UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

BUSCA DE IMAGENS BASEADA NO CONTEÚDO:

APLICAÇÃO EM IMAGENS MÉDICAS

RAFAEL GESSELE

BLUMENAU
2007

2007/1-38

RAFAEL GESSELE

BUSCA DE IMAGENS BASEADA NO CONTEÚDO:

APLICAÇÃO EM IMAGENS MÉDICAS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Mauro Marcelo Mattos, Dr. - Orientador

BLUMENAU
2007

2007/1-38

BUSCA DE IMAGENS BASEADA NO CONTEÚDO:

APLICAÇÃO EM IMAGENS MÉDICAS

Por

RAFAEL GESSELE

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Mauro Marcelo Mattos, Dr. – Orientador, FURB
Membro:	Prof. Paulo César Rodacki Gomes, Dr. – FURB
Membro:	Prof. Jomi Fred Hübner, Dr. – FURB

Blumenau, julho 2007

Dedico este trabalho a meus pais e a minha família, pelas palavras de apóio e pela atenção a mim dedicada.

AGRADECIMENTOS

A Deus, por sempre iluminar meu caminho.

A meus pais pelo amor e confiança em mim depositados.

À minha família, pelos sábios conselhos.

Aos meus amigos, pela compreensão nos momentos de ausência.

Ao meu orientador, por ter acreditado na conclusão deste trabalho.

Ao Alex Sandro da Silva, por ter fornecido as imagens para a base de testes.

A mente que se abre a uma nova idéia, jamais voltará a seu tamanho original.

Albert Einstein

RESUMO

Sistemas de busca de imagens baseados em conteúdo têm como principal característica a habilidade de a partir de uma imagem inicial retornar outras imagens semelhantes. Este trabalho, apoiado em técnicas de visão computacional, demonstra a extração de vetores de características de uma imagem e a utilização deles para comparação entre imagens. A partir de uma imagem chave o sistema desenvolvido extrai os vetores de características de um banco de imagens e efetua uma busca das *n* imagens mais parecidas utilizando um determinado critério. Um vetor de características é uma representação numérica dos detalhes mais significativos de uma determinada imagem e neste trabalho é extraído através de uma aplicação prática de *wavelets* para análise de sinais.

Palavras-chave: *Wavelets*. Vetor de características. Busca por conteúdo.

ABSTRACT

Content-based image retrieval systems have as main characteristic the ability of, from an initial image, searching for other similar ones. This work, based in techniques of computational vision, demonstrates the extraction of feature vectors of an image and the use of them for comparison between images. By giving a query image, the system extracts the feature vectors from an image database and returns the *n* most similar images according to a given criteria. A feature vector is a numerical representation of the most significant details in a determined image, and in this work, it is extracted through a practical application of wavelets for signals analysis.

Key-words: Wavelets. Feature vector. Query by content.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

API - *Application Programming Interface*

CBIR - *Content-Based Image Retrieval*

CWT - *Continues Wavelet Transform*

DICOM - *Digital Imaging and COmmunications in Medicine*

IDE - Integrated Development Environment

JAR - *Java ARchive*

MVC - *Model View Controller*

PACS - *Picture Archiving and Communication Systems*

ROI - Region Of Interest

TC - Tomografia Computadorizada

UH - Unidades de *Hounsfield*

WFT - *Windowed Fourier Transform*

SUMÁRIO

1 INTRODUÇÃO

O dia-a-dia da atividade médica é marcado por uma busca constante de um diagnóstico preciso e da avaliação terapêutica. Para esse fim o médico serve-se de uma grande variedade de técnicas de produção de imagens, entre elas destacando-se os métodos radiológicos (como por exemplo, a Tomografia Computadorizada). Como o volume de imagens gerado em um hospital ou clínica especializada é extremamente grande, o desenvolvimento de um software que auxilie os médicos na busca por imagens para então diagnosticar com maior agilidade e confiança é de extrema valia.

As exigências para modernos sistemas de análise de imagens estão cada vez mais complexas. Isto se torna evidente na área médica, onde imagens com qualidades e conteúdos diferentes necessitam ser analisadas através do uso de complexos conhecimentos acerca dos elementos esperados.

Para atingir o objetivo de reconhecer determinados objetos em imagens, ou imagens similares é necessário submetê-la ao processamento por uma seqüência específica de diferentes processos de análise, que vão desde filtros de intensificação de contraste até a classificação de figuras complexas. A adequação de um determinado método ou processo para a execução de uma determinada tarefa depende fortemente da natureza e dos parâmetros da imagem a ser analisada (SILVA, 2002, p. 1).

Os sistemas de recuperação de imagens baseados em conteúdo também conhecidos por *Content-Based Image Retrieval* (CBIR) estão sendo muito estudados e pesquisados atualmente. Este interesse deve-se principalmente a sua aplicabilidade em diversas áreas do conhecimento (CAMPO, 2002, p. 1). Estas imagens podem conter também informações sobre os pacientes na forma textual. Isto é possível através das imagens no padrão *Digital Imaging and Communications in Medicine* (DICOM) tipicamente utilizado em aplicações médicas (RADIOLOGICAL SOCIETY OF NORTH AMERICA, 2006). Para centralizar e armazenar o grande volume de informações geradas pelos equipamentos médicos alguns centros hospitalares utilizam sistemas *Picture Archiving and Communication Systems* (PACS) desenvolvidos especialmente para esta finalidade. A maioria destes sistemas não possui uma solução de busca de imagens otimizada. A recuperação das imagens é feita de forma tradicional, fazendo a pesquisa baseada em descrições textuais do arquivo que foram previamente cadastradas (CAMPO, 2002, p. 6).

Segundo Campo (2002, p. 1), muitas das consultas de interesse dos médicos visam

procurar imagens de pacientes que tenham semelhança entre si. Mediante isso surge a motivação para o desenvolvimento de um protótipo de busca de imagens baseada no conteúdo.

1.1 OBJETIVOS DO TRABALHO

Este trabalho tem por objetivos estudar a extração de características em imagens médicas, utilizar os dados extraídos em forma de vetores de características para descrever uma imagem e desenvolver um sistema de busca de imagens similares baseado em conteúdo utilizando estes vetores de características.

Dentro deste escopo, o protótipo desenvolvido será capaz de abrir e exibir imagens no formato DICOM, deverá implementar a extração dos vetores de características por meio da transformada *wavelet* e extratores estatísticos, e realizar a comparação das imagens em uma base de dados retornando os resultados em ordem de similaridade.

Para isso foi necessário o estudo de APIs de manipulação de imagens, cálculos estatísticos, manipulação de formato de imagens em 8, 16 ou 32 bits, padrões de projeto para ganhos de performance e outros desafios encontrados durante o desenvolvimento.

1.2 ESTRUTURA DO TRABALHO

Esta monografia divide-se em fundamentação teórica, desenvolvimento do trabalho, e conclusões. No capítulo de fundamentação teórica são mostradas as informações que o autor julga necessárias para o bom entendimento do desenvolvimento deste trabalho e das técnicas em que ele foi baseado. No capítulo 3 é apresentada a metodologia de desenvolvimento e os documentos de análise, assim como os algoritmos criados e adaptados. Neste capítulo também são exibidos alguns resultados e estudos realizados durante o trabalho. Por fim, a conclusão retrata os objetivos alcançados, os não alcançados, e são sugeridos alguns trabalhos de extensão a este projeto.

2 FUNDAMENTAÇÃO TEÓRICA

No presente capítulo são apresentados alguns aspectos teóricos referentes ao trabalho. O tópico 2.1 apresenta uma breve teoria sobre aquisição de imagens médicas. A seção 2.2 aborda uma breve ambientação sobre o padrão de imagens DICOM. O tópico 2.3 apresenta algumas técnicas de processamento de imagens e na seção 2.4 é apresentado detalhes sobre a ferramenta de processamento de imagens utilizada neste trabalho. Por fim, a seção 2.5 traz uma breve teoria sobre *wavelets* e sua aplicação.

2.1 AQUISIÇÃO DE IMAGENS

Neste tópico são abordados alguns aspectos relativos a aquisição de imagens médicas e as formas e condições em que elas são obtidas até serem armazenadas nas bases de dados. Segundo Silva (2004, p. 1-3), as técnicas de aquisição de imagens médicas podem ser divididas em invasivas e não invasivas, de acordo com a forma como são obtidas. Os métodos invasivos caracterizam-se pela introdução de um instrumento no interior do corpo humano, de forma a obter as imagens pretendidas. Nesta categoria incluem-se as angiografias e as imagens de medicina nuclear. Nos métodos não invasivos incluem-se os raios X, ultra-sonografia, tomografia computadorizada e ressonância magnética. Para o presente trabalho a base de imagens utilizadas é na sua maioria de imagens geradas por aparelhos tomografia computadorizada e aparelhos de ressonância magnética.

Os dados volumétricos extraídos desses métodos são geralmente adquiridos na forma de imagens de fatias paralelas uniformemente espaçadas, representando cortes transversais ao eixo longitudinal do paciente. Comumente nas regiões de maior interesse são feitos cortes mais próximos, permitindo uma maior visualização dos dados. A Figura 1 exibe um exemplo da representação 3D de uma imagem de cabeça e a fatia ou *slice* da região de estudo utilizada para formar uma imagem.

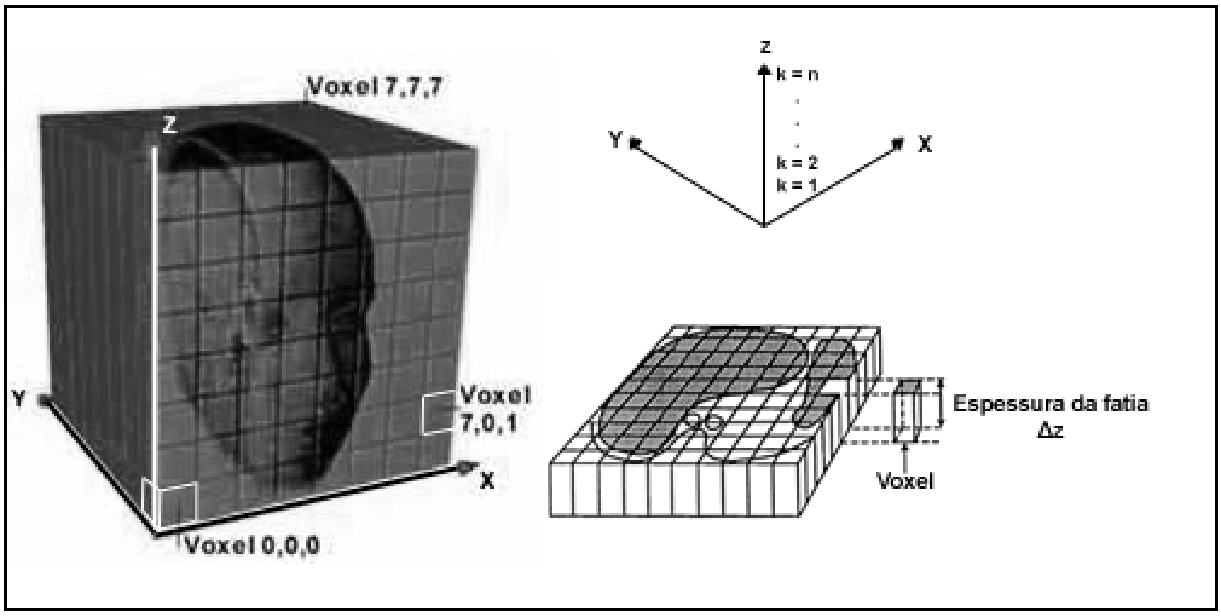


Figura 1 – *Voxel* e *slices*

Cada imagem gerada está associada a uma localização **k, k = 1, 2, ..., n**, no eixo **z** e uma espessura **Δz = e** em torno desta localização, formando um cubóide. O cubóide é subdividido em outros cubóides pequenos chamados *voxels*. O *voxel* é equivalente a *pixel* em 3D e representa uma abreviação para *volume element*. Cada *pixel* da imagem está associado a um *voxel*. O valor associado a cada *pixel* representa a média das atenuações do raio X no volume interno do corpo correspondente ao *voxel*. Os valores destas atenuações são expressos em Unidades de Hounsfield (UH). Tais valores são obtidos pela exposição do corpo ao bombardeamento de raios X em várias direções (SILVA, Aristófanes, 2004, p. 1).

O valor associado a cada *voxel* é um número inteiro, proporcional ao tom de cinza do *pixel* na imagem correspondente, e representa a integração de alguma propriedade física que está sendo mensurada no interior do volume associado ao *voxel*. No caso da tomografia computadorizada, por exemplo, a grandeza física medida é a densidade do tecido. Quanto maior for a densidade do tecido, maiores serão as atenuações e, portanto, maiores serão os valores dos pixels nas imagens dos cortes referentes a este tecido. A seguir são abordados dois métodos não invasivos de aquisição de imagens médicas.

2.1.1 Ressonância magnética

A ressonância magnética é principalmente aplicada a tecidos moles. No interior do

corpo humano, todos os núcleos atômicos possuem um determinado campo magnético, o que significa que eles se comportam como pequenos ímãs. Ao ser colocado dentro de um tubo capaz de gerar um elevado campo magnético, os núcleos se alinham na direção deste campo, vibrando em torno do seu eixo com uma determinada freqüência. Esta freqüência depende fundamentalmente do tipo de núcleo, o que permite distinguir os diversos tipos de tecidos (SILVA, Aristófanes, 2004, p. 3).

2.1.2 Tomografia computadorizada

A Tomografia, derivada da palavra grega “Tomos”, que significa corte ou fatia, e “Grafos”, que significa desenhar uma imagem ou gráfico, emprega os mesmos princípios da radiografia convencional com o objetivo de criar uma representação anatômica baseada na quantidade de atenuação sofrida pela radiação incidente (SILVA, Aristófanes, 2004, p. 3).

O nome Tomografia Computadorizada (TC) deve-se ao fato dessa técnica ser altamente dependente de computadores para realizar os cálculos matemáticos relativamente complexos referentes às informações coletadas durante a emissão e rotação dos raios X. Na TC, o feixe de raios X que atravessa o corpo é muito colimado[1] e fino, reduzindo a produção de raios secundários que degradariam a imagem. Diferentemente do estudo radiológico convencional, os raios X não impressionam filmes após atravessarem o corpo, mas são captados por detectores de fótons e as medidas de atenuação tissular são calculadas e armazenadas no computador. Tais mensurações são feitas em Unidades de Hounsfield (UH). Assim, o ar contido nas vias respiratórias e no tubo digestivo tem valores mais negativos, como -800 UH ou -1000 UH, e os ossos, os mais positivos, tais como 400 UH ou 500 UH. A água é usada para a calibração do equipamento e seus valores de atenuação estão entre 0 e ±10. A imagem obtida com equipamentos de TC é o resultado da disposição na tela do monitor de uma enorme quantidade de números lado a lado e em linhas, que representam coeficientes de atenuação tissular, produtos de cálculos efetuados pelo computador enquanto o feixe de raios X atravessa a área estudada. Cada valor numérico corresponde a uma tonalidade em escala de cinza, que vai do preto ao branco. As áreas mais escuras indicam menor densidade e as mais claras indicam maior densidade (SILVA, Aristófanes, 2004, p. 4).

[1] Raios paralelos a partir de um foco luminoso.

2.2 IMAGENS DICOM

Segundo a *Radiological Society of North America* (2006), o padrão DICOM é uma especificação detalhada que descreve um meio de formatar e trocar imagens juntamente com informações associadas entre computadores. Essa especificação relaciona ligações de redes normatizadas e dispositivos de armazenamento responsáveis pela comunicação de arquivos de imagens digitais, provenientes de tomografia computadorizada, ressonância magnética, medicina nuclear, ultra-sonografia, raios X, etc.

A especificação do padrão DICOM 3.0 encontra-se dividida em partes que podem expandir-se individualmente sem haver necessidade de reeditar todo o padrão. Ao todo a especificação esta dividida em 13 partes. Define itens de hardware e uma interface comum entre dispositivos de imagens médicas e o próprio padrão DICOM. Este padrão possui um cabeçalho de informações sobre o paciente, dados sobre o dispositivo utilizado na captura das imagens e a imagem em si. O sistema DICOM permite que informações sobre um paciente sejam difundidas e compartilhadas na rede reduzindo custos com o envio destas imagens fisicamente. Imagens DICOM não perdem a definição e, conseqüentemente, a interpretação das imagens pelas entidades médicas é mantida, já que a qualidade gráfica não se altera (SILVA, Aristófanes, 2004, p. 35).

As imagens DICOM como as que foram utilizadas neste trabalho podem ser convertidas em formatos de 8, 16 ou 32 bits. Especificamente para este projeto as imagens processadas para a extração de características são convertidas para escala de cinza em 8 bits para desta forma ter valores de *pixel* em seu histograma variando entre 0 e 255.

2.3 PROCESSAMENTO DE IMAGENS

Uma imagem digital é uma matriz bidimensional onde seus elementos são chamados *pixels*. O processo de exibir uma imagem cria uma representação gráfica desta matriz onde os valores dos *pixels* assumem um determinado nível de cinza (monocromático) ou uma determinada cor (JORDÁN; LOTUFO, 1999). O processamento de imagens é uma área de estudos da computação que tem evoluído muito nos últimos anos. Seu grande crescimento se deve principalmente ao desenvolvimento de equipamentos, cada vez mais sofisticados e

baratos, utilizados para a captura e tratamento de imagens digitais (HARALICK, 1983 apud SILVA, 1999, p. 10). No trabalho desenvolvido é aplicada uma seqüência de algoritmos de processamento de imagens na amostra[2] de entrada para permitir a extração das informações pertinentes a ela.

Segundo Hoppe (2005, p. 23), o pré-processamento visa melhorar a qualidade de uma imagem digital com a finalidade de facilitar o processamento subseqüente da análise. Dentro de um sistema de reconhecimento de imagens o pré-processamento exerce, principalmente, as funções de realce e restauração das imagens a serem processadas. No caso deste trabalho a pré-processamento é muito importante pois a utilização da transformada *wavelet* como técnica base para a extração de características prepara as imagens para posterior aplicação dos extratores estatísticos nos subespaços gerados.

Para este trabalho foi necessário aplicar uma normalização nas imagens de entrada na tentativa de igualar imagens com contraste ou pequenos detalhes diferentes. Assim a técnica escolhida é conhecida como *threshold*. Facon (1993, p. 20) ressalta que a maioria das técnicas de realce de imagens são heurísticas e orientadas para aplicações específicas. No caso deste trabalho isso não poderia ser diferente. A aplicação de *threshold* nas imagens antes da aplicação da *wavelet* serve para que seja gerada uma máscara da imagem a ser processada.

Segundo Wangenheim (2007, p. 69), o principio básico da limiarização consiste em separar as regiões de uma imagem quando esta apresenta duas classes (fundo e objeto). A forma mas simples da limiarização consiste na bipartição da imagem, convertendo os *pixels* cujo tom de cinza e maior ou igual a um curto valor **(T)** em brancos e os demais em pretos. Esta operação pode ser descrita como uma técnica de processamento de imagem na qual uma imagem de entrada **f (x,y)** de N níveis de cinza produz uma saída **g(x,y)**, chamado de imagem limiarizada, cujo número de níveis de cinza é menor que N, normalmente **g(x,y)**, apresenta dois níveis de cinza.

Função *threshold:*

$$g(x,y) = \begin{cases} 1 se \ \ f(x,y) >= T \\ 0 se \ f(x,y) < T \end{cases}$$

[2] Imagem que serve de base para a busca das demais.

2.4 FERRAMENTAS DE PROCESSAMETO DE IMAGENS

Durante a definição da seqüência de pré-processamento, utiliza-se um número muito grande de técnicas e algoritmos de processamento de imagens para se chegar ao resultado esperado. A implementação destes algoritmos durante está etapa de estudo não é viável devido à complexidade dos mesmos e o fato desta ser uma etapa de estudos. Desta forma surge a necessidade da utilização de ferramentas de processamento de imagens. Estas ferramentas provêem um número relativamente grande destes algoritmos já implementados. Além disso, estes ambientes fornecem uma interface amigável para aplicação destes algoritmos nas imagens de amostra facilitando assim a definição da seqüência de pré-processamento ideal para a resolução do problema (ACCUSOFT, 2006).

Uma vez definidos os algoritmos pode-se então realizar a implementação dos mesmos para utilização junto à solução aqui proposta. Utilizando estes ambientes o problema a ser resolvido é o foco principal sendo desnecessário o trabalho de implementação em um primeiro momento. Para o presente trabalho foi aprofundado o uso da API ImageJ (RESEARCH SERVICES BRANCH, 2007), que pode ser encontrada no endereço eletrônico `http://rsb.info.nih.gov/ij/index.html`. Esta ferramenta desenvolvida em Java é amplamente difundida na comunidade de visão computacional por ter sua licença livre e possuir uma estrutura de *plugins* bastante flexível. É permitido a qualquer desenvolvedor utilizar sua biblioteca de manipulação de imagens, distribuída em forma de um arquivo JAR, e criar *plugins* para ampliar o leque de funcionalidades desta ferramenta. A Figura 2 exibe um exemplo da interface da ferramenta ImageJ durante a fase de estudos para aplicação da binarização dos arquivos. Além do ImageJ também foram estudadas as ferramentas VisiQuest (ACCUSOFT, 2006) e Khorus (JORDÁN; LOTUFO, 1999). Esta primeira possui uma grande quantidade de algoritmos e recursos para processamento de imagens, porém não é gratuita. Já o Khorus é uma alternativa interessante para desenvolvedores em ambiente Linux.

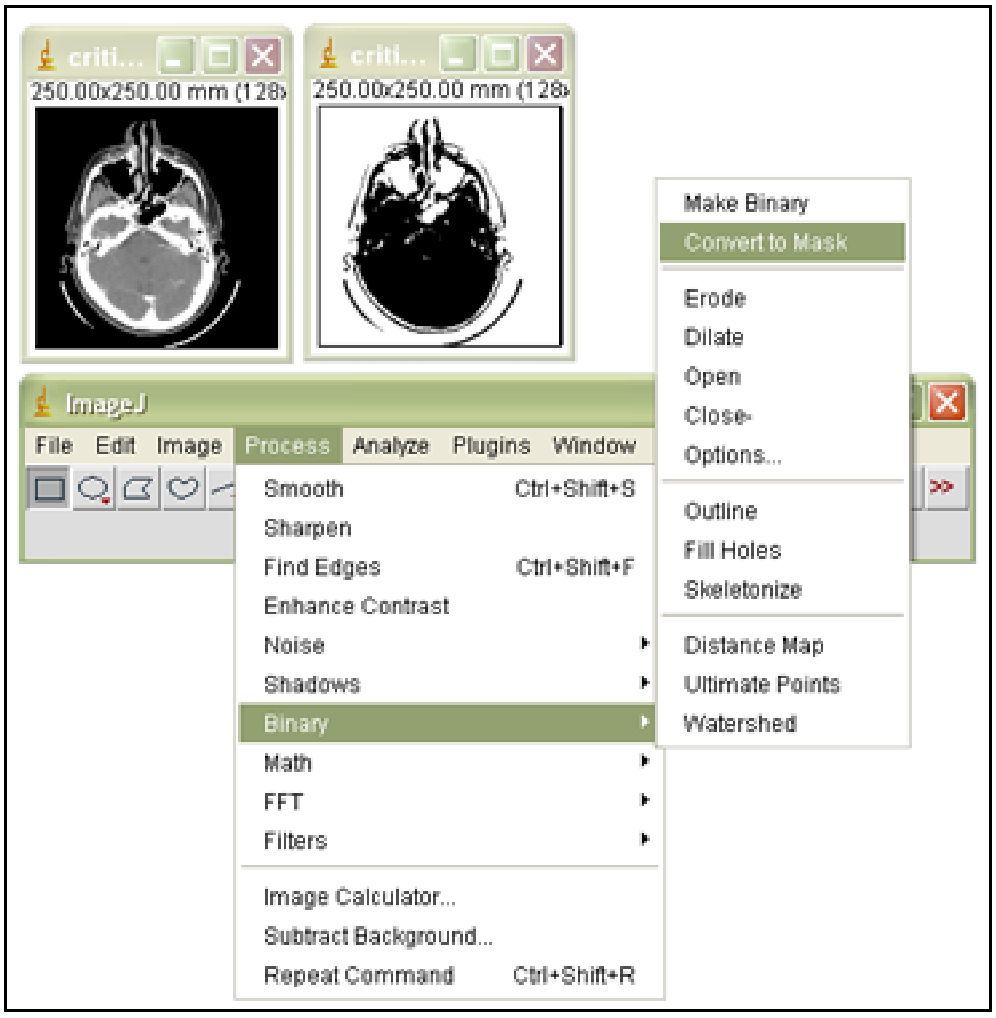


Figura 2 - Interface ImageJ

2.5 *WAVELETS*

Nesta seção será apresentada uma breve ambientação sobre a teoria de *wavelets* uma vez que não é do escopo deste trabalho o detalhamento das propriedades matemáticas que envolvem a transformada. Inicialmente temos um pouco da história e dos estudos que levaram a descoberta e a utilização de *wavelet* para análise de sinais. Após é apresentado um tópico sobre o uso de wavelets na extração de características de imagens e por fim será abordado a aplicação prática que foi utilizada neste trabalho.

2.5.1 Transformada de Fourier

Segundo Castañón(2003, p.26), uma das técnicas mais populares no processamento de sinais é a transformada de Fourier, que realiza a transformação de um sinal (função) do domínio de espaço para o domínio de freqüência. (WANG, 2001 apud CASTAÑÓN, 2003, p. 26). A função da transformada de Fourier é dada pela equação a seguir:

$$F[u] = \int f[t] e^{-i2\pi ut} dt$$

Porém, a equação de Fourier corresponde a um sinal contínuo *f(t)*, e apresenta algumas deficiências. Entre elas está o fato de que esta transformação não está localizada no domínio do espaço, então ela não pode representar adequadamente as mudanças que acontecem no sinal do espaço pois se baseia na integração de toda a função para o cálculo de cada freqüência. Esse não seria um problema se o sinal não mudasse durante o tempo. Porém, muitos sinais apresentam características não estacionárias como fluxos, tendências, mudanças repentinas, início e final de um evento sendo essas variações geralmente a parte mais importante do sinal (CASTAÑÓN, 2003, p. 27).

2.5.2 Transformada por janelas de Fourier

Dado o problema encontrado na transformada de Fourier, Dennis Gabor em 1946 adaptou a transformada de Fourier para analisar só uma porção do sinal em um tempo. Introduziu assim a transformada por janelas de Fourier ou *Windowed Fourier Transform* (WFT). Cada janela de observação é deslocada no domínio do tempo e a transformada de Fourier da porção visível do sinal é calculada para cada posição da janela (CASTAÑÓN, 2003, p. 27). A função que define a WFT é dada pela equação a seguir:

$$F(u,b) = \int j(t-b) f[t] e^{-2\pi ut} dt$$

Assim como Fourier essa transformada também apresenta um ponto negativo. Trata-se do fato de que uma vez definido o tamanho da janela de tempo ela permanece constante para todas as freqüências. Acontece que alguns sinais precisam de um enfoque mais flexível e é deste desafio que trata a transformada *wavelet*.

2.5.3 Transformada *wavelet*

Segundo Castañón (2003, p. 27), a utilização das *wavelets* é o passo lógico seguinte a transformada por janelas de Fourier. Assim, interpreta-se *wavelet* como uma técnica por janelas com regiões de dimensão variável, onde as *wavelets*, diferentemente de Fourier, têm como base uma função de duração limitada, isto é, de suporte compacto, que é uma propriedade na qual seu domínio é diferente de zero em uma extensão finita e igual a zero em todo o resto. Isto torna *wavelets* uma abordagem muito interessante na análise de imagens, pois as mudanças de regiões ou bordas são facilmente detectadas.

A transformada *wavelet* para um sinal contínuo é dada pela fórmula a seguir:

$$F(a,b) = \int f(t)\psi_{a,b}(t)dt$$

A transformada *wavelet* para um sinal discreto é dada pela equação a seguir:

$$F_{m,n}(a,b) = a_0^{-m/2} \int f(t)\psi\left(a_0^{-m}t - nb_0\right)$$

O comportamento destas funções está baseado em dilatações e translações a partir de uma *wavelet* mãe ψ. Observando a equação wavelet para um sinal contínuo percebe-se que a transformada depende de dois parâmetros **a** e **b,** que correspondem as informações de escala e tempo respectivamente.

Para obter os coeficientes de *wavelet* em cada escala possível requer uma grande quantidade de cálculo, tornando muito maçante o trabalho principalmente para a transformada contínua de *wavelets.* Devido a este fato é que a transformada discreta de *wavelets* escolhe um subconjunto de escalas e locações sobre os quais vai realizar os cálculos (CASTAÑÓN, 2003, p. 28).

2.5.4 *Wavelet* e extração de características

Neste tópico será discutido o uso prático da transformada *wavelet* para extração de características em imagens médicas. Além da definição da função *wavelet* adequada ao processamento das imagens, também é importante estabelecer as técnicas de seleção e extração de características que serão utilizadas. Em operações de busca por similaridade o usuário não está interessado nas características de baixo nível das imagens (cor, textura e

forma), isso porque pensa em termos semânticos, ou seja, busca as imagens em categorias que no caso de imagens médicas podem ser raios-X, patologia, grafos, micro-arranjos, etc. Por isso é extramente complicado definir um método adequado de extração de características que tenha a capacidade de diferenciar cada uma das classes semânticas (CASTAÑÓN, 2003, p. 44).

Para o presente trabalho foi levado em consideração a semelhança entre as imagens dos exames radiológicos como forma de classificação dos grupos semânticos. Em outras palavras, a busca realizada a partir de uma imagem inicial deve retornar imagens similares pertencentes ao mesmo grupo semântico, como por exemplo, cortes de abdômen, cabeça, tórax, bacia, etc.

Existe uma variedade de wavelets cada qual com uma finalidade específica. Para o caso de extração de características em imagem as mais conhecidas são a *Continues Wavelet Transform* (CWT) e a *wavelet* de base ortogonal. A primeira é preferencialmente utilizada em análise de sinais, extração e detecção de características. Já as de base ortogonal são mais usadas quando se deseja fazer algum tipo de redução de informação.

Para o presente trabalho foi utilizado *wavelet* conhecida como *fractional splines wavelet* (BIOMEDICAL IMAGING GROUP, 2007). Um dos motivos pelo qual se optou utilizar esta transformada *wavelet* deve-se ao fato de que ela apresenta bons resultados na extração de características em imagens médicas. Além disso, pode ser utilizada na forma de *plugin* para a API de processamento de imagem que foi utilizada neste trabalho.

A *fractional splines wavelet* é uma nova transformada *wavelet* baseada em uma recém definida família de funções de escala conhecida como *fractional B-splines*. O interesse nesta família é que ela interpola entre os graus inteiros das *B-splines* polinomiais e permite uma ordem fracionária de aproximação. As *orthogonal fractional spline wavelets* comportam-se essencialmente como diferenciadores fracionários. Esta propriedade é muito útil para análise de **1/fα,** ou seja, ruído que pode ser atenuado pela escolha correta do parâmetro grau da transformada (BIOMEDICAL IMAGING GROUP, 2007).

O *plugin* utilizado na implementação do protótipo permite a aplicação desta transformada em imagens obtidas através da API ImageJ. Ele pode ser encontrando no seguinte endereço eletrônico: `http://bigwww.epfl.ch/demo/fractsplines/java.html`.

A transformada *B-splines wavelet* utilizada neste trabalho é matematicamente definida segundo o Quadro 1.

B-splines Fracionária:

$$\beta_+^\alpha(x) = \frac{\Delta_+^{\alpha+1} x_+^\alpha}{\Gamma(\alpha+1)}$$

Definições Relacionadas:

Função Gama de Euler

$$\Gamma(u+1) = \int_0^\infty x^u e^{-x} dx$$

Função de potência

$$x_+^a = \begin{cases} x^a, x \geq 0 \\ 0, se\ não \end{cases}$$

Operadores finitos fracionários da diferença

$$\Delta_+^a \xleftarrow{Fourier} \left(1 - e^{-i\omega}\right)^\alpha = \sum_{k \in Z} (-1)^k \binom{\alpha}{k} e^{-j\omega k}$$

$$\Delta_*^a \xleftarrow{Fourier} \left|1 - e^{-i\omega}\right|^\alpha = \sum_{k \in Z} (-1)^k \left|\binom{\alpha}{k}\right| e^{-j\omega k}$$

Binomiais

$$\binom{u}{v} = \frac{\Gamma(u+1)}{\Gamma(v+1)\Gamma(u-v+1)}$$

$$\left|\binom{u}{v}\right| = \binom{u}{v + \frac{u}{2}}$$

Fonte: adaptado de Biomedical Imaging Group (2007).

Quadro 1 – Definição da transformada *wavelet*

3 DESENVOLVIMENTO DO TRABALHO

O desenvolvimento do trabalho dividiu-se em três etapas sendo elas análise, desenvolvimento e testes. Durante este processo foi necessário fazer alguns ajustes no que foi analisado caindo assim num modelo de desenvolvimento em espiral, onde os riscos e custos da implementação eram a todo o momento pesados.

O protótipo foi desenvolvido em JAVA fazendo uso de algumas APIs conforme será explicitado a seguir. Estas APIs foram de uma grande valia pois permitiram a preocupação com o estudo das técnicas de análise para extração de características das imagens e não com processos burocráticos de manipulação das mesmas. Entende-se por processo burocrático por exemplo o suporte a imagens no formato DICOM.

Para a etapa de desenvolvimento foi utilizado a IDE Eclipse (ECLIPSE FOUNDATION, 2007) juntamente com a API de manipulação de imagens e visão computacional ImageJ (RESEARCH SERVICES BRANCH, 2007). Para a extração dos vetores de características e aplicação da transformada *wavelet* foi utilizado o *plugin Fractional Wavelet Tranform* (BIOMEDICAL IMAGING GROUP, 2007).

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O trabalho desenvolvido deve atender aos seguintes requisitos:

a) efetuar o pré-processamento das imagens utilizando os algoritmos de pré-processamento. Realizar os ajustes e normalizações das imagens necessárias antes da extração das características principais;
b) extrair os vetores de características das imagens, ou seja, retirar através do uso de um extrator um conjunto de valores que define a imagem desejada;
c) determinar o grau de similaridade das imagens utilizando a comparação dos vetores de características;
d) possuir uma interface gráfica que permita ao usuário selecionar qual imagem será utilizada como base para a pesquisa;
e) exibir as imagens resultantes da busca na interface gráfica.

Além destes requisitos também foram implementadas outras funcionalidades não previstas em uma primeira análise. Foi implementado o uso de outros extratores para fins de comparação com por exemplo média, entropia, variância, energia e desvio padrão e a geração dos vetores de características em um arquivo *metadata* para o diretório de pesquisa. Com estes vetores previamente calculados o sistema apresenta um melhor desempenho durante a pesquisa por similaridade.

3.2 ESPECIFICAÇÃO

Neste tópico serão apresentados alguns os diagramas utilizados durante a fase de análise e que serviram para descrever e segmentar o problema a ser resolvido em pequenas partes. Durante a fase de implementação algumas dificuldades surgiram fazendo com que esses diagramas tivessem que ser atualizados. Por este motivo, e já inicialmente previsto, foi utilizado o modelo de desenvolvimento em espiral fazendo sempre a análise de riscos a cada etapa do desenvolvimento e retornando aos diagramas de análise sempre que possível.

Para uma boa compreensão do problema a ser resolvido no desenvolvimento deste trabalho é necessário ter uma noção exata do fluxo de vida do protótipo. Logicamente aqui abstraindo toda a etapa de levantamento de requisitos e partindo diretamente para os diagramas de fluxo como o que é apresentado a seguir.

O diagrama apresentado na Figura 3 exibe as etapas que o núcleo do sistema coordena e a relação de dependência que as classes `tcc.process.CBIRSequence` e `tcc.process.CBIRMeasurements` possuem. Como é possível observar neste diagrama de atividades os dados que alimentam a classe `tcc.process.CBIRMeasurements` são provenientes da classe que realiza o pré-processamento da imagem inclusive aplicando a transformada *wavelet*. Isto é necessário pois a segmentação em subespaços para extração de características necessita da imagem já com o filtro *wavelet* aplicado. Esta estrutura permite que caso futuramente seja necessário intercalar etapas de pré-processamento ou de cálculos de extração de características, bastaria estender as classes das interfaces correspondentes neste caso `tcc.process.Measurements` para cálculos e `tcc.process.Sequence` para pré-processamento e fazer um encadeamento entre elas podendo assim ter etapas intermediárias no processo de extração de características.

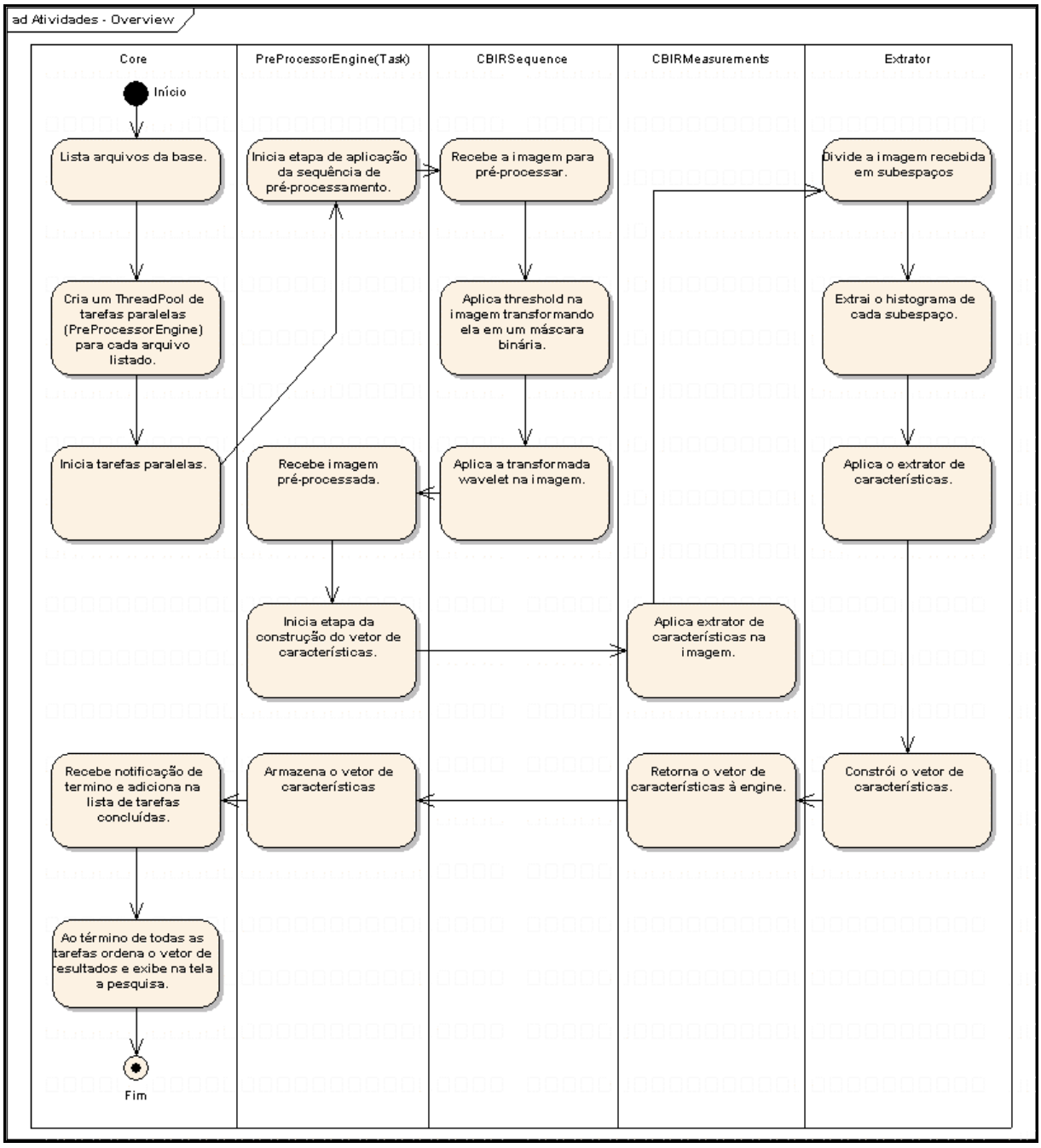


Figura 3 - Diagrama de atividades ciclo de vida

Após realizada a compreensão da visão geral dos procedimentos realizados pelo protótipo é apresentada a estrutura de extratores de características que foram definidos e implementados no protótipo. Estes extratores aplicam um determinado cálculo estatístico em um subespaço de uma imagem pré-processada. Cada extrator concreto é uma classe especializada da classe `tcc.process.extractors.AbstractExtractor` e sabe realizar um determinado cálculo de característica podendo ser energia, média, entropia, desvio padrão ou variância. A classe abstrata da qual os extratores herdam prove algumas funcionalidades

básicas e genéricas a qualquer extrator sendo uma delas, e talvez a mais importante, a segmentação da imagem de entrada em subespaços.

Como o método `apply()` da classe `tcc.process.extractors.AbstractExtractor` é abstrato é necessário que os extratores concretos implementem a sua funcionalidade. Desta forma foi criada uma estrutura totalmente transparente para as classes que utilizam e aplicam os extratores conforme Figura 4.

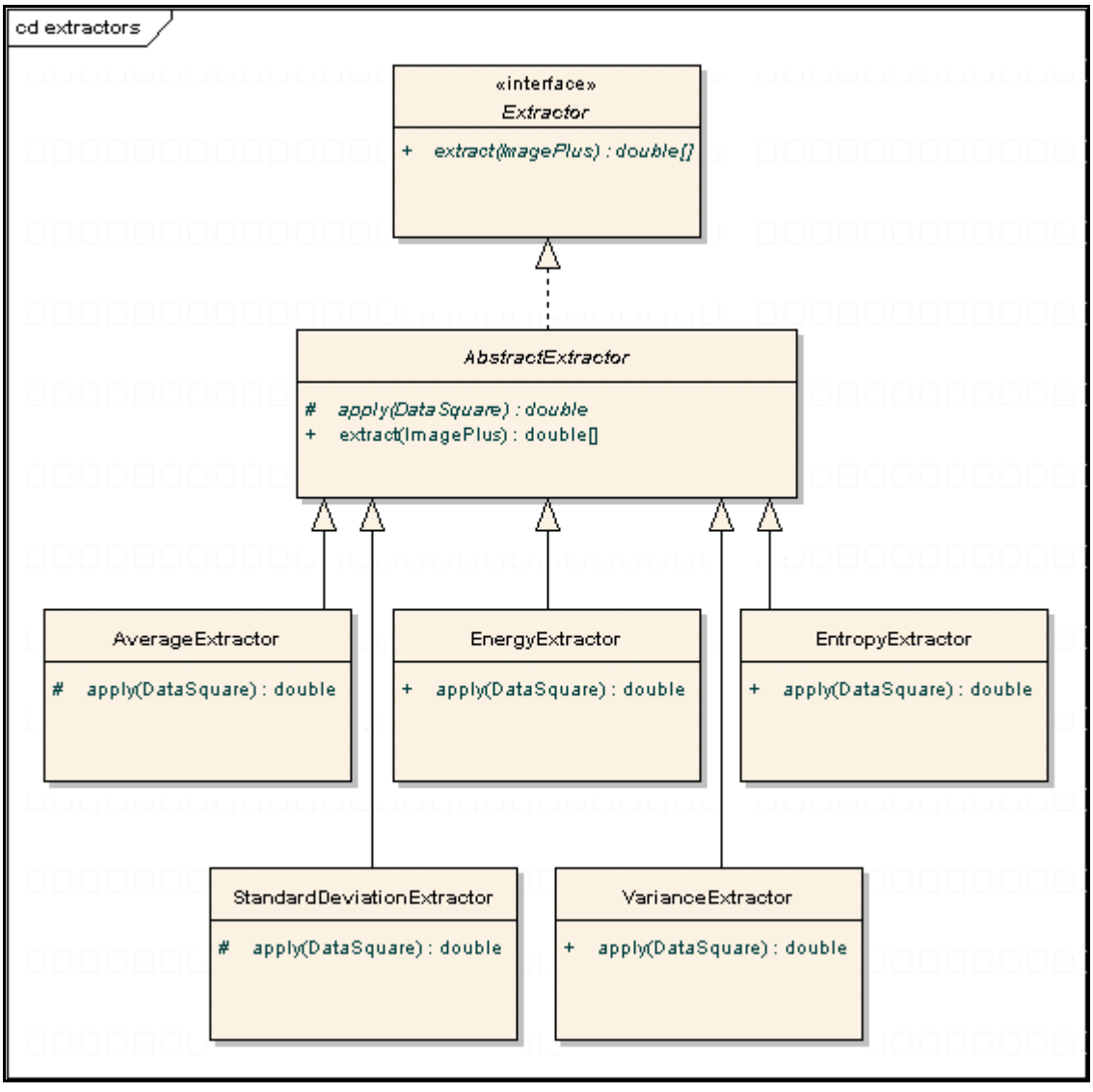


Figura 4 - Diagrama de classe estrutura de extratores

Seguindo a lógica do diagrama de atividades apresentado anteriormente a seguir são apresentadas as classes que utilizadas pelos extratores para realizarem os cálculos de características e a segmentação dos subespaços no caso do extrator abstrato. A classe `tcc.process.DataSquare` é a responsável pela segmentação de subespaços, ela recebe uma

matriz de números inteiros, que representam uma imagem, e a segmenta em quadrantes. A classe `tcc.process.EuclideanDistance` é utilizada para realizar o cálculo de distância euclidiana dado dois vetores e, por fim, a classe `tcc.process.Estatistic` centraliza e realiza todos os cálculos estatísticos utilizados pelos extratores concretos. O diagrama que representa está classes pode ser visto na Figura 5 abaixo.

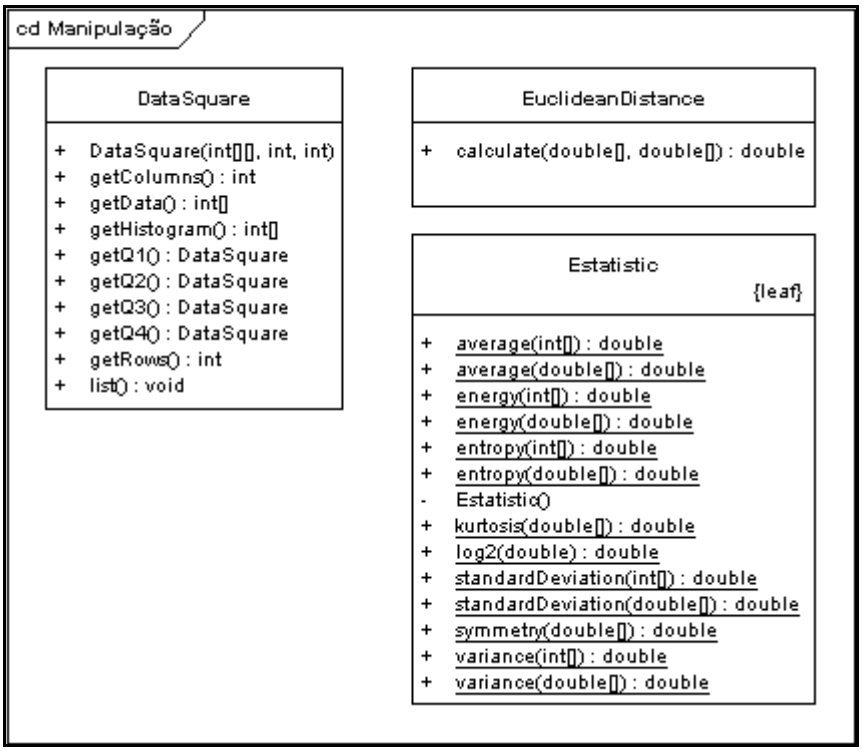


Figura 5 - Diagrama de classe cálculos estatísticos

Conforme dito acima, antes de aplicar os extratores de características na imagem desejada é necessário segmentá-la em subespaços por meio da transformada *wavelet* e da seqüência de pré-processamento. As classes implementadas para realizarem estas tarefas são exibidas no diagrama abaixo (Figura 6). A classe `tcc.process.FSW_Transform` é responsável por interagir com o *plugin* utilizado para aplicação da transformada *wavelet* na imagem. Já a classe `tcc.process.MyThresholder` realiza o ajuste inicial que transforma a imagem de entrada em uma imagem de máscara binária. Ambas as classes são utilizadas pela classe `tcc.process.CBIRSequence` e compõem a seqüência de pré-processamento utilizada neste trabalho.

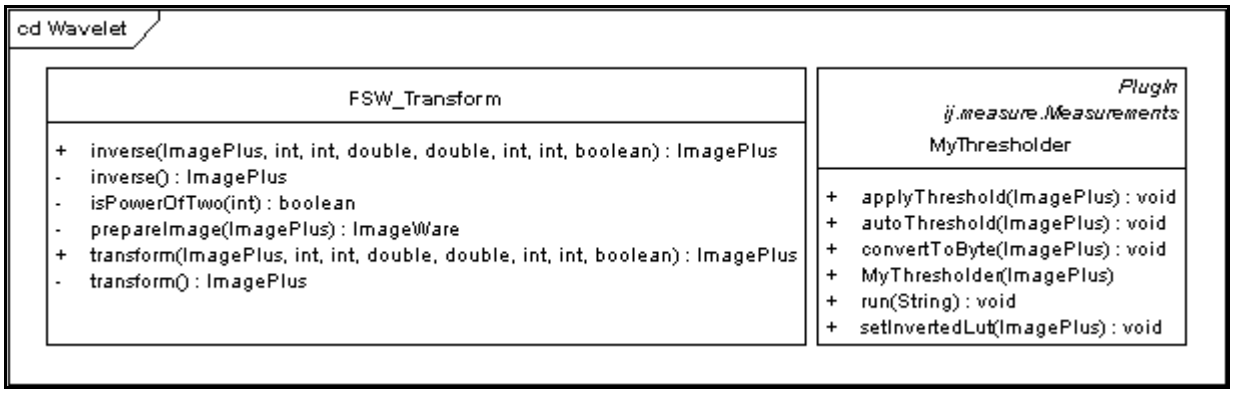


Figura 6 - Diagrama de classe pré-processamento

Uma vez definida a estrutura interna e a hierarquia dos extratores o próximo passo da análise deste trabalho é quanto às classes responsáveis efetivamente por coordenarem a execução das tarefas. O diagrama que representa esta estrutura de classes pode ser visto na Figura 7. Nele observa-se a classe `tcc.process.Core` que é o núcleo de todo o protótipo. Esta classe possui os métodos `search()` e `sort()` que fazem respectivamente a busca e a ordenação dos resultados retornados. O método `search()` realiza uma iteração sobre todos os arquivos da base de dados selecionada e instância um objeto da classe `tcc.process.PreProcessorEngine` para cada arquivo de imagem encontrado. Esta classe também é vista no diagrama abaixo e é a responsável por utilizar a seqüência de processamento e a classe de extração de características passadas como parâmetro na tarefa de processar uma dada imagem. Como esta classe pode ser utilizada de forma assíncrona, ela possui um *listener* que serve para receber as notificações de término do processamento de um destes objetos. O diagrama abaixo exibe a classe `tcc.process.Core` implementando este listener. Assim, a cada conclusão de uma tarefa, o método `done()` é chamado. Também na Figura 7 é possível ver a classe *tcc.process.Metadata* que é responsável pela persistência dos vetores de características calculados em disco.

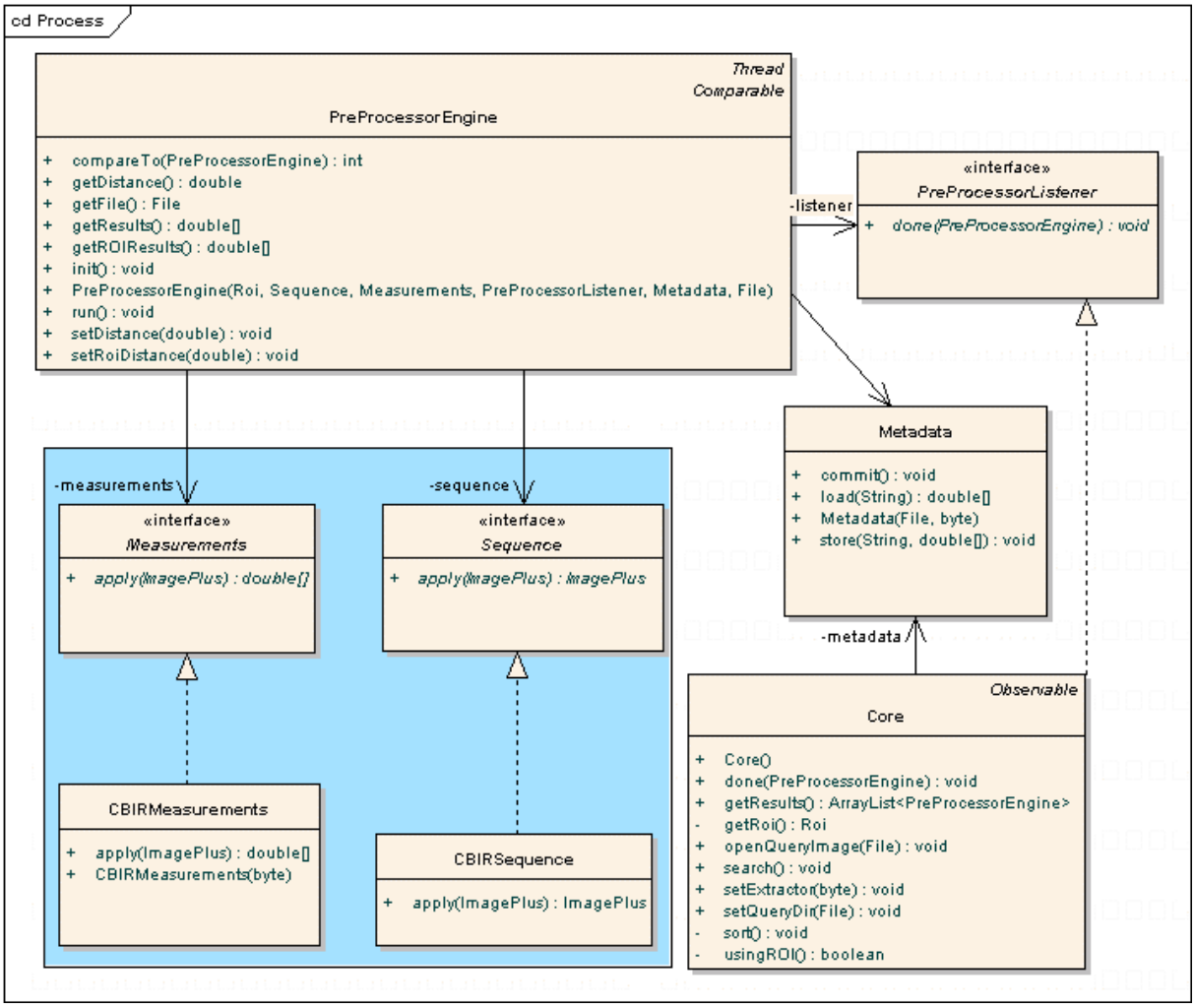


Figura 7 - Diagrama de classe núcleo do sistema

A área sombreada do diagrama acima exibe as classes que realizam a seqüência de pré-processamento e os cálculos dos vetores de características. Por este motivo são apresentados a seguir os diagramas de seqüência das partes e atividades mais relevante ao protótipo.

Inicialmente é apresentado o diagrama que representa as chamadas realizadas pelo método `search()` da classe `tcc.process.Core` quando da solicitação de uma busca pelo usuário. A seqüência de passos pode ser vista conforme Figura 8.

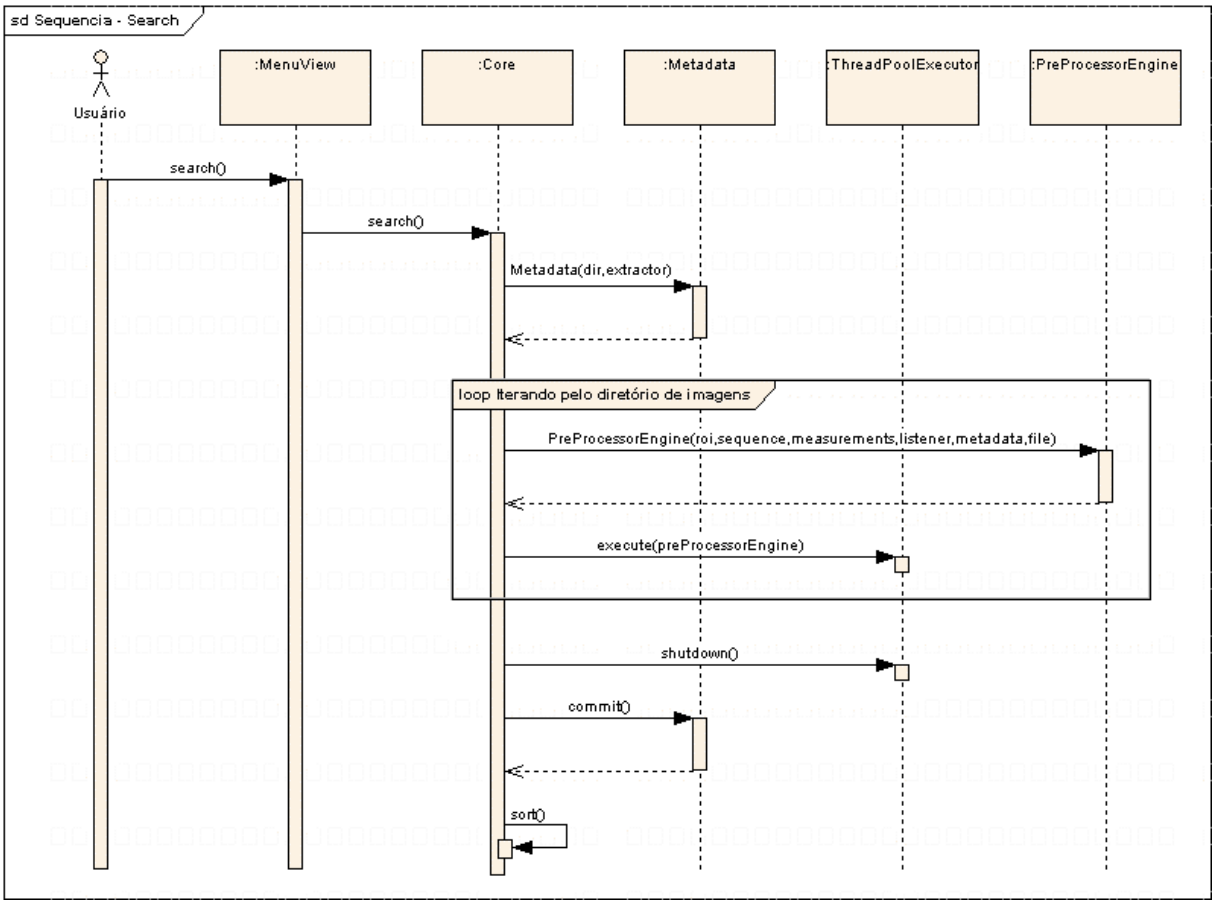


Figura 8 - Diagrama de seqüência método `search()`

Este diagrama detalha o uso da classe `tcc.process.PreProcessorEngine` durante a realização de uma busca. Por este motivo o próximo diagrama exibido (Figura 9) permite uma melhor visualização de como esta classe utiliza os recursos das classes `tcc.process.CBIRSequence` e `tcc.process.CBIRMeasurements`. Nele fica claro o uso da classe para aplicação de *threshold* e a dependência que o cálculo de características tem em relação à seqüência inicial de pré-processamento.

O diagrama de seqüência da Figura 9 detalha o método `init()` da classe `tcc.process.PreProcessorEngine` iniciando com a imagem de entrada e finalizando com a notificação de tarefa cumprida ao *listener* de controle representado aqui por sua interface. Isto porque uma das classes do sistema deve implementar esta interface a fim de receber as notificações. Este papel é desempenhado pela `classe tcc.process.Core`.

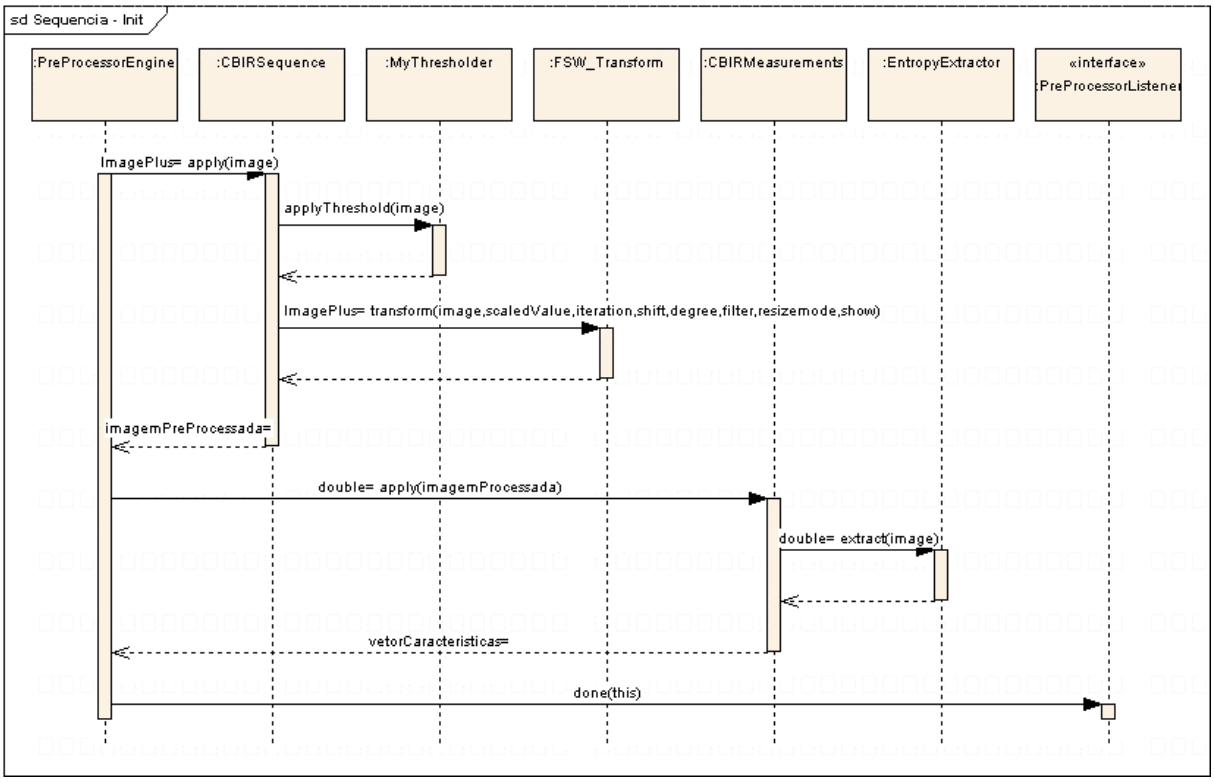


Figura 9 - Diagrama de seqüência método `init()`

Por fim, após todos os arquivos serem calculados é necessário apresentar os resultados para o usuário. O diagrama da Figura 10 detalha esta tarefa mapeando as chamadas realizadas pelo método `sort()` da classe `tcc.process.Core`. Como dito acima, cada término de uma tarefa de pré-processamento é notificado ao listener. Ao término de todas as tarefas o método `sort()` é chamado e então realiza a comparação e o cálculo de distância entre os vetores de características calculados. Para cada *engine* armazenada no *array* de resultados este método consulta os vetores de resultados e os compara com o vetor da imagem base setando através do método `setDistance()` da classe `tcc.process.PreProcessorEngine` a distância calculada. Esta classe implementa o método `compareTo()` que é utilizado para ordenação em função da menor distância obtida. No fim deste procedimento a classe `tcc.process.Core` executa a notificação para as classes de interface através dos métodos `setChanged()` e `notifiyObservers()` que solicitam a exibição dos resultados e apresentam em forma de miniaturas para o usuário.

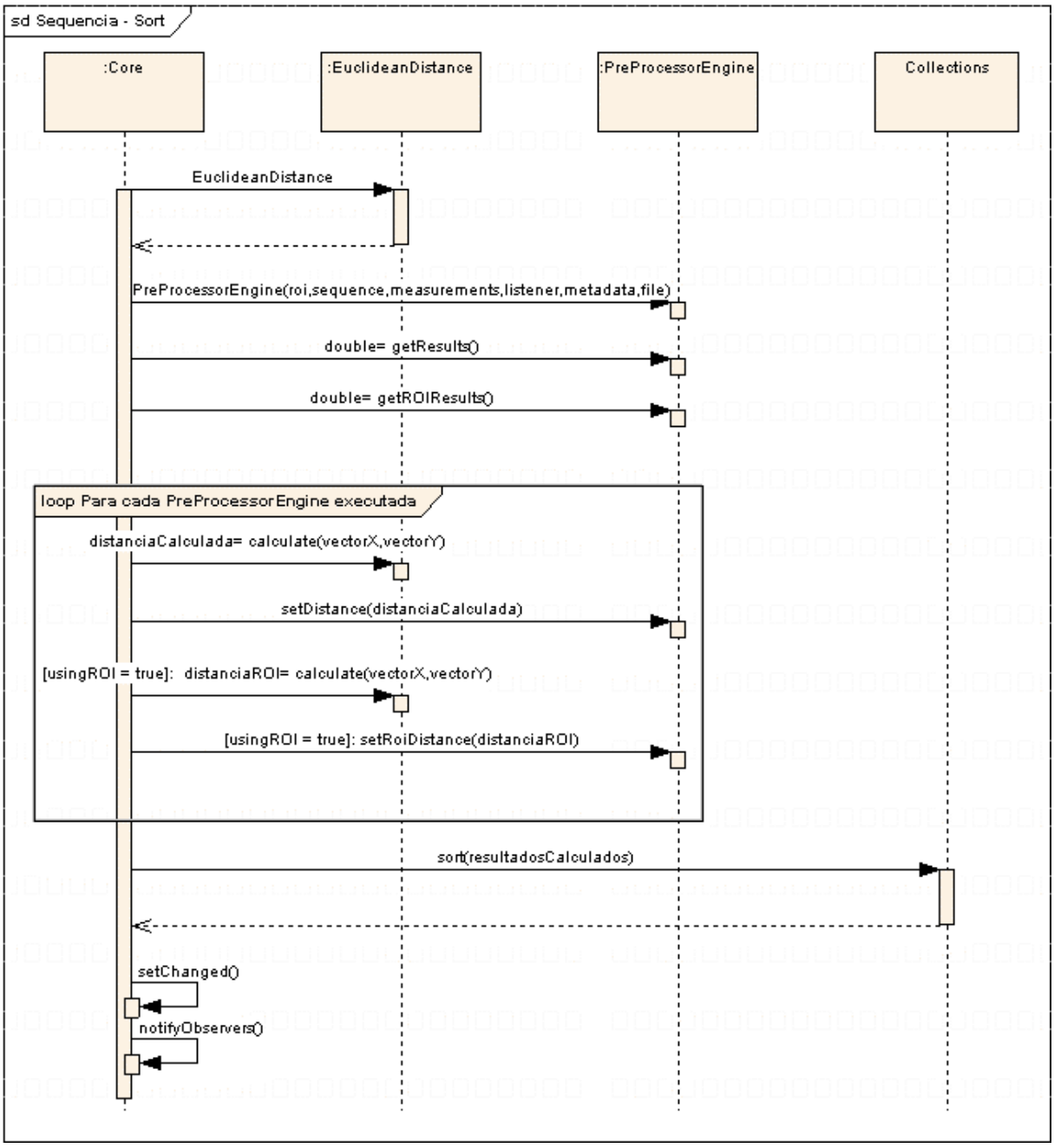


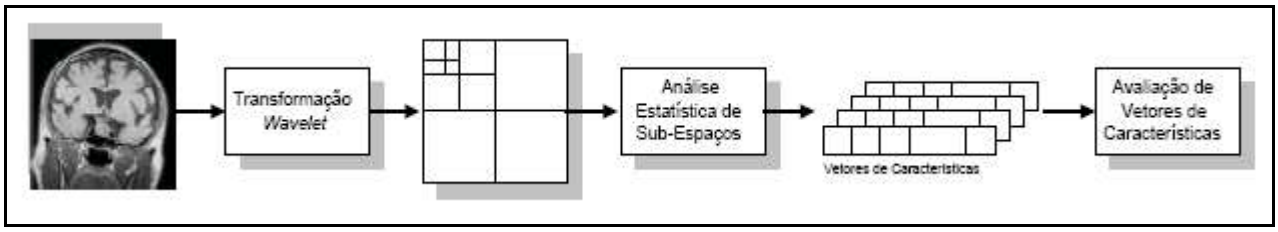
Figura 10 - Diagrama de seqüência método `sort()`

3.3 IMPLEMENTAÇÃO

Neste tópico são apresentadas as principais etapas e implementações realizadas no presente trabalho. De uma maneira geral partindo-se de uma imagem base o sistema é capaz de determinar as imagens similares em um banco de imagens e mostrar os resultados para o usuário. Para realizar este processo, o sistema aplica uma seqüência de passos até obtermos os

vetores de características que definem uma imagem para então podermos realizar a comparação entre as características.

A Figura 11 descreve, de uma maneira geral, o ciclo de vida para obtenção dos vetores de características. Após, os detalhes de cada etapa do processo são individualmente abordados e fundamentados.



Fonte: Castañón (2003, p. 52).

Figura 11 - Processo geral de extração de características

3.3.1 Aquisição das imagens

A API de manipulação de imagens e visão computacional ImageJ foi amplamente utilizada durante o processo de aquisição de imagens. A utilização desta API permite ao sistema abrir imagens no formato DICOM, amplamente utilizado na comunidade médica, e em outros formatos conhecidos como por exemplo JPEG, GIF, PNG e outros. Esta flexibilidade permite que imagens médicas obtidas na internet, geralmente em formatos não DICOM, possam ser utilizadas no banco de imagens da mesma forma. A classe Java da API ImageJ que representa uma imagem recebe o nome de `ij.ImagePlus`. Através desta classe é possível obtermos o conteúdo de uma imagem e suas principais propriedades. Por sua vez, esta classe quando utilizada em conjunto com a classe `ij.gui.ImageWindow` possibilita a exibição da imagem na tela para o usuário. A imagem a seguir (Figura 12) ilustra uma imagem carregada do banco de imagens de testes com a API ImageJ utilizando estas duas classes.

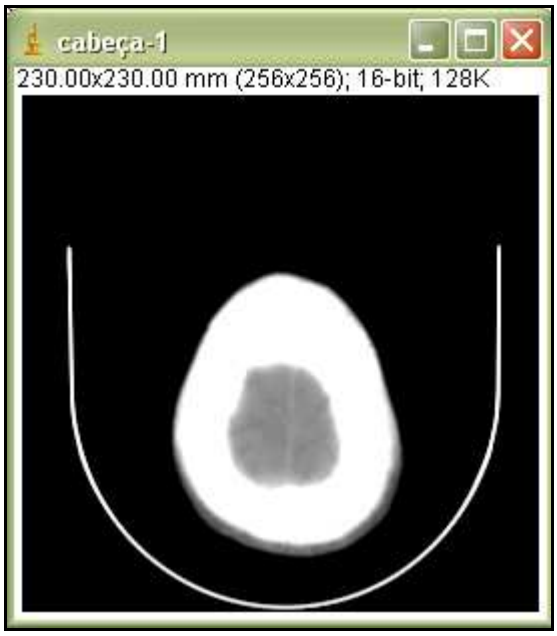


Figura 12 - Imagem DICOM carregada

Além da imagem também são exibidas informações como dimensões, formato e tamanho em *Kbytes*. O banco de imagens utilizado possui imagens nos formatos DICOM e JPEG e possui um total de 300 imagens. Para os testes este banco foi subdivido em bancos menores classificados segundo o tamanho das imagens.

3.3.2 Pré-processamento e ajustes

Algumas vezes ao obtermos uma imagem diretamente de uma fonte de captura ou de um banco de dados de imagens médicas é necessário realizar alguns ajustes nestas imagens. Processos como binarização, *threshold*, ajuste de contraste, ajuste de brilho são necessários para obtermos um resultado melhor ao aplicarmos a análise wavelet sobre a imagem. Estes ajustes não são uma etapa obrigatória no pré-processamento uma vez que é freqüente obtermos imagens com níveis de contraste e brilho aceitáveis e em condições de serem utilizadas.

Para este trabalho foi utilizado o ajuste de binarização para alguns casos. Para realizar este processamento nas imagens foi utilizada a classe `ij.process.ImageProcessor`. O ajuste destas imagens foi feito através de métodos de binarização sendo opcional ao processo geral de extração de características.

Para realizar o processo de binarização foi utilizada a técnica de *threshold*. Segundo

Silva (2002, p. 88), a técnica de limiarização *threshold* consiste em definir um valor de limiar para a cor de um *pixel*. *Pixels* com valor de cor acima de limiar se tornam pretos e de valor abaixo se tornam brancos. A Figura 13 abaixo mostra o resultado da aplicação desta técnica em uma imagem.

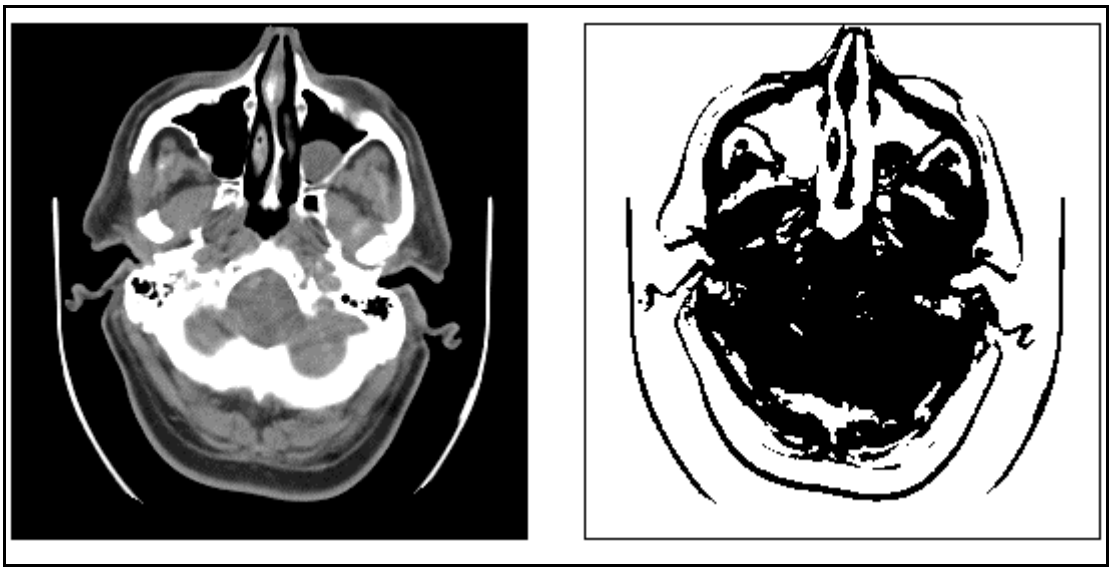


Figura 13 - Aplicação de *threshold*

A imagem da esquerda não está binarizada e apresenta os valores de *pixel* variando em diversos níveis de cinza. Já a imagem da direita é o resultado após a aplicação de *threshold*.

3.3.3 Análise *wavelet* e geração de subespaços

Partindo de uma imagem base, binarizada ou não, o sistema realiza uma análise através da extração de características desta imagem. Para isto é necessário extrair os vetores de características que descrevem uma determinada imagem. Neste trabalho estes vetores são armazenados em um arquivo chamado `metada.properties` sendo um para cada extrator utilizado. Esta armazenagem poderá ser posteriormente substituída por um sistema utilizando banco de dados, por exemplo, onde cada imagem seria armazenada junto ao seu vetor de características. O objetivo deste arquivo é ter os vetores de características previamente calculados. Assim, o tempo de pesquisa é otimizado. A técnica utilizada para extração destes vetores baseia-se no trabalho correlato de Castañón (2003).

Inicialmente a imagem a ser analisada é submetida a uma seqüência de análise por

wavelets. Segundo Castañón (2003) uma observação dos subespaços de *wavelets* (Figura 14) indica que existe uma relação entre os distintos subespaços, o que forma uma estrutura piramidal, de subimagens com várias resoluções correspondentes a diferentes escalas. Por exemplo, o subespaço LLLH tem informação similar do que o subespaço LH. Isto por que ambos os subespaços são gerados pelo mesmo filtro, mas em escalas diferentes.

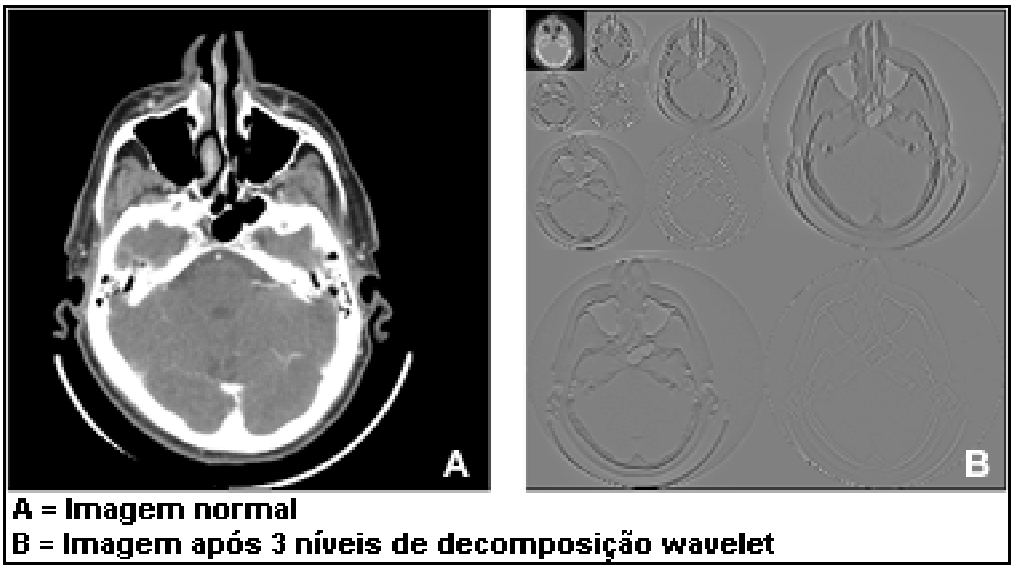


Figura 14 - Decomposição *wavelets*

O cálculo do vetor de características é feito através da aplicação de um extrator como, por exemplo, entropia ou energia, em cada subespaço. Para cada subespaço um valor é gerado e ajudará a compor o vetor de características daquela imagem. No presente trabalho foram aplicados três níveis de decomposição *wavelet*. Assim, para cada imagem nove subespaços são gerados. Em cada subespaço um extrator é aplicado e o valor calculado compõe o vetor de características. A Figura 15 a seguir ilustra a subdivisão de uma imagem em diferentes subespaços.

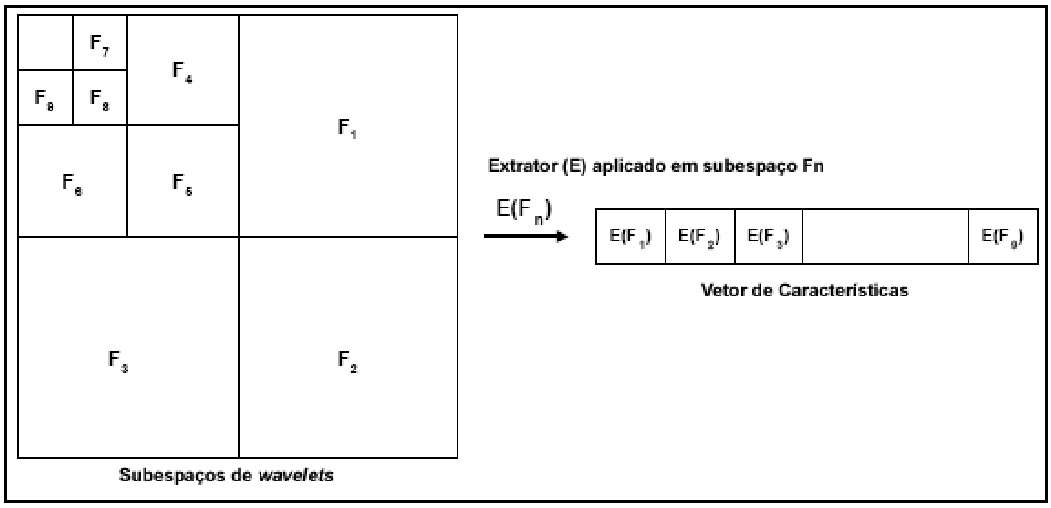


Figura 15 - Subespaços de *wavelets*

Os extratores aplicados neste trabalho são medidas estatísticas que podem ser utilizadas para extrair características e avaliar a luminosidade da imagem (através da média), a suavidade (através da entropia), e a uniformidade (através da energia). Além destes extratores também foram implementados desvio padrão e variância conforme as fórmulas abaixo.

Variância:

$$\sigma^2 = \sum_{i=0}^{G-1} (i-\mu)^2 \, p(i)$$

Média:

$$\mu = \sum_{i=0}^{G-1} i p(i)$$

Entropia:

$$H = -\sum_{i=0}^{G-1} p(i) \log_2 [p(i)]$$

Energia:

$$E = \sum_{i=0}^{G-1} [p(i)]^2$$

Desvio Padrão:

$$\sigma = \sqrt{\sum_{i=0}^{G-1} (i-\mu)^2 \, p(i)}$$

3.3.4 Comparação dos vetores de características

Dois fatores são muito importantes na busca de imagens por conteúdo utilizando a técnica utilizada para este trabalho. São eles: uma representação apropriada das características visuais das imagens e uma medida que determine o grau de similaridade das imagens de resposta. Conforme descrito acima, neste trabalho foi utilizada para geração dos vetores de características a extração de características por análise *wavelets* e extratores estatísticos. Para comparação entre os vetores calculados foi utilizada a métrica de distância euclidiana definida pela seguinte equação:

$$d(x,y) = \sqrt{\sum_{i=1}^{m}(x_i - y_i)^2}$$

Assim, dado o vetor de característica da imagem base é realizada a comparação de distância em relação aos outros vetores de características. Estes resultados são armazenados em memória no objeto `tcc.process.PreProcessorEngine` que é responsável pela execução e armazenamento dos resultados de um processamento sobre uma imagem.

A cada execução deste processamento uma lista de imagens processadas vai sendo criada para posteriormente ser ordenada segundo as distâncias calculadas. Ao término de todo o processamento o sistema exibe os resultados obtidos. Segue abaixo uma imagem (Figura 16) que demonstra uma pesquisa feita utilizando o protótipo desenvolvido.

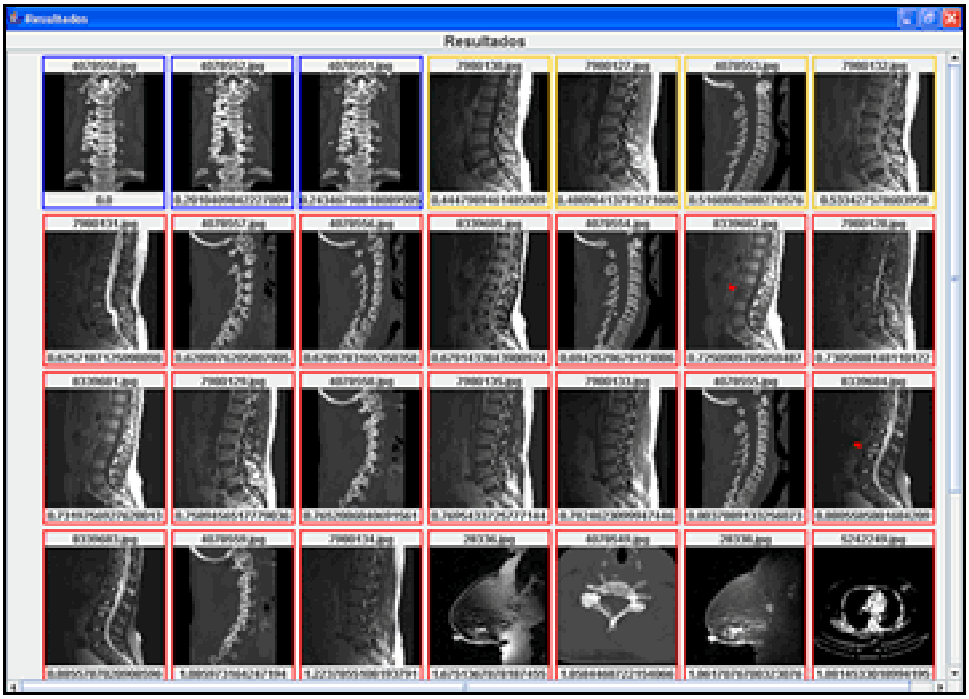


Figura 16 - Apresentação de resultados

3.3.5 Técnicas e ferramentas utilizadas

As técnicas empregadas no desenvolvimento deste trabalho buscaram sempre a facilidade de extensão e usabilidade em outras arquiteturas. Além disso, a utilização de alguns padrões de projeto e hierarquia de classes, através de herança e interfaces, foram amplamente aplicados visando um código limpo de fácil manutenção e possível de reaproveitamento. A seguir é exibido o conjunto de técnicas e alguns códigos produzidos na implementação do

protótipo.

3.3.5.1 Model View Controller (MVC)

Uma das preocupações durante o desenvolvimento de alguns componentes do protótipo é permitir que as classes do núcleo do sistema possam ser independentes da interface de manipulação. Seguindo está abordagem seria possível portar facilmente a implementação para trabalhar em um ambiente cliente servidor, ou até mesmo em uma plataforma WEB 2.0. A única dificuldade para realizar está implementação seria o desenvolvimento de uma nova interface que interagisse com o núcleo principal. Assim, para o desenvolvimento do protótipo foi adotado o padrão de projeto *Model View Controller* (MVC) para poder separar a interface das classes do núcleo.

Segundo Stelting e Maassen (2002, p.209) MVC é um padrão de arquitetura de aplicações que visa separar a lógica da aplicação, *Model*, da interface do usuário, *View*, e do fluxo da aplicação, *Controller*. Permite que a mesma lógica de negócios possa ser acessada e visualizada por várias interfaces. A linguagem de programação utilizada, Java, dá suporte à implementação deste modelo de projeto através da classe `java.util.Observable` e da interface `java.util.Observer`, fazendo o papel de *Model* e *View* respectivamente. O padrão de projeto O*bserver* dá suporte à implementação da MVC.

Os quadros a seguir exibem a utilização destas classes para determinar a classe `tcc.process.Core` como sendo o modelo do protótipo e as classes `tcc.ui.ResultView` e `tcc.ui.MenuView` como sendo as classes de visão.

O quadro abaixo (Quadro 2) exibe o código reduzido da classe `tcc.ui.ResultView` dando ênfase a declaração desta classe que implementa a interface `java.util.Observer` e ao método `update()` que é responsável por tratar as notificações do modelo.

```
package tcc.ui;

{...}

import java.util.Observable;
import java.util.Observer;

public class ResultView extends JFrame implements Observer, ActionListener {

{...}

public void update(Observable o, Object arg) {

               // removendo resultados anteriores
               contentPane.removeAll();

               // resultados atuais
               ArrayList<PreProcessorEngine> results = core.getResults();

               int i = 0;
               // criando as miniaturas
               for (PreProcessorEngine engine : results) {
                      if (i++ == qtdResults)
                             break;
                      contentPane.add(new ImageTipView(core, engine.getFile(),
engine.getEuclideanDistance(), scale));
               }

               // exibindo
               setVisible(true);
               validate();
               repaint();
       }

{...}

}
```

Quadro 2 - Uso do padrão de projeto MVC – Classe `ResultView`

O quadro abaixo (Quadro 3) exibe o código reduzido da classe `tcc.ui.MenuView` dando ênfase a declaração desta classe que implementa a interface `java.util.Observer` e ao construtor que recebe o modelo `tcc.process.Core` e registra-se como uma *view* para receber as atualizações do modelo pelo método `update()` explicado acima.

```
package tcc.ui;

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.util.Observable;
import java.util.Observer;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JToolBar;

import tcc.process.Core;

public class MenuView extends JFrame implements ActionListener, Observer {


    public MenuView(Core core) {
                this.core = core;
                core.addObserver(this);

    {...}

    }

    {...}

}
```

Quadro 3 - Uso do padrão de projeto MVC – Classe `MenuView`

O quadro abaixo (Quadro 4) exibe o código reduzido da classe `tcc.process.Core` dando ênfase a declaração desta classe que estende da classe `java.util.Observable` e implementa o modelo do padrão MVC. Também são exibidos alguns métodos que possuem a chamada para os métodos `setChanged()` e `notifyObservers()` responsáveis por disparar os eventos de atualização das *views* registradas. Respectivamente, o primeiro método é responsável por selecionar a condição para que a próxima notificação seja entregue e o segundo efetivamente realiza a notificação nas *views*.

```
package tcc.process;

import ij.ImagePlus;
import ij.gui.ImageWindow;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Observable;

import tcc.main.Constants;

public class Core extends Observable implements PreProcessorListener {

{...}

      /** Ordena a lista de resultados */
      private void sort() {

           {...}

            // Ordenando a lista
            Collections.sort(results);

            for (PreProcessorEngine ppe : results) {
                  System.out.println("Arquivo: " + ppe.getFile() + "
Distância: " + ppe.getEuclideanDistance());

            }

            // Notifica as views
            setChanged();
            notifyObservers();

      }

{...}

}
```

Quadro 4 - Uso do padrão de projeto MVC – Classe `Core`

Desta forma, o protótipo pode ser facilmente portado para outra arquitetura sem ficar preso a sua interface *stand-alone*.

3.3.5.2 *Classe* `PreProcessorEngine`

A necessidade de aplicar a seqüência de pré-processamento e realizar os cálculos e procedimentos necessários para extrair os vetores de características demanda uma estrutura de classes flexível. Para isso foi implementado a classe `tcc.process.PreProcessorEngine`. Esta classe aplica uma seqüência de processamento em uma imagem e executa os cálculos necessários para geração dos vetores de características utilizando as classes `tcc.process.CBIRSequence` e `tcc.process.CBIRMeasurements` que são capazes de respectivamente aplicar uma seqüência de pré-processamento e extrair os vetores de

características. O vetor resultante é armazenado em um objeto da classe `PreProcessorEngine` para posterior consulta ou comparação.

A classe `PreProcessorEngine` pode trabalhar de maneira síncrona ou assíncrona pois é uma subclasse de `Thread`. Quando trabalhando de maneira assíncrona o término do processamento pode ser capturado por um objeto da classe `tcc.process.PreProcessorListener`. Conforme citado acima os dados dos vetores de características são armazenados em disco para uma consulta mais rápida em um segundo momento. Para isso é utilizado a classe `tcc.process.Metadata`. Ao realizar uma busca o sistema procura primeiro no arquivo *metadata* para obter o vetor de características da imagem. Caso não encontre o vetor pré-calculado no arquivo então o vetor e gerado durante a busca. O Quadro 5 exibe o conteúdo de um arquivo *metadata* gerado.

```
#Vetores de características das imagens deste diretório
#Mon May 14 01:05:12 GMT-03:00 2007
11.2.840.113619.2.22.287.1.32536.2.3.20060323.223529.dcm=[7.185427073712201,
6.505670304139862, 6.29381386399782, 7.4216827101016785, 7.250264121043106,
7.177296593327406, 7.861676116924295, 7.575724086487715, 7.445769822141119]
11.2.840.113619.2.22.287.1.32536.2.40.20060323.223912.dcm=[7.4736884340411045,
6.834877509204756, 7.027651327105941, 7.784532375262196, 7.471049919021016,
7.6838342858955295, 7.81070099422582, 7.760476001637566, 7.799972526376337]
11.2.840.113619.2.22.287.1.32536.2.15.20060323.223646.dcm=[7.375847736926836,
6.689084648897871, 6.726703523071232, 7.695134524238913, 7.373615142233085,
7.448909249025137, 7.875164625715807, 7.763799438644121, 7.6282105879405995]
11.2.840.113619.2.22.287.1.32536.2.37.20060323.223855.dcm=[7.4717927061659175,
6.800344643343095, 6.975433820762091, 7.770636763097506, 7.4318665263152806,
7.666090896275711, 7.83096040144834, 7.743237481533394, 7.856603790319361]
11.2.840.113619.2.22.287.1.32536.2.20.20060323.223715.dcm=[7.341058160301773,
6.7696077817410165, 6.845271405191459, 7.68583337372364, 7.4133385545959065,
7.499097051725296, 7.859902139110438, 7.763331782610979, 7.715094543835367]
11.2.840.113619.2.22.287.1.32536.2.2.20060323.223523.dcm=[7.233580711742658,
6.522607528006936, 6.244612404494359, 7.4642271500958834, 7.270881210673306,
7.143359812334704, 7.832941419962005, 7.598649890821075, 7.434060730809878]
11.2.840.113619.2.22.287.1.32536.2.17.20060323.223658.dcm=[7.312608399931785,
6.738808393116317, 6.754372027432413, 7.641326545535827, 7.374247700362708,
7.431655283261554, 7.841004257209979, 7.719843994936239, 7.674365878185384]
11.2.840.113619.2.22.287.1.32536.2.14.20060323.223640.dcm=[7.3330261513233985,
6.671801478358541, 6.683066352116488, 7.669882842614826, 7.359235113621509,
7.419901384670382, 7.882266664687656, 7.761880506572937, 7.606334862072612]
11.2.840.113619.2.22.287.1.32536.2.22.20060323.223727.dcm=[7.430829528567367,
6.791186142409667, 6.828672339872439, 7.743865671506004, 7.4224409088018595,
7.556140501955716, 7.864677957933681, 7.783599161579602, 7.750813823180288]
11.2.840.113619.2.22.287.1.32536.2.38.20060323.223901.dcm=[7.49470796246714,
6.81962126406009, 6.9955346466430335, 7.783704495719124, 7.4560346675454525,
7.6817020221685315, 7.82888462396896, 7.753254048970633, 7.824456240982165]
```

Quadro 5 - Arquivo *metadata* gerado utilizando extrator entropia

O Quadro 6 exibe a classe `tcc.process.Metadata` responsável por gerar o arquivo do exibido no Quadro 5.

```
package tcc.process;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Arrays;
import java.util.Properties;

import tcc.main.Constants;

public class Metadata {

	private File fproperties = null;

	private Properties properties = null;

	public Metadata(File dir, byte extractor) {

		String prefix = null;

		if (Constants.EXTRACTOR_ENTROPY == extractor) {
			prefix = "entropy";
		}
		if (Constants.EXTRACTOR_ENERGY == extractor) {
			prefix = "energy";
		}
		if (Constants.EXTRACTOR_AVERAGE == extractor) {
			prefix = "average";
		}
		if (Constants.EXTRACTOR_VARIANCE == extractor) {
			prefix = "variance";
		}
		if (Constants.EXTRACTOR_STANDARD_DEVIATION == extractor) {
			prefix = "s_deviation";
		}
		if (Constants.EXTRACTOR_COMBINATED == extractor) {
			prefix = "combinated";
		}

		this.fproperties = new File(dir, prefix +
"_metadata.properties");
		this.properties = new Properties();

		System.out.println("Metadata selecionado: " +
fproperties.getName());

		// carregando para memória caso o arquivo exista
		if (fproperties.exists()) {
			FileInputStream input = null;
			try {
				input = new FileInputStream(fproperties);
				properties.load(input);
			} catch (Exception e) {
				e.printStackTrace();
			}
			try {
				input.close();
			} catch (IOException e) {
				e.printStackTrace();
			}
		}
	}

	/** Grava um vetor de caracteristicas no metadata */
	public synchronized void store(String key, double[] vector) {
		properties.setProperty(key, Arrays.toString(vector));
	}
```

```
	/** Lê um vetor de caracteristicas do metadata */
	public synchronized double[] load(String key) {
		String sVector = properties.getProperty(key);

		if (sVector == null)
			return null;

		// removendo os caracteres indesejados
		sVector = sVector.replace("[", "");
		sVector = sVector.replace("]", "");

		String[] aVector = sVector.split(",");
		double[] results = new double[aVector.length];

		for (int i = 0; i < aVector.length; i++) {
			results[i] = Double.parseDouble(aVector[i]);
		}

		return results;
	}

	/** Envia os dados para o arquivo */
	public synchronized void commit() throws IOException {
		FileOutputStream output = null;
		try {
			output = new FileOutputStream(fproperties, false);
			properties.store(output,
					"Vetores de características das imagens deste
diretório");
		} catch (Exception e) {
			e.printStackTrace();
		} finally {
			output.close();
		}
	}
}
```

Quadro 6 - Classe `tcc.process.Metadata`

O quadro 7 apresenta a classe `tcc.process.PreProcessorEngine` responsável pelo pré-processamento das imagens.

```
package tcc.process;

import ij.ImagePlus;
import ij.gui.Roi;

import java.io.File;
import java.util.Arrays;

public class PreProcessorEngine extends Thread implements
            Comparable<PreProcessorEngine> {

      private Sequence sequence;
      private Measurements measurements;
      private double[] results;
      private double[] roiResults;
      private File file;
      private PreProcessorListener listener;
      private Metadata metadata;
      private double distance;
      private double roiDistance;
      private Roi roi;

      public PreProcessorEngine(Roi roi, Sequence sequence,
                  Measurements measurements, PreProcessorListener listener,
                  Metadata metadata, File file) {
            this.roi = roi;
            this.sequence = sequence;
            this.measurements = measurements;
            this.listener = listener;
            this.metadata = metadata;
            this.file = file;
      }

      /** Executa engine de forma assincrona */
      public void run() {
            init();
      }

      /** Inicia o processamento */
      public void init() {
            ImagePlus image = new ImagePlus(file.getAbsolutePath());
            try {

                  // aplicando sequencia a uma ROI
                  if (roi != null) {
                        image.setRoi(roi);
                        image.copy(false);

                        int width = roi.getBounds().width;
                        int height = roi.getBounds().height;

                        ImagePlus imageROI = new ImagePlus("ROI - Image",
image
                                    .getProcessor().createProcessor(width,
height));
                        imageROI.paste();
                        // Limpando ROI
                        imageROI.setRoi((Roi) null);
                        image.setRoi((Roi) null);

                        roiResults =
measurements.apply(sequence.apply(imageROI));
                        System.out.println("RESULTADOS (ROI): "
                                    + Arrays.toString(roiResults));
                  }

                  // procura vetor de características no metadata
                  if (metadata != null) {
                        results = metadata.load(file.getName());
```

```
                        System.out.println("RESULTADOS metadata: "
                                    + Arrays.toString(results));
                  }

                  if (results == null) {
                        results =
measurements.apply(sequence.apply(image));
                        System.out.println("RESULTADOS aplicado: "
                                    + Arrays.toString(results));
                        if (metadata != null)
                              metadata.store(file.getName(), results);
                  }

            } catch (Exception e) {
                  e.printStackTrace();
                  System.err.println("Erro ao processar arquivo: "
                              + file.getAbsolutePath());
            }

            // Notifica término...
            if (listener != null)
                  listener.done(this);
            // Liberando memória
            image = null;
      }

      /** Retorna os dados calculados sobre esta imagem */
      public double[] getResults() {
            return results;
      }

      /** Retorna os dados calculados sobrea a ROI desta imagem */
      public double[] getROIResults() {
            return roiResults;
      }

      /** Retorna o arquivo que foi processado */
      public File getFile() {
            return file;
      }

      /** Seta a distância deste processamento */
      public void setDistance(double value) {
            this.distance = value;
      }

      /** Seta a distância da ROI */
      public void setRoiDistance(double value) {
            this.roiDistance = value;
      }

      /** Retorna o valor da distancia */
      public double getDistance() {
            if (roi != null)
//                return distance + roiDistance;
                  return ((distance * 1.0) + (roiDistance * 2.0)) / 3.0;
            else
                  return distance;
      }

      public int compareTo(PreProcessorEngine o) {
            double ed = o.getDistance();
            double d = getDistance();

            if (d == ed)
                  return 0;

            if (d < ed)
                  return -1;
```

```
            else
                  return 1;

      }
}
```

Quadro 7 - Classe `tcc.process.PreProcessorEngine`

3.3.5.3 Classe *`ThreadPoolExecutor`*

A geração dos vetores de características possui um *overhead* de processamento girando em torno de 1seg por imagem. Para tentar reduzir este tempo foi utilizado um padrão de projeto conhecido como *WorkerThread* (STELTING; MAASSEN, 2002, p.231). Este padrão introduz concorrência na extração dos vetores de características disparando processos em paralelo diminuindo assim o tempo de processamento. Para esta implementação foi utilizado a classe `java.util.concurrent.ThreadPoolExecutor` que implementa este padrão. Uma lista de tarefas é submetida a um objeto desta classe e vão sendo executadas paralelamente obedecendo um número máximo de tarefas permitidas. O quadro abaixo exibe o trecho de código responsável por disparar estas tarefas de forma paralela (Quadro 8).

```
{...}
taskExecutor = new ThreadPoolExecutor(5,5, 50000L,
      TimeUnit.MILLISECONDS, new LinkedBlockingQueue<Runnable>());

      for (File file : files) {
            Sequence sequence = new CBIRSequence();
            Measurements measurements = new CBIRMeasurements(extractor);
            taskExecutor.execute(new PreProcessorEngine(getRoi(), sequence,
                        measurements, this, metadata, file));
      }
      taskExecutor.shutdown();
{...}
```

Quadro 8 - `ThreadPoolExecutor`, pré-processamento paralelo.

3.3.5.4 Classe Core

A classe `tcc.process.Core` é apresentada a seguir e representa o núcleo do protótipo desenvolvido. Ela é a responsável por disparar as tarefas de pré-processamento, executar as buscas e notificar as interfaces dos resultados obtidos. Os dois métodos principais desta classe são `search()` e `sort()`. Respectivamente são responsáveis por executar a busca e ordenar os resultados encontrados do mais próximo ao mais distante. O Quadro 9 exibe o conteúdo desta

classe.

```
package tcc.process;

{...}

public class Core extends Observable implements PreProcessorListener {
	/** Array de resultados */
	private ArrayList<PreProcessorEngine> results = null;
	/** Imagem base */
	private File queryFile;
	/** Diretório para pesquisa */
	private File queryDir;
	private ImageWindow iw;
	private DrawROI drawROI;
	private byte extractor = Constants.EXTRACTOR_ENERGY;
	private ThreadPoolExecutor taskExecutor = null;
	private Metadata metadata;

	public Core() {
		results = new ArrayList<PreProcessorEngine>();
	}

	/** Abre a imagem que será usada para a consulta */
	public void openQueryImage(File file) {
		this.queryFile = file;
		ImagePlus image = new ImagePlus(file.getAbsolutePath());
		// Destroi janela
		if (iw != null)
			iw.dispose();
		// Destroi drawROI
		if (drawROI != null) {
			drawROI.destroy();
		}
		iw = new ImageWindow(image);
		drawROI = new DrawROI(iw);
	}


	/** Seta o diretório para pesquisar */
	public void setQueryDir(File queryDir) {
		this.queryDir = queryDir;
	}

	/** Inicia a busca */
	public void search() {
		if (queryDir == null) return;
		// Limpando o array de resultados
		results.clear();
		File[] files = queryDir.listFiles();
		metadata = new Metadata(queryDir, extractor);
		taskExecutor = new ThreadPoolExecutor(5,5, 50000L,
				TimeUnit.MILLISECONDS, new LinkedBlockingQueue<Runnable>());

		for (File file : files) {
			/* controle para evitar os arquivos .db gerados pelo windows */
			if (file.getName().endsWith(".db")
					|| file.getName().endsWith(".properties")
					|| file.getName().endsWith(".svn"))
				continue;

			Sequence sequence = new CBIRSequence();
			Measurements measurements = new CBIRMeasurements(extractor);
			// new PreProcessorEngine(getRoi(), sequence, measurements, this,
			// metadata, file).init();

			taskExecutor.execute(new PreProcessorEngine(getRoi(), sequence,
					measurements, this, metadata, file));
		}
		taskExecutor.shutdown();

		synchronized (this) {
			while (!taskExecutor.isTerminated())
				try {
					wait(500);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
		}

		try {
			metadata.commit();
		} catch (IOException e) {
			e.printStackTrace();
		}
		sort();
		taskExecutor = null;
	}
```

```
	/** Retorna os resultados, array com os arquivos classificados pela busca */
	public ArrayList<PreProcessorEngine> getResults() {
		return results;
	}

	public void done(PreProcessorEngine source) {
		results.add(source);
	}

	/** Seleciono qual extrator de características será utilizado */
	public void setExtractor(byte extractor) {
		this.extractor = extractor;
	}

	/** Ordena a lista de resultados */
	private void sort() {
		System.out.println("\nSORTING...\n");
		// Comparador de distância
		EuclideanDistance ed = new EuclideanDistance();

		// Engine para imagem base
		Sequence sequence = new CBIRSequence();
		Measurements measurements = new CBIRMeasurements(extractor);
		PreProcessorEngine engine = new PreProcessorEngine(getRoi(), sequence,
				measurements, null, null, queryFile);

		// Gerando o vetor de caracteristicas da imagem base
		engine.init();
		double[] baseVector = engine.getResults();
		double[] baseROIVector = engine.getROIResults();

		// Comparando a distancia entre todos os vetores
		for (PreProcessorEngine ppe : results) {
			ppe.setDistance(ed.calculate(baseVector, ppe.getResults()));
			if (usingROI())
				ppe.setRoiDistance(ed.calculate(baseROIVector, ppe
						.getROIResults()));
		}

		// Ordenando a lista
		Collections.sort(results);

		for (PreProcessorEngine ppe : results) {
			System.out.println("ORDENANDO... Arquivo: " + ppe.getFile()
					+ " Distância: " + ppe.getDistance() + " Usando ROI: "
					+ usingROI());
		}
		setChanged();
		notifyObservers();
		System.out.println("\nSORT DONE\n");
	}

	/** Retorna se a imagem possui alguma ROI setada */
	private boolean usingROI() {
		return iw != null && iw.getImagePlus().getRoi() != null;
	}

	/** Retorna o ROI da imagem base */
	private Roi getRoi() {
		if (iw != null)
			return iw.getImagePlus().getRoi();
		else
			return null;
	}
}
```

Quadro 9 - Classe `tcc.process.Core`

3.3.5.5 Classe `DataSquare` segmentador de subespaços

Conforme citado anteriormente, após a aplicação da transformada *wavelet* a imagem desejada é necessário calcular os valores para cada subespaço gerado. Conforme descrito na Figura 15 a imagem se divide em diversos quadrantes sendo que para a aplicação de cada

extrator é necessário trabalhar com cada um dos subespaços ***Fn*** de forma individual. A classe `tcc.process.DataSquare` foi implementada com o objetivo de executar esta tarefa. Esta classe segmenta uma imagem dada em quatro quadrantes e possui métodos para retornar cada quadrante individualmente assim como retornar informações sobre o histograma da imagem passada como parâmetro para seu construtor. Cada quadrante retornado é também um objeto desta mesma classe. Assim, é possível uma chamada de métodos seqüencial ou até mesmo recursiva. O Quadro 10 exibe o conteúdo desta classe.

```
package tcc.process;

public class DataSquare {

	private int[][] data; //matriz de pixels da imagem total
	private int[][] q1;   //matriz de pixels do quadrante 1 da imagem
	private int[][] q2;   //matriz de pixels do quadrante 2 da imagem
	private int[][] q3;   //matriz de pixels do quadrante 3 da imagem
	private int[][] q4;   //matriz de pixels do quadrante 4 da imagem
	private int r_data;   //controle da linha da matriz data
	private int c_data;   //controle da coluna da matriz data
	private int r_zero;   //linha inicial
	private int r_half;   //meio do total de linhas
	private int c_zero;   //coluna inicial
	private int c_half;   //meio do total de colunas

	public DataSquare(int[][] data, int r_data, int c_data) {
		this.data = data;
		this.r_data = r_data;
		this.c_data = c_data;
		// coordenadas
		this.r_zero = 0;
		this.r_half = Math.round(this.r_data / 2) + (r_data % 2);
		this.c_zero = 0;
		this.c_half = Math.round(this.c_data / 2) + (c_data % 2);

		// preenchendo q1
		q1 = new int[r_half][c_half];

		for (int r = r_zero; r < r_half; r++)
			for (int c = c_zero; c < c_half; c++)
				q1[r - r_zero][c - c_zero] = this.data[r][c];

		// preenchendo q2
		q2 = new int[r_half][c_half];

		for (int r = r_zero; r < r_half; r++)
			for (int c = c_half - (c_data % 2); c < c_data; c++)
				q2[r - r_zero][c - c_half + (c_data % 2)] =
this.data[r][c];

		// preenchendo q3
		q3 = new int[r_half][c_half];

		for (int r = r_half - (r_data % 2); r < r_data; r++)
			for (int c = c_half - (c_data % 2); c < c_data; c++)
				q3[r - r_half + (r_data % 2)][c - c_half + (c_data
% 2)] = this.data[r][c];

		// preenchendo q4
		q4 = new int[r_half][c_half];

		for (int r = r_half - (r_data % 2); r < r_data; r++)
			for (int c = c_zero; c < c_half; c++)
				q4[r - r_half + (r_data % 2)][c - c_zero] =
this.data[r][c];

	}

	/** Retorna os dados deste array */
	public int[][] getData() {
		return data;
	}

	/** Retorna o número de linhas */
	public int getRows() {
		return r_data;
	}
```

```
	/** Retorna o número de colunas */
	public int getColumns() {
		return c_data;
	}

	/** Retorna o primeiro quadrante */
	public DataSquare getQ1() {
		return new DataSquare(q1, r_half, c_half);
	}

	/** Retorna o segundo quadrante */
	public DataSquare getQ2() {
		return new DataSquare(q2, r_half, c_half);
	}

	/** Retorna o terceiro quadrante */
	public DataSquare getQ3() {
		return new DataSquare(q3, r_half, c_half);
	}

	/** Retorna o quarto quadrante */
	public DataSquare getQ4() {
		return new DataSquare(q4, r_half, c_half);
	}

	/** Retorna o histograma */
	public int[] getHistogram() {

		int[] h = new int[256];
		for (int r = r_zero; r < r_data; r++)
			for (int c = c_zero; c < c_data; c++)
				h[data[r][c]]++;
		return h;
	}

	/** Lista o conteúdo deste array */
	public void list() {
		int jump = 0;
		for (int r = 0; r < r_data; r++) {
			for (int c = 0; c < c_data; c++) {
				System.out.print(data[r][c] + " ");
				jump++;
				if (jump == c_data) {
					System.out.println();
					jump = 0;
				}
			}
		}
	}

}
```

Quadro 10 - Classe `tcc.process.DataSquare`

3.3.5.6 Extratores

Assim, com as classes apresentadas até agora já temos a estrutura e funcionalidades necessárias para aplicar os extratores de características desejados. Os extratores implementados seguem uma estrutura hierárquica sendo todos subclasses de `tcc.process.extractors.AbstractExtractor`. Esta classe recebe uma imagem como

entrada e fazendo uso da classe `tcc.process.DataSquare` para segmentar os subespaços aplica um determinado extrator em cada subespaço. Ao término do processo o vetor de características para o extrator utilizado é retornado. Esta classe possui o método abstrato `apply()` que é implementado em suas classes especializadas sendo que para cada extrator utilizado o conteúdo deste método é a única coisa a se alterar. Assim, esta estrutura é altamente flexível para implementação de novos extratores. A Figura 17 exibe a estrutura dos extratores implementados.

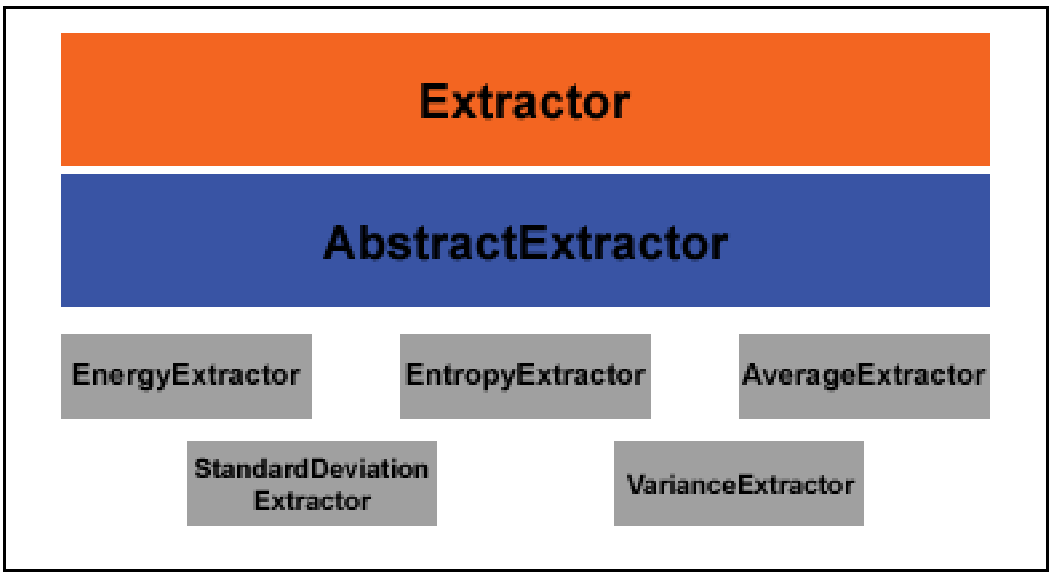


Figura 17 - Pilha de extratores

O Quadro 11 apresenta a classe `tcc.process.extractors.AbstractExtractor` que é responsável por aplicar os extratores em cada subespaço da imagem de entrada. Os atributos **f1** à **f9** contem os subespaços da imagem.

```
package tcc.process.extractors;

import ij.ImagePlus;

import java.util.Arrays;

import tcc.process.DataSquare;

public abstract class AbstractExtractor implements Extractor {

	public double[] extract(ImagePlus image) {

		// Sub espaços wavelets para 3 iterações
		DataSquare f1;
		DataSquare f2;
		DataSquare f3;
		DataSquare f4;
		DataSquare f5;
		DataSquare f6;
		DataSquare f7;
		DataSquare f8;
		DataSquare f9;

		// imagem original
		// passando os parâmetros invertido pois o imageJ forma a matriz de
		// valores de forma invertida
		DataSquare imageData = new DataSquare(image.getProcessor()
				.getIntArray(), image.getWidth(), image.getHeight());

		// primeira iteração
		f1 = imageData.getQ2();
		f2 = imageData.getQ3();
		f3 = imageData.getQ4();

		// segunda iteração
		f4 = imageData.getQ1().getQ2();
		f5 = imageData.getQ1().getQ3();
		f6 = imageData.getQ1().getQ4();

		// terceira iteração
		f7 = imageData.getQ1().getQ1().getQ2();
		f8 = imageData.getQ1().getQ1().getQ3();
		f9 = imageData.getQ1().getQ1().getQ4();

		// calculando o vetor de características
		double[] fv = new double[9];

		fv[0] = apply(f1);
		fv[1] = apply(f2);
		fv[2] = apply(f3);
		fv[3] = apply(f4);
		fv[4] = apply(f5);
		fv[5] = apply(f6);
		fv[6] = apply(f7);
		fv[7] = apply(f8);
		fv[8] = apply(f9);

		System.out.println("EXTRATOR: "+ Arrays.toString(fv));

		return fv;
	}

	/** Aplica o extrator desejado */
	protected abstract double apply(DataSquare data);

}
```

Quadro 11 - Classe `tcc.process.extractors.AbstractExtractor`

3.3.5.7 Classe `CBIRSequence` e `CBIRMeasurements`

A mesma estrutura hierárquica visando a implementação de novos extratores também foi utilizada para implementação da classe que aplica a seqüência de pré-processamento nas imagens. A classe `tcc.process.CBIRSequence` implementa a interface `tcc.process.Sequence` e assim, caso seja necessário implementar novas seqüências de pré-processamento, a estrutura já se encontra pronta. A seqüência de pré-processamento utilizada aplica uma binarização por *threshold* na imagem de entrada e a seguir aplica a transformada *wavelet*. Após a imagem ser pré-processada ela é entregue a classe `tcc.process.CBIRMeasurements` que implementa a interface `tcc.process.Measurements` sendo assim também expansível para futuras implementações. Esta classe realiza a instanciação dos extratores e aplica-os na imagem pré-processada extraindo assim o vetor de característica. As classes `tcc.process.CBIRSequence` e `tcc.process.CBIRMeasurements` podem ser vistas respectivamente nos Quadros 12 e 13.

```
package tcc.process;

import fractsplinewavelets.Filters;
import ij.ImagePlus;

public class CBIRSequence implements Sequence {

      public ImagePlus apply(ImagePlus image) {

            MyThresholder t = new MyThresholder(image);
            t.run("mask");

            FSW_Transform fsw = new FSW_Transform();
            ImagePlus retImage = fsw.transform(image,
                        FSW_Transform.ABSOLUTE_VALUES8, 3, 0.0, -0.49,
                        FSW_Transform.KEEP_MODE, Filters.ORTHONORMAL,
false);

            return retImage;
      }

}
```

Quadro 12 – Classe `tcc.process.CBIRSequence`

```
package tcc.process;

import ij.ImagePlus;
import tcc.main.Constants;
import tcc.process.extractors.AverageExtractor;
import tcc.process.extractors.EnergyExtractor;
import tcc.process.extractors.EntropyExtractor;
import tcc.process.extractors.Extractor;
import tcc.process.extractors.StandardDeviationExtractor;

public class CBIRMeasurements implements Measurements {

	private byte extractor;

	public CBIRMeasurements(byte extractor) {
		this.extractor = extractor;
	}

	public double[] apply(ImagePlus image) {

		Extractor extractor = new EntropyExtractor();
		Extractor extractor2 = new EnergyExtractor();
		Extractor extractor3 = new AverageExtractor();
		Extractor extractor4 = new AverageExtractor();
		Extractor extractor5 = new StandardDeviationExtractor();

		double[] v1 = {};
		double[] v2 = {};
		double[] v3 = {};
		double[] v4 = {};
		double[] v5 = {};

		if (Constants.EXTRACTOR_ENTROPY == this.extractor
				|| Constants.EXTRACTOR_COMBINATED ==
this.extractor) {
			v1 = extractor.extract(image);
		}
		if (Constants.EXTRACTOR_ENERGY == this.extractor
				|| Constants.EXTRACTOR_COMBINATED ==
this.extractor) {
			v2 = extractor2.extract(image);
		}
		if (Constants.EXTRACTOR_AVERAGE == this.extractor
				|| Constants.EXTRACTOR_COMBINATED ==
this.extractor) {
			v3 = extractor3.extract(image);
		}
		if (Constants.EXTRACTOR_VARIANCE == this.extractor
				|| Constants.EXTRACTOR_COMBINATED ==
this.extractor) {
			v4 = extractor4.extract(image);
		}
		if (Constants.EXTRACTOR_STANDARD_DEVIATION == this.extractor
				|| Constants.EXTRACTOR_COMBINATED ==
this.extractor) {
			v5 = extractor5.extract(image);
		}

		double[] rv = new double[v1.length + v2.length + v3.length +
v4.length
				+ v5.length];

		for (int i = 0; i < v1.length; i++) {
			rv[i] = v1[i];
		}

		for (int i = 0; i < v2.length; i++) {
			rv[i + v1.length] = v2[i];
		}
```

```
            for (int i = 0; i < v3.length; i++) {
                  rv[i + v1.length + v2.length] = v3[i];
            }

            for (int i = 0; i < v4.length; i++) {
                  rv[i + v1.length + v2.length + v3.length] = v4[i];
            }

            for (int i = 0; i < v5.length; i++) {
                  rv[i + v1.length + v2.length + v3.length + v4.length] =
v5[i];
            }

            System.out.println("Usando extrator: " + this.extractor + " FV
size = "
                        + rv.length);

            return rv;
      }

}
```

Quadro 13 – Classe `tcc.process.CBIRMeasurements`

3.3.5.8 Classe `Estatistic`

Cada extrator consiste em um cálculo estatístico específico. Como maneira de centralizar estes cálculos em uma só classe, fazendo uma analogia a classe `java.lang.Math` da linguagem JAVA, foi implementada a classe `java.process.Estatistic`. Nesta classe estão todas as funções estatísticas utilizadas pelos extratores. O conteúdo desta classe é exibido no quadro Quadro 14.

```
package tcc.process;

public final class Estatistic {

	private Estatistic() {
		// Construtor privado
	}

	/** Calcula a média de um dado vetor */
	public static double average(int[] data) {

		double Edata = 0;
		double result = 0;
		double pI = 0;

		// Soma todo o vetor
		for (int i = 0; i < data.length; i++) {
			Edata += data[i];
		}

		for (int i = 0; i < data.length; i++) {
			pI = (double) data[i] / Edata;
			result += data[i] * pI;
		}

		return result;
	}

	/** Calcula a média de um dado vetor */
	public static double average(double[] data) {

		double Edata = 0;
		double result = 0;
		double pI = 0;

		// Soma todo o vetor
		for (int i = 0; i < data.length; i++) {
			Edata += data[i];
		}

		for (int i = 0; i < data.length; i++) {
			pI = (double) data[i] / Edata;
			result += data[i] * pI;
		}

		return result;
	}

	/** Calcula a energia de um dado vetor */
	public static double energy(int[] data) {

		double Edata = 0;
		double result = 0;
		double pI = 0;

		// Soma todo o vetor
		for (int i = 0; i < data.length; i++) {
			Edata += data[i];
		}

		for (int i = 0; i < data.length; i++) {

			pI = (double) data[i] / Edata;

			result += pI * pI;
		}

		return result;
	}
```

```
/** Calcula a energia de um dado vetor */
public static double energy(double[] data) {

	double Edata = 0;
	double result = 0;
	double pI = 0;

	// Soma todo o vetor
	for (int i = 0; i < data.length; i++) {
		Edata += data[i];
	}

	for (int i = 0; i < data.length; i++) {

		pI = (double) data[i] / Edata;

		result += pI * pI;
	}

	return result;
}

/** Calcula a entropia de um dado vetor */
public static double entropy(int[] data) {

	double Edata = 0;
	double result = 0;
	double pI = 0;

	// Soma todo o vetor
	for (int i = 0; i < data.length; i++) {
		Edata += data[i];
	}

	for (int i = 0; i < data.length; i++) {

		pI = (double) data[i] / Edata;

		if (pI <= 0)
			continue;

		result += pI * log2(pI);
	}

	return -result;
}

/** Calcula a entropia de um dado vetor */
public static double entropy(double[] data) {

	double Edata = 0;
	double result = 0;
	double pI = 0;
	// Soma todo o vetor
	for (int i = 0; i < data.length; i++) {
		Edata += data[i];
	}

	for (int i = 0; i < data.length; i++) {

		pI = (double) data[i] / Edata;

		if (pI <= 0)
			continue;

		result += pI * log2(pI);
	}
```

```
            return -result;
      }

      /** Calcula a variância de um dado vetor */
      public static double variance(int[] data) {
            double Edata = 0;
            double result = 0;
            double pI = 0;
            // Média do vetor
            double average = average(data);

            // Soma todo o vetor
            for (int i = 0; i < data.length; i++) {
                  Edata += data[i];
            }

            for (int i = 0; i < data.length; i++) {
                  pI = (double) data[i] / Edata;
                  result += Math.pow(data[i] - average, 2) * pI;
            }

            return result;
      }

      /** Calcula a variância de um dado vetor */
      public static double variance(double[] data) {
            double Edata = 0;
            double result = 0;
            double pI = 0;
            // Média do vetor
            double average = average(data);

            // Soma todo o vetor
            for (int i = 0; i < data.length; i++) {
                  Edata += data[i];
            }

            for (int i = 0; i < data.length; i++) {
                  pI = (double) data[i] / Edata;
                  result += Math.pow(data[i] - average, 2) * pI;
            }

            return result;
      }

      /** Calcula o desvio padrão de um dado vetor */
      public static double standardDeviation(int[] data) {
            return Math.sqrt(variance(data));
      }

      /** Calcula o desvio padrão de um dado vetor */
      public static double standardDeviation(double[] data) {
            return Math.sqrt(variance(data));
      }

      /** Calcula a simetria de um dado vetor */
      public static double symmetry(double[] data) {
            double Edata = 0;
            double result = 0;
            double pI = 0;
            // Média do vetor
            double average = average(data);

            // Soma todo o vetor
            for (int i = 0; i < data.length; i++) {
                  Edata += data[i];
            }

            for (int i = 0; i < data.length; i++) {
```

```
                  pI = (double) data[i] / Edata;
                  result += Math.pow(data[i] - average, 3) * pI;
            }
            return Math.pow(standardDeviation(data), -3) * result;
      }

      /** Calcula o achatamento de um dado vetor */
      public static double kurtosis(double[] data) {

            double Edata = 0;
            double result = 0;
            double pI = 0;
            // Média do vetor
            double average = average(data);

            // Soma todo o vetor
            for (int i = 0; i < data.length; i++) {
                  Edata += data[i];
            }

            for (int i = 0; i < data.length; i++) {
                  pI = (double) data[i] / Edata;
                  result += Math.pow(data[i] - average, 4) * pI - 3;
            }

            return Math.pow(standardDeviation(data), -4) * result;
      }
      /** Calcula log na base 2 de um dado valor */
      public static double log2(double d) {
            return (double) Math.log10(d) / Math.log10(2.0);
      }
}
```

Quadro 14 - Classe `Estatistic`

3.3.6 Operacionalidade da implementação

A interface com o usuário é muito simples de ser operada uma vez que o objetivo do trabalho era fazer um estudo e implementar uma busca por similaridade e não um visualizador de imagens DICOM com recursos avançados. Inicialmente é apresentada ao usuário uma interface como a da Figura 18.

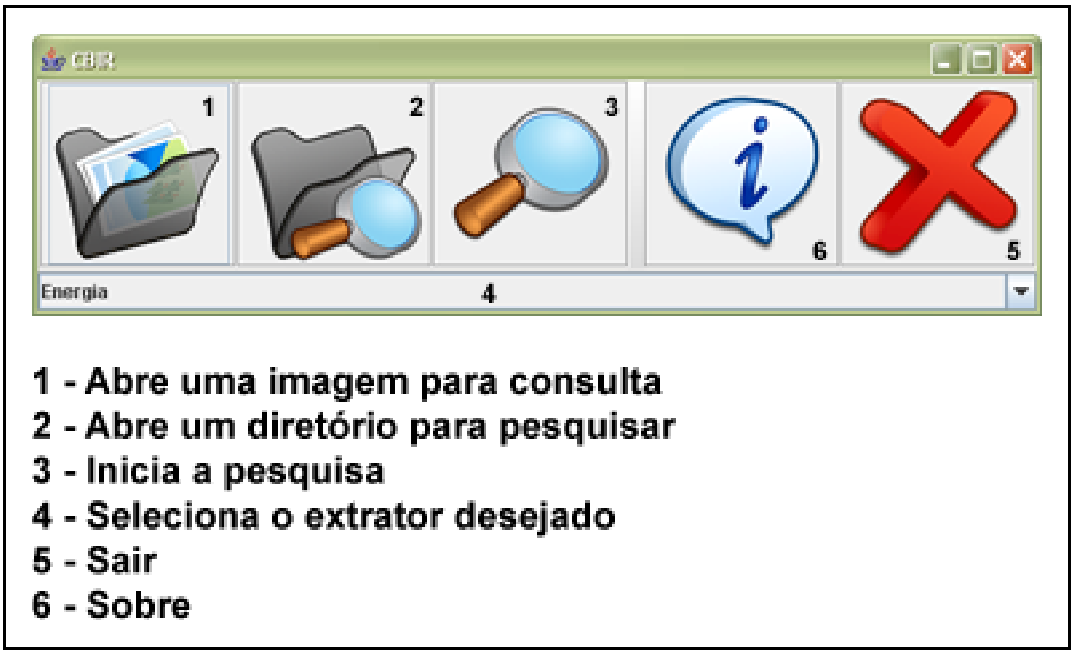


Figura 18 - Interface gráfica

Está é a interface de controle principal do protótipo. Nela é possível selecionar a imagem desejada para iniciar a busca, selecionar o diretório onde o sistema vai pesquisar e disparar a busca pela imagem. A opção descrita com o número 1 na imagem acima permite ao usuário navegar entre seus arquivos de imagens DICOM e selecionar qual a imagem que será utilizada como imagem base da busca.

Uma vez selecionada a imagem o próximo passo é selecionar qual será o diretório com a base de imagens. Para isso o usuário clica o botão com o índice de número 2. O sistema exibe uma tela de navegação de arquivos e permite ao usuário selecionar somente diretórios. O usuário então, seleciona o diretório que possui as imagens para pesquisa. Após esta etapa o protótipo está pronto para realizar a pesquisa bastando apenas clicar no botão de índice número 3. O sistema então começa a pesquisa no diretório selecionado e dentro de alguns segundos exibe as imagens ordenadas por nível de similaridade. Um exemplo da tela de respostas é exibido na imagem a seguir (Figura 19). Nesta tela é permitido selecionar o total de imagens de resposta desejado e a escala de visualização destas imagens.

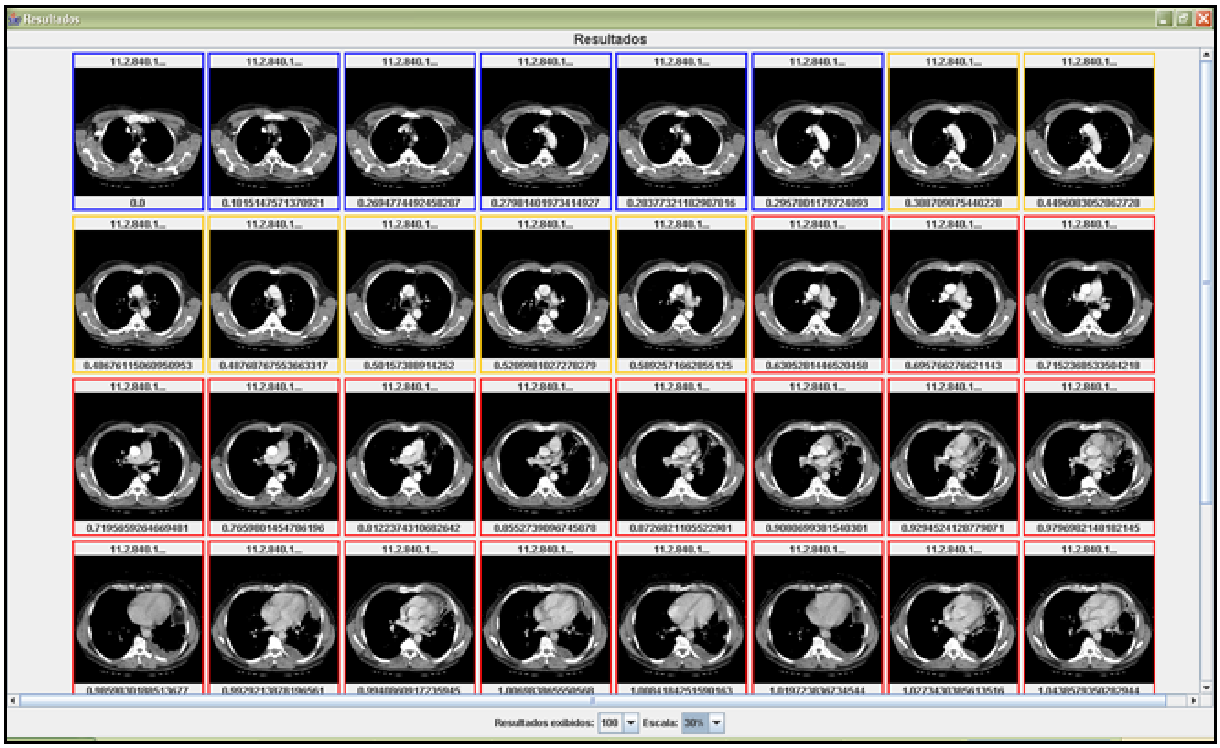


Figura 19 - Interface gráfica dos resultados

Nesta interface o usuário pode passar o mouse sobre uma das imagens de retorno e poderá perceber que o cursor alterna para um ícone de manipulação de conteúdo. Assim é possível dar um duplo clique sobre uma das imagens e selecioná-la para ser a imagem de base para uma nova consulta. Ao clicar a imagem aparece onde antes era a imagem de pesquisa e para realizar novamente a pesquisa o usuário clica novamente no botão de busca de índice 3.

3.4 RESULTADOS E DISCUSSÃO

Os resultados obtidos durante a fase de testes foram muito satisfatórios quanto a classificação das imagens. Como dito anteriormente, foi levado em consideração a semelhança entre as imagens dos exames radiológicos como forma de classificação dos grupos semânticos. A busca realizada a partir de uma imagem inicial deve retornar imagens similares pertencentes ao mesmo grupo semântico, como por exemplo, cortes de abdômen, cabeça, tórax, bacia, etc. Para os testes foram utilizado duas bases de dados com diferentes características. Uma das bases possuía um número reduzido de imagens com características semelhantes. E outra base possuía um número mais elevado de imagens para atestar a

eficiência com grandes quantidades de imagens. A Tabela 1 detalha as bases utilizadas.

Tabela 1 – Imagens utilizadas nos testes

BANCO DE IMAGENS DICOM					
	Qtd	**Tamanho**	**Tipos de imagens**	**Série**	**Não série**
Base 1	57	512x512	Cabeça e abdômen	X	-
Base 2	458	512x512	Cabeça, abdômen, outros	X	X

A base 1 possui somente cortes de cabeça e abdômen em séries provenientes do mesmo exame. Esta base foi basicamente utilizada para analisar a dispersão dos resultados e constatar a classificação que o protótipo realizava entre as duas classes semânticas de imagens utilizadas. Já a base 2 possui imagens provenientes de vários exames e além de imagens de cabeça, abdômen, tórax, e outras.

Para a classificação de quão similar são as imagens foi utilizado a métrica de distância euclidiana. A representação gráfica da classificação feita por esta distância para os vetores de características gerados na base 1 pode ser vista na Figura 20. A imagem base utilizada foi uma imagem de cabeça e foi comparada em relação a todas as outras imagens de cabeça e abdômen da base 1. Neste gráfico é possível ver claramente que as imagens de cabeça ficam mais próximas da imagem base e as de abdômen ficam ao seu redor.

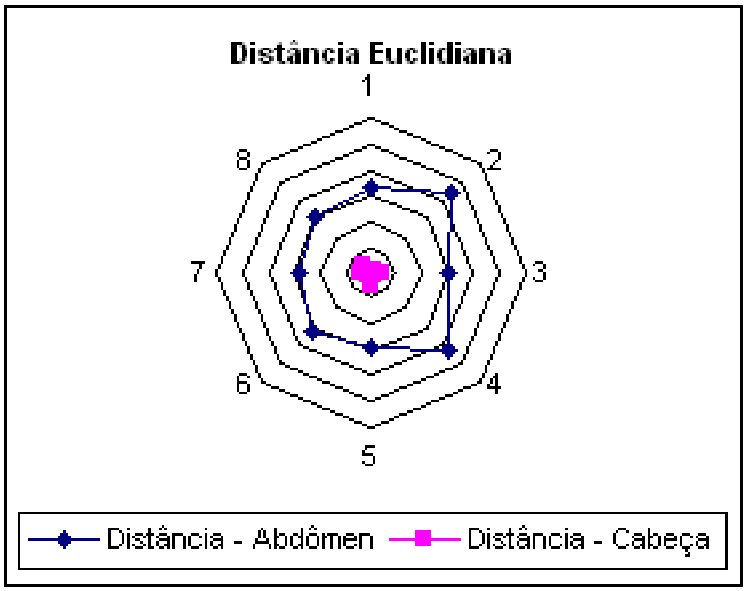


Figura 20 - Distância Euclidiana em relação a uma imagem de cabeça

Os vetores de características calculados sobre a base 1 foram analisados por meio de gráficos e foi constatado que os valores calculados apresentavam um padrão nas imagens analisadas. Este dado é muito importante, pois representa que os extratores conseguiram uma

boa dispersão das classes semânticas. A Figura 21 exibe os gráficos gerados para os vetores de características da base 1.

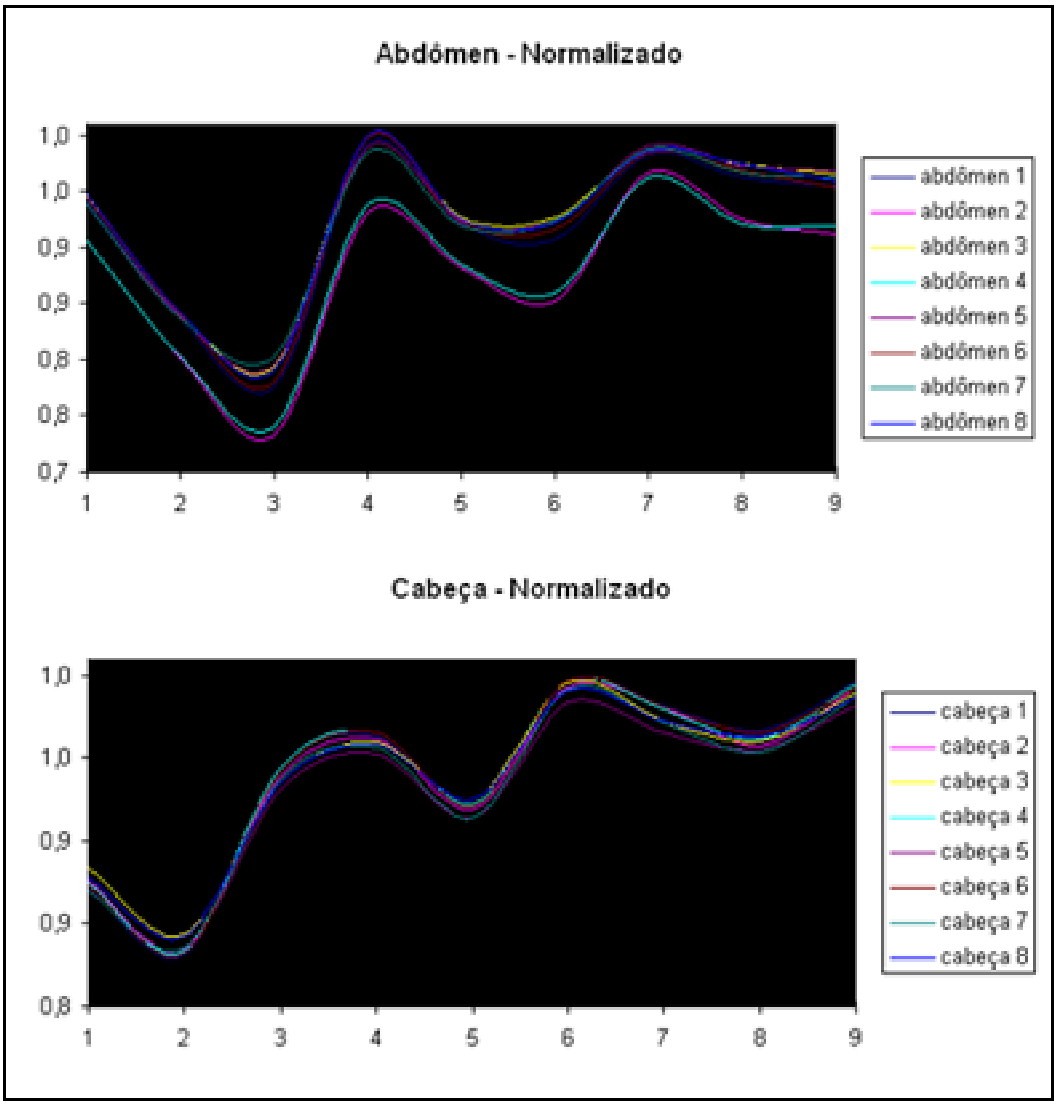


Figura 21 - Gráfico de padrões

Pode-se notar que para as 8 imagens de cabeça analisadas a curva gerada é bastante semelhante. Já para as imagens de abdômen ouve casos em que as curvas mantiveram o mesmo padrão porem em um nível deslocado das demais imagens.

Quanto ao desempenho da pesquisa e da geração dos vetores de características, o tempo de processamento foi otimizado em aproximadamente 5 vezes ao utilizar-se processos paralelos. Em testes realizados fazendo o uso de uma abordagem síncrona, ou seja, utilizando 1 processo por vez o tempo obtido foi perto de 1 segundo por imagem. Porém a utilização de um número maior de processos tem como limitação o poder de processamento e a quantidade

de memória disponível nesta máquina. Para os testes foi utilizada uma máquina com processador Pentium 4 de 3.3GHz HT e 512MB de memória RAM. Abaixo são exibidas algumas telas de pesquisas realizadas sobre as bases de dados utilizando os extratores implementados e métrica de distância euclidiana.

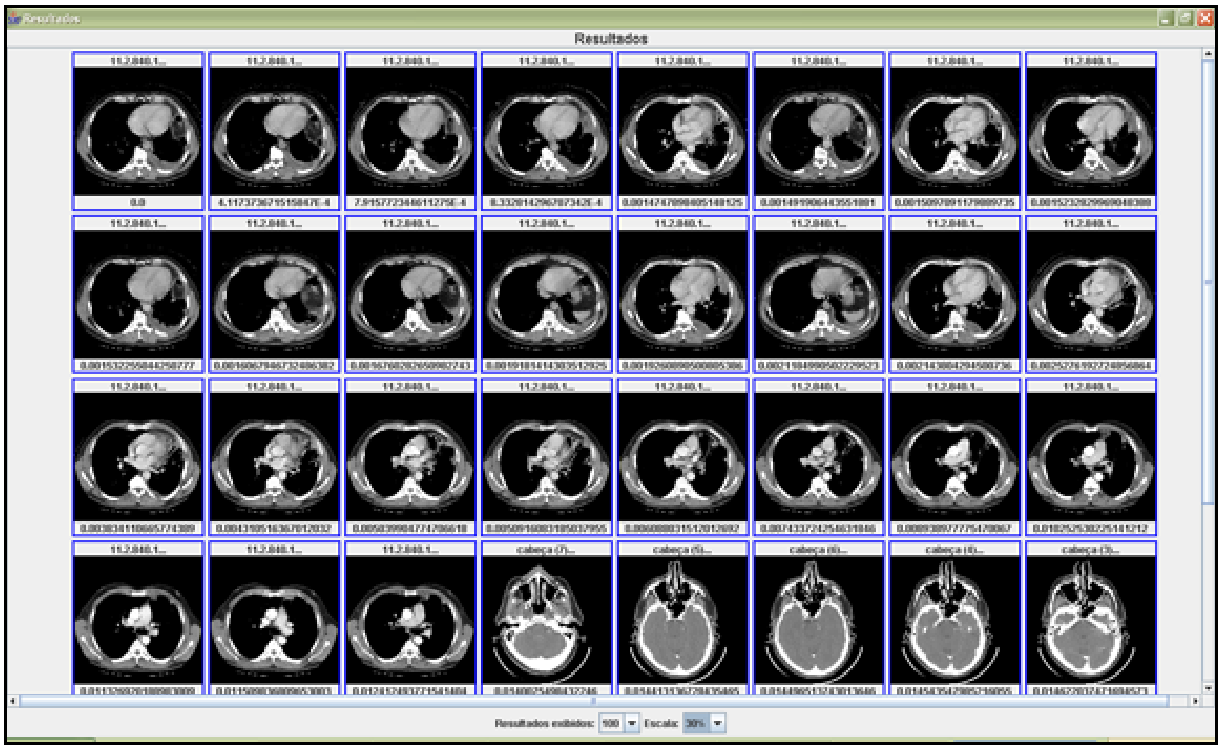


Figura 22 - Resultado utilizando extrator energia

Pode-se observar que na Figura 22 os resultados obtidos apresentam uma interseção entre os diferentes tipos de cortes das imagens, ou seja, para uma imagem base de *slice* de abdômen obteve-se com resposta algumas imagens de *slices* de cabeça. Isso ocorre pois o vetor de características calculado pelo extrator energia não consegue obter um nível de dispersão ao se utilizar distância euclidiana para obter a similaridade com a imagem base.

O gráfico da Figura 20 pode ser utilizado para compreender esta situação. Neste gráfico temos as imagens de cabeça mais ao centro e os cortes de abdômen mais afastados dos pontos que representam as *slices* de cabeça. O que ocorre no extrator energia ou em outros que apresentam resultados semelhantes é que, por exemplo, um ponto representando uma imagem de abdômen é plotado junto ao grupo de imagens de cabeças. Isso ocorre pois os vetores de características das imagens conflitantes não foram calculados com um extrator que consiga obter uma boa classificação semântica das imagens.

Abaixo são exibidos outros resultados para a mesma imagem base utilizando outros

extratores. Pode-se notar que o extrator de entropia foi o que apresentou os melhores resultados para a busca realizada. Este extrator utilizado em conjunto com a métrica de distância euclidiana apresenta um bom nível de dispersão entre os diferentes grupos semânticos de imagens que constituem a base de dados utilizada.

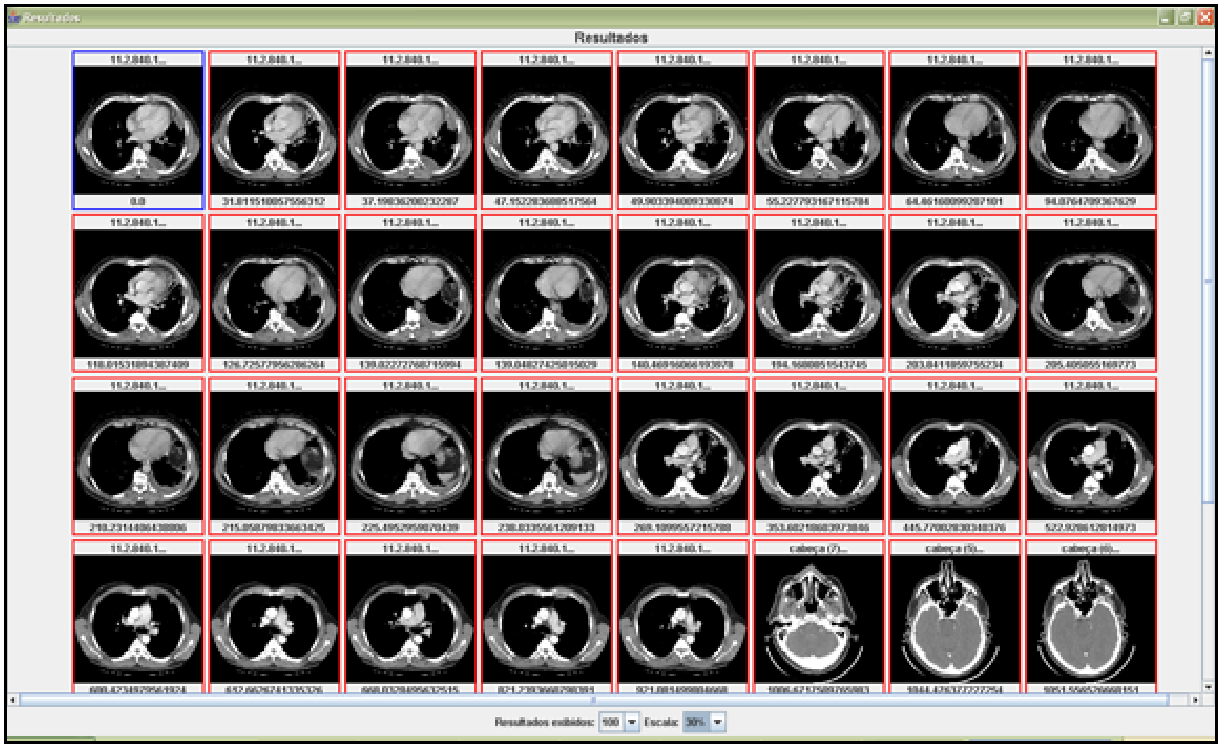


Figura 23 - Resultado utilizando extrator variância

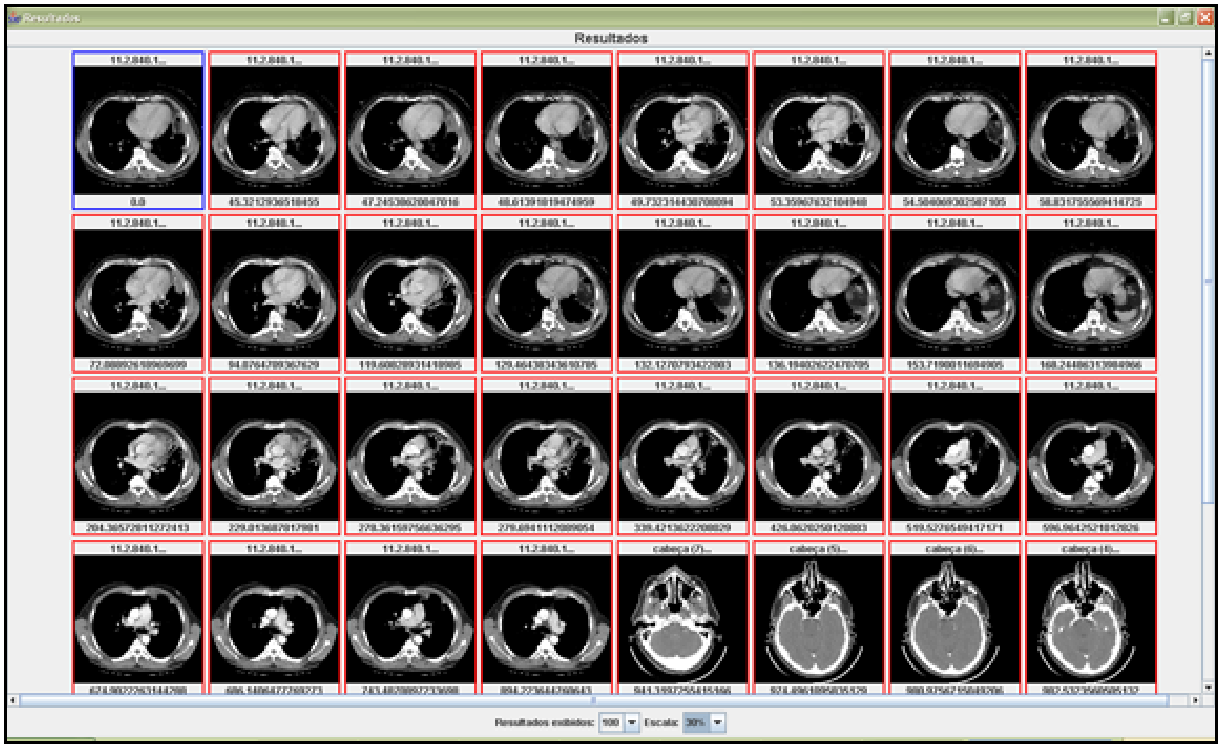


Figura 24 - Resultado utilizando extrator média

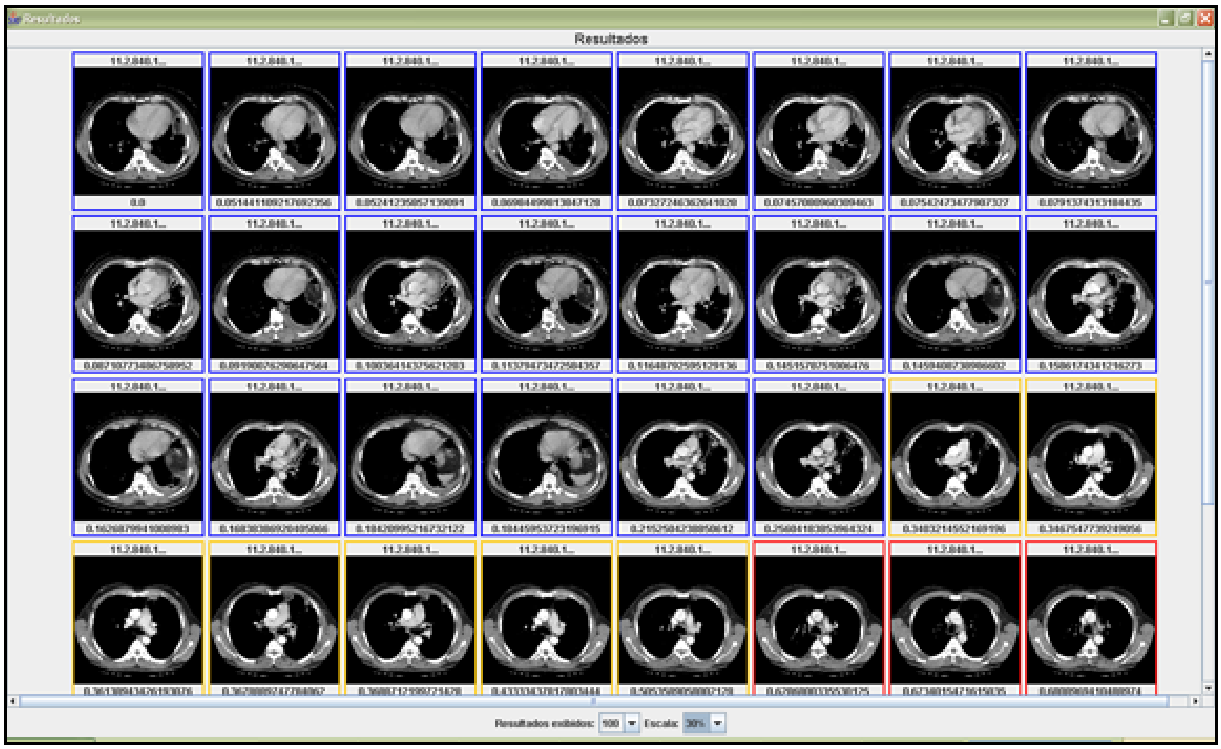


Figura 25 - Resultado utilizando extrator entropia

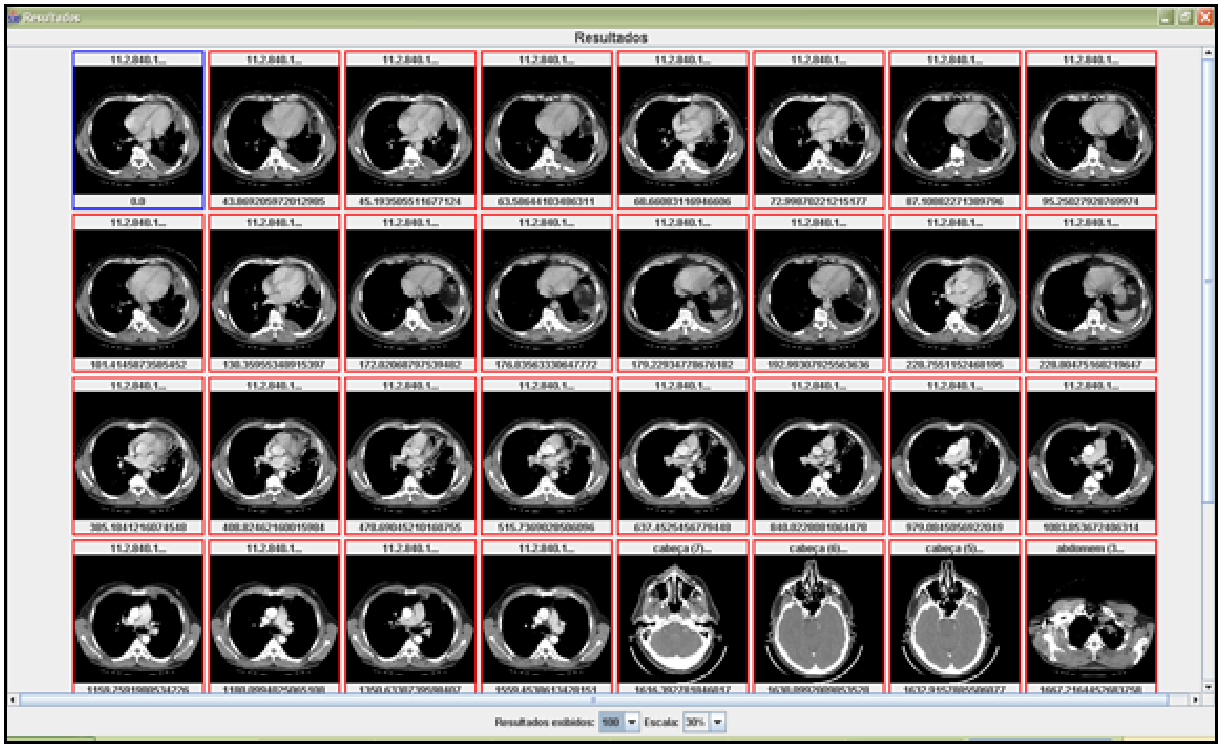


Figura 26 - Resultado utilizando extrator desvio padrão

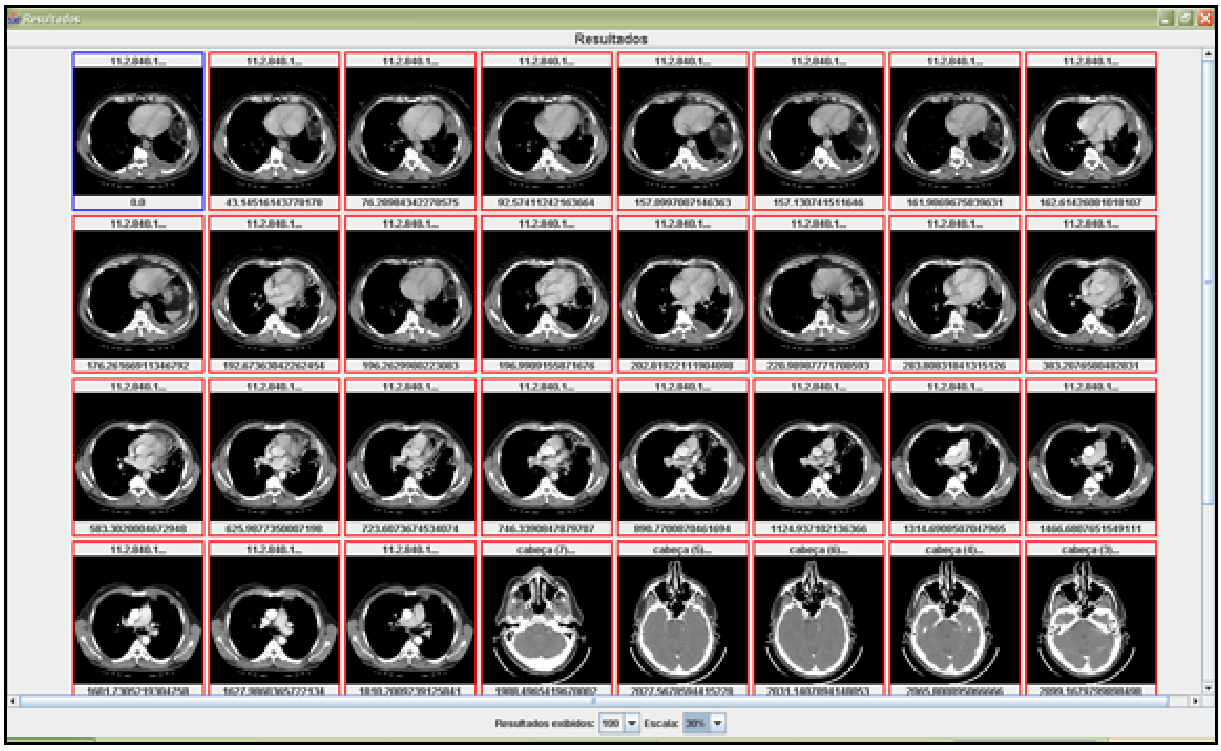


Figura 27 - Resultado utilizando extratores combinados

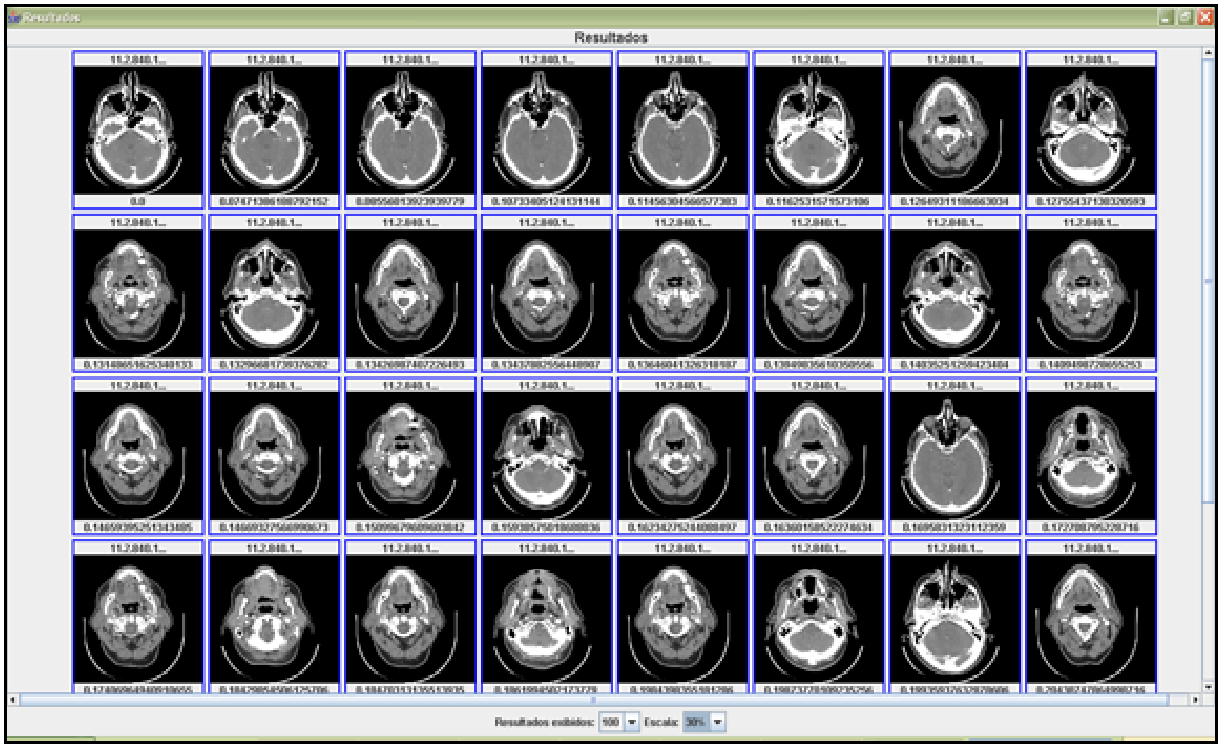


Figura 28 - Resultado utilizando entropia por imagem fora da base 2

Em relação aos trabalhos correlatos utilizados como base para a realização deste trabalho podemos ter a seguinte tabela de características (Tabela 2).

Tabela 2 - Trabalhos correlatos

TRABALHOS CORRELATOS					
	Wavelet	**Métrica de Distância**	**Pré-Processamento**	**ROI**	**Formato DICOM**
Este protótipo (2007)	*Fractional Spline Wavelet*	Distância Euclidiana	*Threshold* e máscara binária	Sim	Sim
Campo (2002)	-	Histograma Métrico	-	-	Sim
Hoppe (2005)	-	-	-	-	-
Castañón (2003)	*Wavelet* Gabor	Distância Euclidiana Normalizada	Ajustes de contraste e brilho	-	Sim

Estes trabalhos foram utilizados como base de consulta e apoio para implementação deste protótipo. O trabalho de Castañón (2003) foi utilizado como base dos estudos com *wavelet* para extração de características. Neste trabalho Castañón utilizou a transformada de Gabor como base para a implementação e alguns ajustes de pré-processamento conforme Tabela 2. Somente o trabalho de Hoppe (2005) não possui suporte a imagens Dicom, porém foi utilizado como base aos estudos de busca por similaridade. O projeto desenvolvido por Campo (2002) foi utilizado para estudos com métricas de distância optando-se por distância Euclidiana como métrica para este projeto. A aplicação da máscara binária neste trabalho mostrou resultado satisfatório por normalizar as imagens antes da aplicação da transformada wavelet. Durante o desenvolvimento buscou-se extrair as características mais relevantes de cada trabalho que ajudariam na conclusão deste projeto.

4 CONCLUSÕES

O presente trabalho teve como proposta o desenvolvimento de um protótipo de um sistema de busca de imagens médicas baseada no conteúdo. Durante o desenvolvimento deste trabalho muitas dificuldades foram encontradas e superadas. Dentre elas o fato da documentação da API de manipulação de imagens ImageJ ser bastante precária. Mesmo assim, o fato de ser uma API livre levou o autor ao estudo de suas funcionalidades e métodos de utilização. Outra dificuldade encontrada foi quanto ao *plugin* responsável pela aplicação da transformada *wavelet*. Este *plugin* foi construído para trabalhar originalmente junto a API ImageJ não sendo possível utilizá-lo diretamente no código do protótipo de forma eficaz. Assim foi necessário a adaptação de uma classe específica para conseguir utilizá-lo.

É interessante ressaltar que todo o desenvolvimento do trabalho pôde ser feito com software livre, desde as IDEs de desenvolvimento e sistemas operacionais utilizados, até as ferramentas de manipulação de imagens e *plugins*. Todas as ferramentas utilizadas conseguiram realizar seu papel da maneira como descrita em suas documentações.

Os objetivos previstos para este trabalho foram todos alcançados:

a) efetuar o pré-processamento das imagens utilizando os algoritmos de pré-processamento. Realizar os ajustes e normalizações das imagens necessárias antes da extração das características principais;
b) extrair os vetores de características das imagens, ou seja, retirar através do uso de um extrator um conjunto de valores que define a imagem desejada;
c) determinar o grau de similaridade das imagens utilizando a comparação dos vetores de características;
d) possuir uma interface gráfica que permita ao usuário selecionar qual imagem será utilizada como base para a pesquisa;
e) exibir as imagens resultantes da busca na interface gráfica.

Esperava-se mais quanto ao desempenho do processamento das imagens. Durante a fase de análise não foi observado o custo que o pré-processamento de uma imagem possui. Isto se deve principalmente a aplicação da transformada *wavelet* e a segmentação dos subespaços, ficando então uma proposta a ser superada em futuras implementações.

A abordagem utilizada neste trabalho levanta questões a serem consideradas quanto ao futuro dos sistemas de busca convencionais. Focado somente em imagens médicas os resultados já são bastante satisfatórios e nos faz pensar que podemos evoluir para imagens

cada vez mais genéricas.

> É fácil encontrar a posição da cadeia “gato”, se existir, em um documento - qualquer editor de textos oferece esse recurso. Agora, considere o problema de encontrar o subconjunto de pixels em uma imagem que corresponde à imagem de um gato. Se tivéssemos essa capacidade, poderíamos responder a consultas referentes a imagens, como “Bill Clinton e Nelson Mandela juntos”, “um patinador no ar”, “a torre Eiffel à noite” e assim por diante, sem termos de digitar palavras-chaves de legendas relativas a cada fotografia em uma coleção. (RUSSELL; NORVIG, 2004, p. 855).

Esta evolução chegará ao ponto em que poderemos utilizar a busca por similaridade em nossas pesquisas na internet, em nossos computadores e em todo o lugar em que se faça necessária.

4.1 EXTENSÕES

Possíveis extensões para este trabalho podem ser realizadas. Uma abordagem da qual não se tratou neste projeto seria a criação de um módulo que interagisse diretamente com um banco de dados onde os vetores de características estariam armazenados junto a imagem. Outra extensão interessante seria a ampliação da pesquisa a fim de realizar buscas por similaridade em imagens de tamanhos diferentes pois a abordagem atual não apresenta bons resultados quando as imagens da base pesquisada possuem tamanhos diferenciados. Seguindo a mesma linha dos CBIRs e da WEB 2.0 (O’REILLY, 2005) já aproveitando o fato do protótipo não estar amarrado a interface gráfica seria muito interessante a criação de um sistema baseado neste trabalho utilizando arquitetura cliente servidor podendo executar em um navegador de páginas.

REFERÊNCIAS BIBLIOGRÁFICAS

ACCUSOFT. **Rapidly discover, design and deliver in a visual programming environment.** [S.l.], 2006. Disponível em: <http://www.accusoft.com/products/visiquest/>. Acesso em: 18 set. 2006.

BIOMEDICAL IMAGING GROUP. **Fractional splines wavelet**. [Lausanne], 2007. Disponível em: <http://bigwww.epfl.ch/>. Acesso em: 20 maio 2007.

CAMPO, Camilo Y. **Ampliando o poder de recuperação de imagens por conteúdo utilizando histogramas adaptados**: aplicações em imagens médicas. São Paulo: USP/ICMC, 2002. Disponível em: <http://netuno.icmc.usp.br/pn/files/2004/FabioChino_Agma.pdf >. Acesso em: 14 set. 2006.

CASTAÑÓN, César A. B. **Recuperação de imagens por conteúdo através de análise multiresolução por wavelets.** 2003. 95 f. Dissertação (Mestrado em Ciências da Computação e Matemática Computacional) - Centro de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos. Disponível em: <http://netuno.icmc.usp.br/pn/files/2004/CesarCastanon_Agma.pdf >. Acesso em: 14 set. 2006.

ECLIPSE FOUNDATION. **Eclipse main page**. [Ottawa], 2007. Disponível em: <http://www.eclipse.org/>. Acesso em: 23 maio 2007.

FACON, Jacques. **Processamento e análise de imagens**. Embalse: EBAI, 1993.

HOPPE, Aurélio F. **Recuperação distribuída de imagens por similaridade.** 2005. 79 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

JORDÁN, Ramiro; LOTUFO, Roberto. **Digital image processing (DIP) with Khoros 2.** [Florianópolis], 1999. Disponível em: <http://www.inf.ufsc.br/~visao/khoros/>. Acesso em: 9 set. 2006.

O'REILLY. **What is web 2.0.** [S.l.], 2005. Disponível em: <http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html>. Acesso em: 23 maio 2007.

RADIOLOGICAL SOCIETY OF NORTH AMERICA. **The value and importance of an imaging standard**. [S.l.], 2006. Disponível em: <http://www.rsna.org/Technology/DICOM/>. Acesso em: 12 set. 2006.

RESEARCH SERVICES BRANCH. **ImageJ API**. [S.l.], 2007. Disponível em: <http://rsb.info.nih.gov/ij/developer/api/index.html>. Acesso em: 23 maio 2007.

RUSSELL, Stuart J.; NORVIG, Peter. **Inteligência artificial.** Tradução Vandenberg D. de Souza. Rio de Janeiro: Campus, 2004.

SILVA, Alex S. **Desenvolvimento de metodologias para a detecção automatizada de pontos de referência anatômicos (landmarks) em volumes tomográficos cerebrais humanos.** 2002. 113 f. Dissertação (Mestrado em Ciências da Computação) - Universidade Federal de Santa Catarina, Florianópolis.

_____. **Protótipo de software para classificação de impressão digital.** 1999. 70 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SILVA, Aristófanes C. **Algoritmos para diagnóstico assistido de nódulos pulmonares solitários em imagens de tomografia computadorizada**. 2004. 140 f. Tese de Doutorado (Doutor em Informática) - Programa de Pós-graduação em Informática do Departamento de Informática da PUC - Rio, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro. Disponível em: <http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0016025_04_cap_02.pdf>. Acesso em: 18 set. 2006.

STELTING, Stephen; MAASSEN, Olav. **Applied Java patterns.** Palo Alto: Sun Microsystems Press, 2002.

WANGENHEIM, Aldo V. **Morfologia matemática.** [Florianópolis], 2007. Disponível em: <http://www.inf.ufsc.br/~visao/morfologia.pdf>. Acesso em: 23 maio. 2007.