanciaContr=1;
    private String sinalEntr="", sinalSMA="", sinalContr="";
    private JPanel parEntrop, parSMA, parContraste;
    private PainelParam painelEntr, painelSMA, painelCont;
    private boolean entropia, sma, contraste;

    public ParamMCO() {
    }

    public ParamMCO( boolean entropia, boolean sma, boolean contraste ) {
        super( "Parâmetros - Características de Textura" );

        this.entropia = entropia;
        this.sma = sma;
        this.contraste = contraste;

        Container container = getContentPane();
        container.setLayout( new BorderLayout() );

        JPanel centro = new JPanel( new GridLayout( 3, 2 ) );

        centro.add( new JLabel( "Entropia: " ) );
        painelEntr = new PainelParam();
        parEntrop = painelEntr.mostrar( entropia );
        centro.add( parEntrop );

        centro.add( new JLabel( "Segundo Momento Angular: " ) );
        painelSMA = new PainelParam();
        parSMA = painelSMA.mostrar( sma );
        centro.add( parSMA );

        centro.add( new JLabel( "Contraste: " ) );
        painelCont = new PainelParam();
        parContraste = painelCont.mostrar( contraste );
        centro.add( parContraste );

        container.add( centro, BorderLayout.CENTER );

        GridBagLayout layout = new GridBagLayout();
        GridBagConstraints constraints = new GridBagConstraints();
```

```
        JPanel botao = new JPanel( new FlowLayout( FlowLayout.CENTER ) );
        JButton salvar = new JButton( "Salvar Parâmetros" );
        ButtonHandler handler = new ButtonHandler();
        salvar.addActionListener( handler );

        constraints.gridx = 1;
        constraints.gridy = 1;

        constraints.gridwidth = 1;
        constraints.gridheight = 1;

        layout.setConstraints( salvar, constraints );
        botao.add( salvar );

        container.add( botao, BorderLayout.SOUTH );

        setVisible( true );
        setSize( 410, 160 );
        setLocation( 150, 200 );
    }

    private class ButtonHandler implements ActionListener {
        public void actionPerformed( ActionEvent event ) {
            if( event.getActionCommand() == "Salvar Parâmetros" ) {
                if( entropia == true ) {
                    // Entropia
                    anguloEntr = Integer.parseInt( painelEntr.angulos[
                                    painelEntr.angulo.getSelectedIndex() ] );
                    distanciaEntr = Integer.parseInt(
                                        painelEntr.distancia.getText() );
                    sinalEntr = painelEntr.sinais[
                                    painelEntr.sinal.getSelectedIndex() ];
                }
                if( sma == true ) {
                    // SMA
                    anguloSMA = Integer.parseInt( painelSMA.angulos[
                                    painelSMA.angulo.getSelectedIndex() ] );
                    distanciaSMA = Integer.parseInt(
                                        painelSMA.distancia.getText() );
                    sinalSMA = painelSMA.sinais[
                                    painelSMA.sinal.getSelectedIndex() ];
                }
                if( contraste == true ) {
                    // Contraste
                    anguloContr = Integer.parseInt( painelCont.angulos[
                                    painelCont.angulo.getSelectedIndex() ] );
                       distanciaContr = Integer.parseInt(
                                        painelCont.distancia.getText() );
                       sinalContr = painelCont.sinais[
                                    painelCont.sinal.getSelectedIndex() ];
                }
                setVisible( false );
            }
        }
    }

/* * * * * * * * * * * * * * *  E N T R O P I A  * * * * * * * * * * * * * */
    public int getAnguloEntropia() {
        return anguloEntr;
    }
```

```
    public int getDistanciaEntropia(){
        return distanciaEntr;
    }

    public String getSinalEntropia() {
        return sinalEntr;
    }

/* * * * * * * * * * * * * * * * * *  S M A  * * * * * * * * * * * * * * * * */
    public int getAnguloSMA() {
        return anguloSMA;
    }

    public int getDistanciaSMA(){
        return distanciaSMA;
    }

    public String getSinalSMA() {
        return sinalSMA;
    }

/* * * * * * * * * * * * * * *  C O N T R A S T E  * * * * * * * * * * * * */
    public int getAnguloContraste() {
        return anguloContr;
    }

    public int getDistanciaContraste(){
        return distanciaContr;
    }

    public String getSinalContraste() {
        return sinalContr;
    }
}



/**
 * Código fonte da classe Principal.java que implementa uma
 * tela para as consultas das imagens presentes no banco de
 * acordo com as características selecionadas.
 */
package interfaces;

import dbmanager.ConnectionManager;
import dbmanager.ImageManager;
import dbmanager.CaractManager;
import dbmanager.ImageCaractManager;
import interfaces.TelaImagem;
import interfaces.TelaCaract;
import interfaces.ParamMCO;
import segmentacao.TiraFundo;
import similarity.CharVector;
import extractorimpl.*;

import java.util.Arrays;
import java.awt.*;
import java.awt.event.*;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
import java.util.*;
import java.text.DateFormat;

import com.sun.media.jai.widget.DisplayJAI;
import javax.media.jai.*;
import java.awt.image.*;
import java.awt.image.renderable.ParameterBlock;
import javax.management.timer.Timer;

import javax.swing.*;

public class Principal extends JFrame {
   private JComboBox selecao;
   private JCheckBox caracteristicas[];
   private JPanel esquerda, recuperadas, carac;
   private JTextField tqtdade;
   private Container container;
   private GridBagLayout layout, layoutCar;
   private GridBagConstraints constraints, constraintsCar;
   private String imagem_paths[], caract_nomes[];
   private int imagem_codigos[], caract_codigos[], caract_selecionadas[];
   private int tamanhoIm, tamanhoCar, indiceIm, indiceCar;
   private DisplayJAI dj = new DisplayJAI();
   private ParamMCO parametros;

   public Principal() {
      super( "Sistema de Recuperação de Imagens Mamográficas" );

      JMenuBar barraMenus = new JMenuBar();
      setJMenuBar( barraMenus );

      JMenu menuArq = new JMenu( "Arquivo" );
      menuArq.setMnemonic( 'A' );
      JMenu menuManut = new JMenu( "Manutenção" );
      menuManut.setMnemonic( 'M' );

      JMenuItem nova = new JMenuItem( "Nova Consulta" );
      nova.setMnemonic( 'N' );
      menuArq.add( nova );
      nova.addActionListener (
         new ActionListener() {
            public void actionPerformed( ActionEvent event ) {
               /* * * * * * * * * * * * * * * * * * * * * * * * * * * *
                  Trecho que limpa todos os componentes da tela
                * * * * * * * * * * * * * * * * * * * * * * * * * * * */
               new Principal();
               setVisible( false );
            }
         }
      );

      JMenuItem sair = new JMenuItem( "Sair" );
      sair.setMnemonic( 'S' );
      menuArq.add( sair );
      sair.addActionListener (
         new ActionListener() {
            public void actionPerformed( ActionEvent event ) {
               /* * * * * * * * * * * * * * * * * * *
                     Trecho que fecha o aplicativo
                * * * * * * * * * * * * * * * * * * * */
               System.exit( 0 );
```

```
        }
    }
);

JMenuItem imagemManut = new JMenuItem( "Imagens" );
imagemManut.setMnemonic( 'I' );
menuManut.add( imagemManut );
imagemManut.addActionListener (
    new ActionListener() {
        public void actionPerformed( ActionEvent event ) {
            /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
               Trecho que chama a tela de manutenção de Imagem
             * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
             TelaImagem telaImagem = new TelaImagem();
         }
    }
);

JMenuItem caractManut = new JMenuItem( "Características" );
caractManut.setMnemonic( 'C' );
menuManut.add( caractManut );
caractManut.addActionListener (
    new ActionListener() {
        public void actionPerformed( ActionEvent event ) {
            /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              Trecho que chama a tela de manutenção de Características
             * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
             TelaCaract telaCaract = new TelaCaract();
         }
    }
);

barraMenus.add( menuArq );
barraMenus.add( menuManut );

container = getContentPane();
container.setLayout( new BorderLayout() );

// inicia conexão banco
ConnectionManager.ConnectDB();

    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
              CRIAÇÃO DOS JCHECKBOX DAS CARACTERÍSTICAS
     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    CaractManager auxC = new CaractManager();
    tamanhoCar = auxC.length();
    caracteristicas = new JCheckBox[ tamanhoCar ];
    caract_nomes = new String[ tamanhoCar ];
    caract_codigos = new int[ tamanhoCar ];
    caract_selecionadas = new int[ tamanhoCar ];
    ResultSet rs = auxC.findall();
    indiceCar = 0;
    try{
        while( rs.next() ) {
            CaractManager temp = CaractManager.SetObject(rs);
            caracteristicas[ indiceCar ] = new JCheckBox(
                                                temp.getnome() );
            caract_nomes[ indiceCar ] = temp.getnome();
            caract_codigos[indiceCar ] = temp.getcodigo();
            caract_selecionadas[ indiceCar ] = -1; // não selecionado
            indiceCar++;
```

```
        }
    }
    catch( SQLException e ) {
        System.out.println( "Erro: " + e.getErrorCode() + " msg: " +
                                e.getMessage());
    }

    /* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
        CRIAÇÃO DA STRING DO JCOMBOBOX DAS IMAGENS
     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
    ImageManager auxI = new ImageManager();
    tamanhoIm = auxI.length();
    String names[] = new String[ tamanhoIm + 1 ];
    names[0] = "";
    rs = auxI.findall();
    indiceIm = 1;
    imagem_paths = new String[ tamanhoIm ];
    imagem_codigos = new int[ tamanhoIm ];
    try{
        while( rs.next() ) {
            ImageManager temp = ImageManager.SetObject(rs);
            names[indiceIm] = temp.getpath();
            imagem_paths[ indiceIm-1 ] = temp.getpath();
            imagem_codigos[ indiceIm-1 ] = temp.getcodigo();
            indiceIm++;
        }
    }
    catch( SQLException e ) {
        System.out.println( "Erro: " + e.getErrorCode() + " msg: " +
                                e.getMessage());
    }

// encerra a conecão com o banco
ConnectionManager.CloseDB();

JLabel imModelo = new JLabel( " Imagem de Referência:" );

selecao = new JComboBox( names );
selecao.setMaximumRowCount( 7 );
selecao.addItemListener(
    new ItemListener() {
        public void itemStateChanged( ItemEvent event ) {
            /* * * * * * * * * * * * * * * * * * * * * * * * *
                Trecho que trata o ComboBox das Imagens
             * * * * * * * * * * * * * * * * * * * * * * * * */
            indiceIm = selecao.getSelectedIndex();
            if( indiceIm != 0 ) {
                PlanarImage selecao = JAI.create("fileload",
                                        imagem_paths[ indiceIm-1 ]);
                ParameterBlock pb = new ParameterBlock();
                pb.addSource(selecao);
                pb.add(0.10f);
                pb.add(0.10f);
                pb.add(0.0F);
                pb.add(0.0F);
                PlanarImage imRed = JAI.create("scale", pb);
                dj.set( imRed );
            }
        }
    }
);
```

```
    JPanel painelQtdd = new JPanel();
    JLabel qtdade = new JLabel( " Quantidade de imagens:" );
    tqtdade = new JTextField( "5", 4 );
    painelQtdd.add( qtdade );
    painelQtdd.add( tqtdade );

    JButton procura = new JButton( "Procurar" );
    ButtonHandler bhandler = new ButtonHandler();
    procura.addActionListener( bhandler );
    procura.setMnemonic( 'P' );

    JButton altPar = new JButton( "Altera Parâmetros" );
    altPar.addActionListener( bhandler );
    altPar.setMnemonic( 'l' );

    layout = new GridBagLayout();
    constraints = new GridBagConstraints();

    esquerda = new JPanel( layout );

    layoutCar = new GridBagLayout();
    constraintsCar = new GridBagConstraints();

    carac = new JPanel( new GridLayout( tamanhoCar, 1 ) );
    carac.setBorder( BorderFactory.createTitledBorder(
                                              "Características:" ) );
    CheckBoxHandler cbhandler = new CheckBoxHandler();
    for( indiceCar = 0; indiceCar < tamanhoCar; indiceCar++ ) {
       String nome = caract_nomes[ indiceCar ];
       caracteristicas[ indiceCar ].addItemListener( cbhandler );
       addComponentCar(caracteristicas[indiceCar], indiceCar, 0, 1, 1);
    }


    // redimensiona apenas na horizontal
    constraints.fill = GridBagConstraints.HORIZONTAL;
    addComponent( carac, 1, 1, 1, tamanhoCar );
    constraints.fill = GridBagConstraints.NONE;
    addComponent( altPar, tamanhoCar+2, 1, 1, 1);
    constraints.fill = GridBagConstraints.HORIZONTAL;
    addComponent( imModelo, tamanhoCar+3, 1, 1, 1 );
    addComponent( selecao, tamanhoCar+4, 1, 1, 1 );
    constraints.fill = GridBagConstraints.NONE;
    addComponent( dj, tamanhoCar+5, 1, 1, 1 );
    addComponent( painelQtdd, tamanhoCar+6, 1, 1, 1 );
    addComponent( procura, tamanhoCar+7, 1, 1, 1 );


    container.add( esquerda, BorderLayout.WEST );

    JPanel rec = new JPanel();
    rec.setBorder( BorderFactory.createTitledBorder(
                                 "Imagens Recuperadas do Banco:" ) );
    container.add( rec, BorderLayout.CENTER );

    setSize( 900, 650 );
    setVisible( true );
}

private void addComponentCar( Component component, int row, int column,
```

```
                                int width, int height ) {
    constraintsCar.gridx = column;
    constraintsCar.gridy = row;

    constraintsCar.gridwidth = width;
    constraintsCar.gridheight = height;

    layoutCar.setConstraints( component, constraintsCar );
    carac.add( component );
}

private void addComponent( Component component, int row, int column,
                           int width, int height ) {
    constraints.gridx = column;
    constraints.gridy = row;

    constraints.gridwidth = width;
    constraints.gridheight = height;

    layout.setConstraints( component, constraints );
    esquerda.add( component );
}

/* Classe que seta as variaveis booleanas area, densidade e forma com
 * verdadeiro caso estejam selecionadas e falso se não estiverem
 * selecionadas
 */
private class CheckBoxHandler implements ItemListener {
    public void itemStateChanged( ItemEvent event ) {
        String nome = "";
        for( indiceCar = 0; indiceCar < tamanhoCar; indiceCar++ )
            if( event.getSource() == caracteristicas[ indiceCar ] )
                if( event.getStateChange() == ItemEvent.SELECTED )
                    // a característica daquela posição está selecionada
                    caract_selecionadas[ indiceCar ] = 1;
                else if( event.getStateChange() == ItemEvent.DESELECTED )
                // a característica daquele posição não está selecionada
                    caract_selecionadas[ indiceCar ] = -1;
    }
}

private class ButtonHandler implements ActionListener {
    public void actionPerformed( ActionEvent event ) {
        if( event.getActionCommand() == "Procurar" ) {
            long timer = System.currentTimeMillis();
            long startTime = System.nanoTime();
            indiceIm = selecao.getSelectedIndex()-1;
            CharVector cv1 = getVetorDaImagemAtual();
            double[] similaridades = calculaTodasSimilaridades( cv1 );
            double[] similOrdenada = new double[ similaridades.length ];
            for( int i = 0; i < similaridades.length; i++ )
                similOrdenada[ i ] = similaridades[ i ];
            Arrays.sort( similOrdenada );

            int qtImagens = Integer.parseInt( tqtdade.getText() );
            double val = (double) qtImagens / 2;
            recuperadas = new JPanel( new GridLayout( 2,
                                        (int) Math.ceil( val ) ) );
            recuperadas.setBorder( BorderFactory.createTitledBorder(
                                "Imagens Recuperadas do Banco:" ) );
```

```
            int[] cods = new int[qtImagens];
            for( int indice = 0; indice < qtImagens; indice++ ) {
               int i = 0, terminou = 0;
               while( terminou == 0 ) {
                  while( similOrdenada[ indice ] != similaridades[ i ] )
                     i++;
                  terminou = 1;
                  for( int j=0; j<indice; j++ ) {
                     if( cods[j] == i ) {
                        i++;
                        terminou = 0;
                     }
                  }
               }
               cods[indice] = i;
               recuperadas.add( imagemLabel( imagem_paths[ i ] ) );
            }
            container.add( recuperadas, BorderLayout.CENTER );
            timer = System.currentTimeMillis() - timer;
            System.out.println ("Processamento levou " + timer +
                                " milissegundos.");
            System.out.println ("Tempo de processamento (mm:ss:milis)= " +
                                (int)((int)timer/1000)/60 + ":" +
                                ((int)timer/1000)%60 + ":" + timer%1000 );
            long estimatedTime = System.nanoTime() - startTime;
            System.out.println ("Processamento levou " + estimatedTime +
                                " nanosegundos.");

            setVisible( true );
         }
         if( event.getActionCommand() == "Altera Parâmetros" ) {
            boolean entropia = false;
            boolean sma = false;
            boolean contraste = false;
            for( indiceCar = 0; indiceCar < tamanhoCar; indiceCar++ )
               if( caract_selecionadas[ indiceCar ] != -1 ) {
                  String nome = caract_nomes[ indiceCar ];
                  if( nome.equals("Entropia") )
                     entropia = true;
                  else if( nome.equals("Segundo Momento Angular") )
                     sma = true;
                  else if( nome.equals("Contraste") )
                     contraste = true;
               }
            parametros = new ParamMCO( entropia, sma, contraste );
         }
      }
   }

   private JScrollPane imagemLabel( String nomeImagem ) {
      JPanel imagem1 = new JPanel();
      JLabel nome = new JLabel( nomeImagem );
      // Cria DisplayJAI
      DisplayJAI img1 = new DisplayJAI();
         PlanarImage rec = JAI.create("fileload", nomeImagem );
         ParameterBlock pb = new ParameterBlock();
         pb.addSource(rec);
         pb.add(0.08f);
         pb.add(0.08f);
         pb.add(0.0F);
         pb.add(0.0F);
```

```
        PlanarImage imRed = JAI.create("scale", pb);
        img1.set(imRed);
    imagem1.setLayout( new BorderLayout() );
    imagem1.add( nome, BorderLayout.NORTH );
    imagem1.add( img1, BorderLayout.CENTER );
    return new JScrollPane( imagem1 );
}

/**
 * Método que obtem um vetor de double que representa a similaridade
 * entre a imagem de referência e as demais imagens do banco (extrai
 * o vetor de caracteríticas de todas as imagens do banco e calcula a
 * similaridade com a imagens de referência)
 */
public double[] calculaTodasSimilaridades( CharVector vc1 ) {
    double[] simil = new double[ tamanhoIm ];
    for( indiceIm = 0; indiceIm < tamanhoIm; indiceIm++ ) {
        CharVector vc2 = getVetorDaImagemAtual();
        simil[ indiceIm ] = vc1.computeSimilarity( vc2 );
    }
    return simil;
}

/**
 * Retorna o vetor de características da imagem Atual, ou seja,
 * aquela que está sendo representada pelo indice da imagem (indiceIm)
 */
private CharVector getVetorDaImagemAtual() {
    ConnectionManager.ConnectDB();

    CharVector cv = new CharVector();
    int numExtratores = 0;
    String nome;
    ImageCaractManager ic = new ImageCaractManager();
    ic.setCodImagem( imagem_codigos[ indiceIm ] );
    for( indiceCar = 0; indiceCar < tamanhoCar; indiceCar++ )
        if( caract_selecionadas[ indiceCar ] != -1 ) {
            ic.setCodCaract( caract_codigos[ indiceCar ] );
            nome = caract_nomes[ indiceCar ];
            /**
             * Se a imagem já tem suas carterísticas extraídas e
             * armazenadas na tabela imagem_caract, recupera os dados,
             * caso contrário,tira o fundo da imagem, passa a imagem sem
             * fundo para a extração das características selecionadas
             * armazena o valor (computeValue) na tabela, seguindo o
             * código da imagem e o código da característica.
             */
            try {
                ResultSet rs = ic.findlike();
                if( rs.next() ){
                    ImageCaractManager resultado =
                                    ImageCaractManager.SetObject( rs );
                    cv.setValue( nome, resultado.getValor() );
                } else {
                    // Não encontrou no banco
                    TiraFundo tf = new TiraFundo( imagem_paths[indiceIm] );
                    PlanarImage im = JAI.create("fileload","sfundo.tiff" );

                    // Tratamento diferenciado para cada característica!!!
                    // Seta o campo valor do objeto ic com o resultado da
                    // extração da característica
```

```
                    if( nome.equals( "Área" ) ) {
                       ExtraiArea ea = new ExtraiArea( im );
                       ic.setValor( ea.computeValue() );
                    }
                    else if ( nome.equals( "Densidade" ) ) {
                       ExtraiDensidade ed = new ExtraiDensidade( im );
                       ic.setValor( ed.computeValue() );
                    }
                    else if ( nome.equals( "Forma" ) ) {
                       ExtraiForma ef = new ExtraiForma( im );
                       ic.setValor( ef.computeValue() );
                    }
                    else if ( nome.equals( "Entropia" ) ) {
                       if( parametros == null )
                          parametros = new ParamMCO();
                       ExtraiEntropia ee = new ExtraiEntropia( im,
                                       parametros.getAnguloEntropia(),
                                       parametros.getDistanciaEntropia(),
                                       parametros.getSinalEntropia() );
                       ic.setValor( ee.computeValue() );
                    }
                    else if ( nome.equals( "Segundo Momento Angular" ) ) {
                       ExtraiSMA esma = new ExtraiSMA( im,
                                          parametros.getAnguloSMA(),
                                          parametros.getDistanciaSMA(),
                                          parametros.getSinalSMA() );
                       ic.setValor( esma.computeValue() );
                    }
                    else if ( nome.equals( "Contraste" ) ) {
                       ExtraiContraste ec = new ExtraiContraste( im,
                                       parametros.getAnguloContraste(),
                                       parametros.getDistanciaContraste(),
                                       parametros.getSinalContraste() );
                       ic.setValor( ec.computeValue() );
                    }
                    // Grava o valor no banco
                    ic.save();
                    // adiciona característica e valor no vetor de
                    // características
                    cv.setValue( nome, ic.getValor() );
                 }
              } catch( SQLException e ) {
                 System.out.println( "Erro: " + e.getErrorCode() + " msg: "
                                     + e.getMessage());
                 return null;
              }
           }
        ConnectionManager.CloseDB();
        return cv;
     }

     public static void main( String args[] ) {
        Principal application = new Principal();
        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
     }
}
```

ANEXO A – ESTRUTURA DE CLASSES GENÉRICA PARA CBIR

```
/*
 * Created on 04/07/2006
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
package extractor;

import java.util.Vector;

/** <P>Os objetos dessa classe funcionam como um vetor de elementos que
 * representam os parâmetros a serem utilizados por um extrator.
 * Dessa forma, a interface de definição dos parâmetros dentro da
 * classe AbstractExtrator é sempre a mesma, independentemente do
 * número e do tipo dos parâmetros</P>
 *
 * <P> O programador pode adicionar os valores que desejam no
 * ParameterBlock e recuperá-los, de acordo com a ordem em que foram
 * inseridos. Caso sejam adicionados n parâmetros, eles devem ser
 * recuperados pelos seus números de 0 a n-1.
 * </P>
 * @author delamaro
 *
 */
public class ParameterBlock {


   private Vector parameters;

   /** Ãšnico construtor. Cria um objeto sem nenhum valor nele inserido */
   public ParameterBlock()
   {
      parameters = new Vector();
   }

   /** Adiciona um objeto ao ParameterBlock
    *
    * @param o - objeto a ser adicionado
    */
   public void addParameter(Object o)
   {
      parameters.add(o);
   }

   /** Adiciona um inteiro ao ParameterBlock
    *
    * @param i - valor inteiro a ser adicionado
    */
   public void addParameter(int i)
   {
      parameters.add(new Integer(i));
   }

   /** Adiciona um long ao ParameterBlock
    *
    * @param l - valor long a ser adicionado
```

```
 */
public void addParameter(long l)
{
   parameters.add(new Long(l));
}

/** Adiciona um float ao ParameterBlock
 *
 * @param f - valor float a ser adicionado
 */
public void addParameter(float f)
{
   parameters.add(new Float(f));
}

/** Adiciona um double ao ParameterBlock
 *
 * @param d - valor double a ser adicionado
 */
public void addParameter(double d)
{
   parameters.add(new Double(d));
}

/** Adiciona um booleano ao ParameterBlock
 *
 * @param b - valor booleano a ser adicionado
 */
public void addParameter(boolean b)
{
   parameters.add(new Boolean(b));
}

/** Adiciona um byte ao ParameterBlock
 *
 * @param b - valor byte a ser adicionado
 */
public void addParameter(byte b)
{
   parameters.add(new Byte(b));
}

/** Adiciona um char ao ParameterBlock
 *
 * @param c - valor char a ser adicionado
 */
public void addParameter(char c)
{
   parameters.add(new Character(c));
}

/** Adiciona um short ao ParameterBlock
 *
 * @param s - valor short a ser adicionado
 */
public void addParameter(short s)
{
   parameters.add(new Short(s));
}

/** Obtem o valor de um objeto que foi inserido ao ParameterBlock
```

```
 *
 * @param i - ordem do valor a ser recuperado, dentro do ParameterBlock
 * @return o valor do parametro armazenado.
 * @throws ArrayIndexOutOfBoundsException - caso não exista o
 * parâmetro requisitado pelo argumento de entrada i.
 */
public Object getParameter(int i) throws ArrayIndexOutOfBoundsException
{
   Object o = parameters.get(i);
   return o;
}

/** Obtem o valor de um inteiro que foi inserido ao ParameterBlock
 *
 * @param i - ordem do valor a ser recuperado, dentro do ParameterBlock
 * @return o valor do parametro armazenado.
 * @throws ArrayIndexOutOfBoundsException - caso não exista o
 * parâmetro requisitado pelo argumento de entrada i.
 * @throws ClassCastException - caso o argumento armazenado não seja
 * do tipo requisitado
 */
public int getIntParameter(int i)
   throws ArrayIndexOutOfBoundsException, ClassCastException
{
   Integer o = (Integer) parameters.get(i);
   return o.intValue();
}

/** Obtem o valor de um long que foi inserido ao ParameterBlock
 *
 * @param i - ordem do valor a ser recuperado, dentro do ParameterBlock
 * @return o valor do parametro armazenado.
 * @throws ArrayIndexOutOfBoundsException - caso não exista o
 * parâmetro requisitado pelo argumento de entrada i.
 * @throws ClassCastException - caso o argumento armazenado não seja
 * do tipo requisitado
 */
public long getLongParameter(int i)
   throws ArrayIndexOutOfBoundsException, ClassCastException
{
   Long o = (Long) parameters.get(i);
   return o.longValue();
}

/** Obtem o valor de um float que foi inserido ao ParameterBlock
 *
 * @param i - ordem do valor a ser recuperado, dentro do ParameterBlock
 * @return o valor do parametro armazenado.
 * @throws ArrayIndexOutOfBoundsException - caso não exista o
 * parâmetro requisitado pelo argumento de entrada i.
 * @throws ClassCastException - caso o argumento armazenado não seja
 * do tipo requisitado
 */
public float getFloatParameter(int i)
   throws ArrayIndexOutOfBoundsException, ClassCastException
{
   Float o = (Float) parameters.get(i);
   return o.floatValue();
}
```

```
/** Obtem o valor de um double que foi inserido ao ParameterBlock
 *
 * @param i - ordem do valor a ser recuperado, dentro do ParameterBlock
 * @return o valor do parametro armazenado.
 * @throws ArrayIndexOutOfBoundsException - caso não exista o
 * parâmetro requisitado pelo argumento de entrada i.
 * @throws ClassCastException - caso o argumento armazenado não seja
 * do tipo requisitado
 */
public double getDoubleParameter(int i)
   throws ArrayIndexOutOfBoundsException, ClassCastException
{
   Double o = (Double) parameters.get(i);
   return o.doubleValue();
}

/** Obtem o valor de um booleano que foi inserido ao ParameterBlock
 *
 * @param i - ordem do valor a ser recuperado, dentro do ParameterBlock
 * @return o valor do parametro armazenado.
 * @throws ArrayIndexOutOfBoundsException - caso não exista o
 * parâmetro requisitado pelo argumento de entrada i.
 * @throws ClassCastException - caso o argumento armazenado não seja
 * do tipo requisitado
 */
public boolean getBooleanParameter(int i)
   throws ArrayIndexOutOfBoundsException, ClassCastException
{
   Boolean o = (Boolean) parameters.get(i);
   return o.booleanValue();
}

/** Obtem o valor de um char que foi inserido ao ParameterBlock
 *
 * @param i - ordem do valor a ser recuperado, dentro do ParameterBlock
 * @return o valor do parametro armazenado.
 * @throws ArrayIndexOutOfBoundsException - caso não exista o
 * parâmetro requisitado pelo argumento de entrada i.
 * @throws ClassCastException - caso o argumento armazenado não seja
 * do tipo requisitado
 */
public char getCharParameter(int i)
   throws ArrayIndexOutOfBoundsException, ClassCastException
{
   Character o = (Character) parameters.get(i);
   return o.charValue();
}

/** Obtem o valor de um byte que foi inserido ao ParameterBlock
 *
 * @param i - ordem do valor a ser recuperado, dentro do ParameterBlock
 * @return o valor do parametro armazenado.
 * @throws ArrayIndexOutOfBoundsException - caso não exista o
 * parâmetro requisitado pelo argumento de entrada i.
 * @throws ClassCastException - caso o argumento armazenado não seja
 * do tipo requisitado
 */
public byte getByteParameter(int i)
   throws ArrayIndexOutOfBoundsException, ClassCastException
{
   Byte o = (Byte) parameters.get(i);
```

```
        return o.byteValue();
    }

    /** Obtem o valor de um short que foi inserido ao ParameterBlock
     *
     * @param i - ordem do valor a ser recuperado, dentro do ParameterBlock
     * @return o valor do parametro armazenado.
     * @throws ArrayIndexOutOfBoundsException - caso não exista o
     * parâmetro requisitado pelo argumento de entrada i.
     * @throws ClassCastException - caso o argumento armazenado não seja
     * do tipo requisitado
     */
    public short getShortParameter(int i)
        throws ArrayIndexOutOfBoundsException, ClassCastException
    {
        Short o = (Short) parameters.get(i);
        return o.shortValue();
    }

    /** Diz quantos parâmetros existem no ParameterBlock.
     *
     * @return - número de elementos presentos no objeto.
     */
    public int getParameterCount()
    {
        return parameters.size();
    }

}




/*
 * Created on 05/07/2006
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
package extractor;

import javax.media.jai.PlanarImage;


/**<P>
 * Essa classe implementa as funcionalidades de um "extrator" de
 * características. É uma classe abstrata que serve de base para as
 * implementações dos extratores reais. Na verdade, o extrator devers er
 * capaz de calcular o valor da diferença entre duas imagens. </P>
 *
 * <P> Para tanto, a classe possui um construtor que recebe como
 * parâmetros uma imagem e um método que permite que essa imagem seja
 * recuperada. Além dessas, são definidos dois métodos abstratos, um
 * que permite que os parâmetros do extrator sejam definidos e outra que
 * faz a comparação entre duas imagens.
 * </P>
 *
 * @author delamaro
 *
 */
public abstract class AbstractExtractor  {
```

```
PlanarImage image;


    /** Construtor. Recebe como parâmetro uma imagem, de onde as
     * características serão obtidas.
     * @param p - imagem da qual as características serão extraídas.
     */
    public AbstractExtractor(PlanarImage p)
    {
       image = p;
    }

    /** Obriga subclasse a definir um construtor
     *
     *
     */
    private AbstractExtractor()
    {

    }

    /** Obtém a imagem com a qual o objeto foi criado.
     *
     * @return imagem relacionada ao extrator
     */
    public PlanarImage getImage()
    {
       return image;
    }

    /** Método abstrato que faz a comparação entre a imagem associada ao
     * objeto e uma outra, associada a outro extrator, em geral, da mesma
     * classe.
     *
     * @param x - segundo extrator, do qual é utilizada a imagem ou
     * qualquer medida que permita calcular a diferença entre as imagens.
     *
     * @return valor double, que é a diferença entre as medidas das duas
     * imagens. Em geral, é um valor no intervalo [0, 1]
     */
    abstract public double compare(AbstractExtractor x);

    /** Método abstrato que retorna um valor associado à  característica
     * que a classe implementa. Note que um extrator pode implementar ou o
     * método compare ou o método computeValue (ou ambos). No caso do
     * primeiro, não existe necessariamente um valor associado à
     * característica mas sim, o valor da diferença entre duas imagens. O
     * programador deve saber qual é o tipo do extrator para poder usá-lo
     * corretamente.
     *
     * @return valor double representa o valor da característica para a
     * imagem. Em geral, deve ser um valor no intervalo [0,1].
     */

    abstract public double computeValue();

    /** Permite que os parâmetros de comparação ou de extração das
     * características sejam definidos para o extrator. Todos os
     * parâmetros necessários devem ser agrupados num objeto do tipo
     * ParameterBlock.
     *
```

```
     * @param pb - objeto que agrupa todos os parâmetros a serem usados por
     * esse extrator.
     */
    abstract public void setParameters(ParameterBlock pb);
}



/*
 * Created on 05/07/2006
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
package similarity;

import java.lang.reflect.Constructor;
import java.util.Hashtable;
import java.util.Iterator;

import javax.media.jai.PlanarImage;

import extractor.AbstractExtractor;
import extractor.ParameterBlock;

/** <P>Essa classe permite que diversos extratores sejam combinados para
 * formar uma função de similaridade, entre duas imagens.</P>
 * <P> O programador pode definir quais extratores devem ser usados,
 * adicionando-os ao objeto através do método addExtractor. Para esse
 * método devem ser fornecidos um nome e o nome completo da classe que
 * implementa esse extrator. A classe deve estar acessível a partir do
 * classpath corrente</P>
 * <P>Uma vez adicionados os extratores, seus parâmetros podem ser
 * definidos através do método setParameters. O valor da função de
 * similaridade é a soma dos valores devolvidos pelo método compara(), de
 * cada um dos estratores.</P>
 *
 * @author delamaro
 *
 */
public class SimilarityFunction {

    private static final Class[]
             EXTRACTOR_CONSTRUCTOR_PARAMS = new Class[] {PlanarImage.class};

    private PlanarImage image1, image2;
    private Hashtable extractors, extractors2;

    /** Para criar um objeto dessa classe devem ser fornecidas as duas
     * imagens a serem comparadas.
     *
     * @param p1 - primeira imagem a ser comparada.
     * @param p2 - segunda imagem a ser comparada.
     */
    public SimilarityFunction(PlanarImage p1, PlanarImage p2)
    {
        image1 = p1;
        image2 = p2;
        extractors = new Hashtable();
        extractors2 = new Hashtable();
```

```
}

/** Adiciona um extrator à  função de similaridade.
 *
 * @param name - nome arbitrário, dado pelo programador e que serve
 * para identificar o extrator dentro do objeto. Através desse nome o
 * programador pode adicionar parâmetros ao extrator.
 * @param extractorClass - nome completo da classe que implementa o
 * extrator.
 * Tal classe deve estar acessível no classpath corrente da
 * aplicação.
 * @throws ClassNotFoundException - caso a classe que implementa o
 * extrator não seja encontrada
 * @throws IllegalArgumentException - caso o nome do extrator já esteja
 * em uso ou caso o extrator não possa ser instanciado.
 * @throws SecurityException - operação não permitida
 * @throws NoSuchMethodException - extrator não possui construtor
 * adequado para instanciação.
 */
public void addExtractor(String name, String extractorClass)
    throws ClassNotFoundException, IllegalArgumentException,
           SecurityException, NoSuchMethodException
{
    if (extractors.containsKey(name) )
    {
        throw new
        IllegalArgumentException("Extractor already defined: " + name);
    }
    Class exclass = Class.forName(extractorClass);
    Constructor ct =
              exclass.getConstructor(EXTRACTOR_CONSTRUCTOR_PARAMS);
    AbstractExtractor ex = null, ex2 = null;
    try {
        ex = (AbstractExtractor) ct.newInstance(new Object[] {image1});
        ex2 = (AbstractExtractor) ct.newInstance(new Object[] {image2});
    } catch (Exception e) {
        throw new
            IllegalArgumentException("Cannot instantiate extractor.");
    }

    extractors.put(name, ex);
    extractors2.put(name, ex2);
}

/** Permite que sejam definidos os parâmetros de um dado estrator. O
 * extrator deve ter sido adicionado anteriormente usando-se o método
 * addExtractor.
 *
 * @param name - nome dado ao extrator, quando este foi adicionado
 * @param pb - conjunto de parâmetros a ser utilizado.
 */
public void setParameters(String name, ParameterBlock pb)
{
    AbstractExtractor ex1 = (AbstractExtractor) extractors.get(name);
    AbstractExtractor ex2 = (AbstractExtractor) extractors2.get(name);
    ex1.setParameters(pb);
    ex2.setParameters(pb);
}

/** Calcula o valor da função de similaridade para as imagens.
 * Atualmente
```

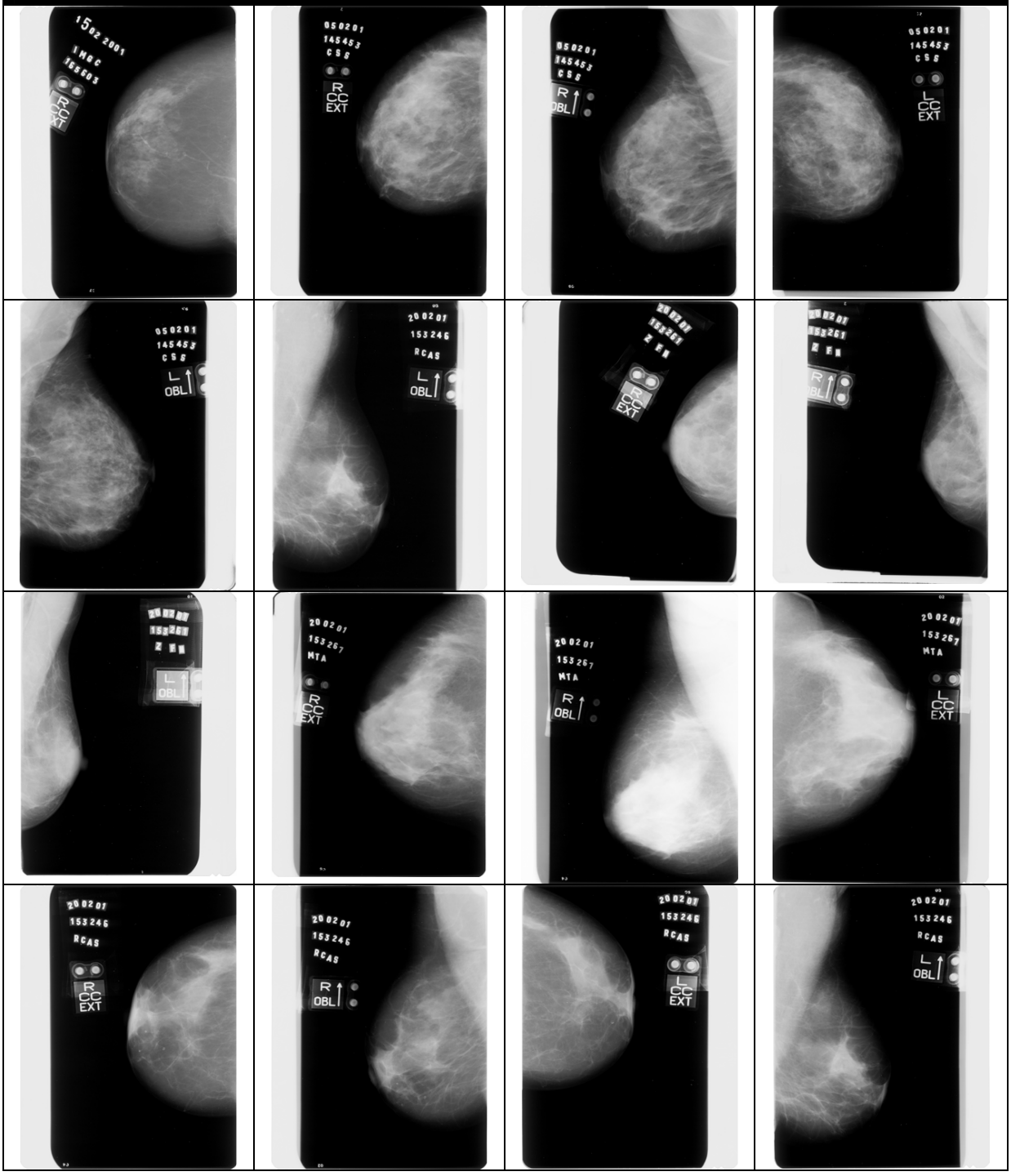
```
     * é calculado através da soma dos valores devolvidos pelo método
     * compara de cada um dos extratores.
     *
     * @return valor da função de similaridade calculado.
     */
    public double computeSimilarity()
    {
       double d = 0;
       Iterator it = extractors.keySet().iterator();
       while (it.hasNext())
       {
          String name = (String) it.next();
          AbstractExtractor ex1 = (AbstractExtractor) extractors.get(name);
          AbstractExtractor ex2 = (AbstractExtractor)extractors2.get(name);
          d += ex1.compare(ex2);
       }
       return d;
    }

}




/*
 * Created on 09/10/2006
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
package similarity;

import java.util.Hashtable;
import java.util.Iterator;
import java.util.Map;

public class CharVector {

    private Hashtable vector;

    public CharVector()
    {
       vector = new Hashtable();
    }

    /* Adiciona uma característica e seu respectivo valor.
     *
     * @param name - nome da característica
     * @param v - valor associado à  característica
     */
    public void setValue(String name, double v)
    {
       vector.put(name, new Double(v));
    }

    /**
     * Obtem o valor de uma das características.
     *
     * @param name - nome da característica
     * @return - valor associado à  caracterÅ›itica
     * @throws - IllegalArgumentException caso a característica não tenha
```

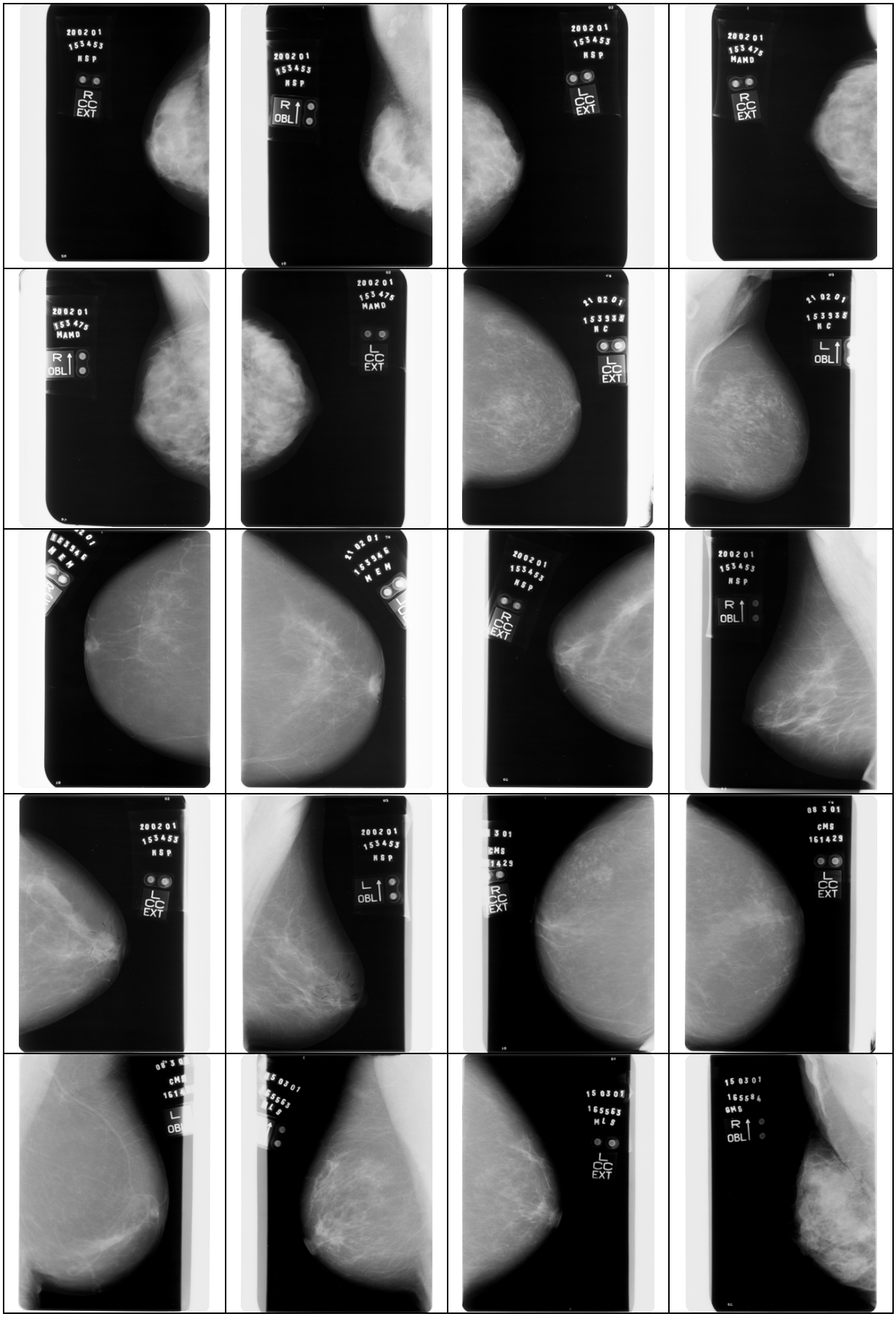
```
     * sido definida.
     */
    public double getValue(String name) throws IllegalArgumentException
    {
        Double d = (Double) vector.get(name);
        if ( d == null ) {
            throw new IllegalArgumentException("Characteristic not found.");
            //System.exit( 0 );
        }
        return d.doubleValue();
    }

    /**
     * Computa a função de similaridade baseado na características desse
     * objeto e do outro passado como argumento. No caso de algum vetor de
     * característica não possui as mesmas características do outro, as
     * características extras são simplesmente ignoradas. Ou seja, somente
     * as características comuns são consideradas.
     * @param v2 - segundo vetor de características usado na comparação
     * @return retorna a distânca Euclidiana dos valores dos dois vetores.
     */
    public double computeSimilarity(CharVector v2)
    {
        Map mp = v2.getVector();
        double distance = 0.0;
        Iterator it = mp.keySet().iterator();
        while ( it.hasNext() )
        {
            String name = (String) it.next();
            if ( ! vector.containsKey(name) )
            {
                continue;
            }
            double d1 = ((Double) vector.get(name)).doubleValue();
            double d2 = ((Double) mp.get(name)).doubleValue();
            distance += (d1-d2) * (d1-d2);
        }
        return Math.sqrt(distance);
    }


    /**
     * Computa a função de similaridade baseado na características desse
     * objeto e do outro passado como argumento. No caso de algum vetor de
     * característica não possuir as mesmas características do outro, uma
     * excceção é lançada
     * @param v2 - segundo vetor de características usado na comparação
     * @return retorna a distânca Euclidiana dos valores dos dois vetores.
     * @throws IllegalArgumentException
     */
    public double computeStrictySimilarity(CharVector v2)
        throws IllegalArgumentException
    {

        Map mp = v2.getVector();
        if ( mp.size() != vector.size() )
            throw   new   IllegalArgumentException("Charactristics   do   not
match.");
        double distance = 0.0;
        Iterator it = mp.keySet().iterator();
        while ( it.hasNext() )
```

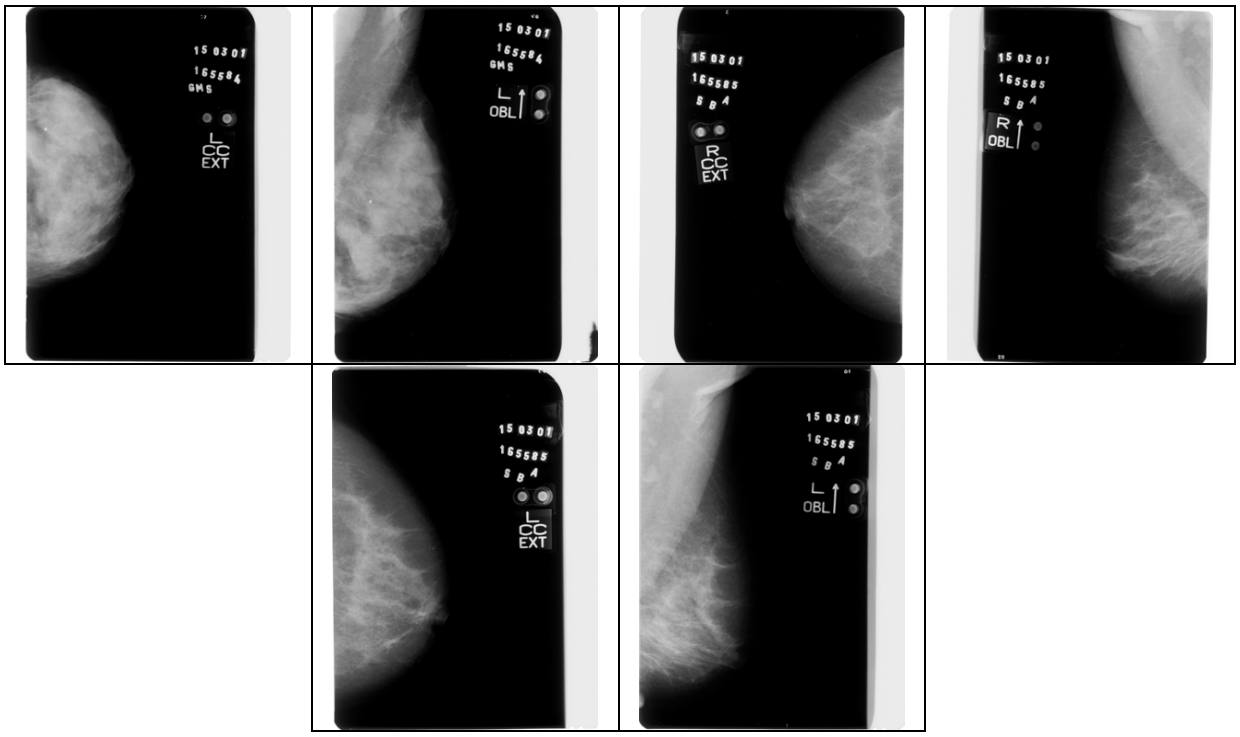
```
        {
            String name = (String) it.next();
            if ( ! vector.containsKey(name) )
            {
                throw new
                    IllegalArgumentException("Charactristics do not match.");
            }
            double d1 = ((Double) vector.get(name)).doubleValue();
            double d2 = ((Double) mp.get(name)).doubleValue();
            distance += (d1-d2) * (d1-d2);
        }
        return Math.sqrt(distance);
    }


    /** Retorna o vetor de características deste objeto, na forma de uma
     * Hashtable
     *
     * @return
     */
    public Map getVector()
    {
        return vector;
    }
}
```

ANEXO B – CONJUNTOS DE IMAGENS UTILIZADAS PARA OS TESTES DO SISTEMA

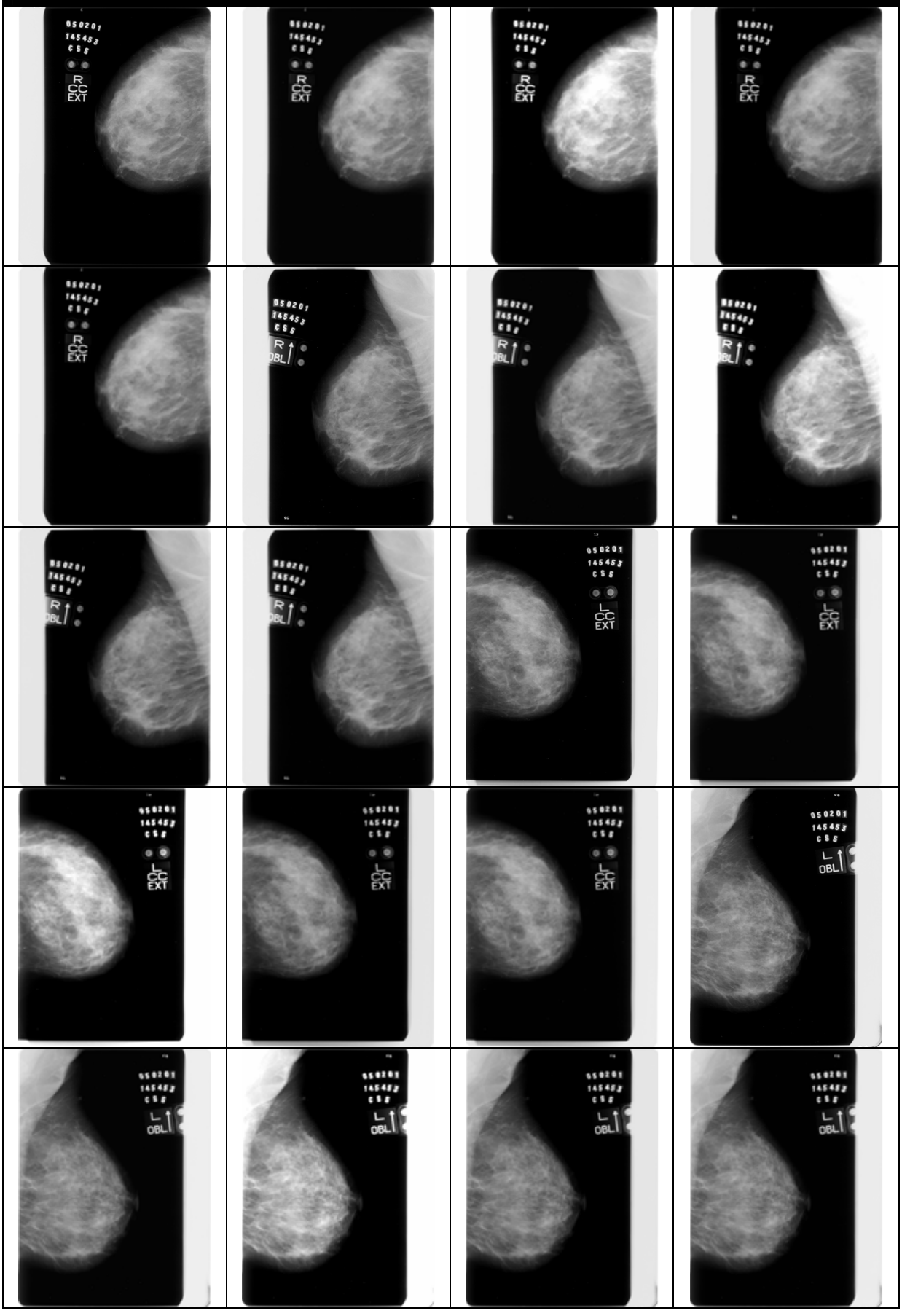
Imagens do Primeiro Conjunto




R CC EXT
R OBL
L CC EXT
L OBL


15 03 01
165584
GMS
L
CC
EXT
15 03 01
165584
GMS
L
OBL
15 03 01
165585
S B A
R
CC
EXT
15 03 01
165585
S B A
R
OBL
15 03 01
165585
S B A
L
CC
EXT
15 03 01
165585
S B A
L
OBL

Imagens do Segundo Conjunto



Imagens do Terceiro Conjunto