UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

TIAGO DIAS CARVALHO DO NASCIMENTO

SEGMENTAÇÃO NÃO SUPERVISIONADA DE TEXTURAS BASEADA NO ALGORITMO PPM

João Pessoa
2010

TIAGO DIAS CARVALHO DO NASCIMENTO

SEGMENTAÇÃO NÃO SUPERVISIONADA DE TEXTURAS BASEADA NO ALGORITMO PPM

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal da Paraíba como parte dos requisitos para a obtenção do título de Mestre em Informática.

ORIENTADOR: Prof. Dr. Leonardo Vidal Batista

João Pessoa
2010

Ata da Sessão Pública de Defesa de Dissertação de Mestrado do TIAGO DIAS CARVALHO DO NASCIMENTO, candidato ao Título de Mestre em Informática na Área de Sistemas de Computação, realizada em 26 de março de 2010.

Aos vinte e seis dias do mês de março do ano dois mil e dez, às nove horas, na Sala do REUNI do Centro de Ciências Exatas e da Natureza da Universidade Federal da Paraíba, reuniram-se os membros da Banca Examinadora constituída para examinar o candidato ao grau de Mestre em Informática, na área de "*Sistemas de Computação*", na linha de pesquisa "*Processamento de Sinais e Sistemas Gráficos*", o Sr. Tiago Dias Carvalho do Nascimento. A comissão examinadora composta pelos professores doutores: Leonardo Vidal Batista (DI - UFPB), Orientador e Presidente da Banca Examinadora, Ronei Marcos de Moraes (DI-UFPB) e Herman Martins Gomes (UFCG), como examinador externo. Dando início aos trabalhos, o Prof. Leonardo Vidal Batista, cumprimentou os presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato para que o mesmo fizesse, oralmente, a exposição do trabalho de dissertação intitulado "*SEGMENTAÇÃO NÃO SUPERVISIONADA DE TEXTURAS BASEADA NO ALGORITMO PPM* ". Concluída a exposição, o candidato foi argüido pela Banca Examinadora que emitiu o seguinte parecer: "*aprovado*". Assim sendo, deve a Universidade Federal da Paraíba expedir o respectivo diploma de Mestre em Informática na forma da lei e, para constar, a professora Tatiana Aires Tavares, Sra. Coordenadora do PPGI, lavrou a presente ata, que vai assinada por ela, e pelos membros da Banca Examinadora. João Pessoa, 26 de março de 2010.

Tatiana Aires Tavares

Prof. Dr. Leonardo Vidal Batista
Orientador (DI-UFPB)

Prof. Dr. Ronei Marcos de Moraes
Examinador Local (DI-UFPB)

Prof. Dr. Herman Martins Gomes
Examinador Externo (UFCG)

AGRADECIMENTOS

Agradeço aos meus pais, acima de tudo, por todo o amor e apoio que me deram durante toda a minha vida, ao meu orientador, por sempre acreditar que esse trabalho daria certo e pela amizade que construímos desde os tempos da graduação, a Guido e aos amigos do meu grupo de pesquisa (em especial os do PET) e do Lavid (em especial os da Linear), por terem sido agradáveis companhias tanto nas festas quanto nas horas de trabalho.

RESUMO

O problema da segmentação de imagens está presente em diversas tarefas como sensoriamento remoto, detecção de objetos em robótica, automação industrial, recuperação de imagens por conteúdo, segurança, e outras relacionadas à medicina. Quando há um conjunto de padrões pré-classificados, a segmentação é denominada supervisionada. No caso da segmentação não supervisionada, as classes são extraídas diretamente dos padrões. Dentre as propriedades de uma imagem, a textura está entre as que proporcionam os melhores resultados no processo de segmentação. Este trabalho propõe um novo método de segmentação não supervisionada de texturas que utiliza como medida de similaridade entre regiões as taxas de bits resultantes da compressão utilizando modelos produzidos pelo algoritmo *Prediction by Partial Matching* (PPM) extraídos das mesmas. Para segmentar uma imagem, a mesma é dividida em regiões retangulares adjacentes e cada uma delas é atribuída a um grupo distinto. Um algoritmo aglomerativo guloso, que une os dois grupos mais próximos em cada iteração, é aplicado até que o número de grupos seja igual ao número de classes (fornecido pelo usuário). Na etapa seguinte, cujo objetivo é refinar a localização das fronteiras, a imagem é dividida em regiões ainda menores, as quais são atribuídas ao agrupamento cujo modelo PPM resulta na taxa de bits mais baixa. Para avaliar o método proposto, foram utilizados três bancos de imagens: o de Trygve Randen, o de Timo Ojala e um criado pelo autor deste trabalho. Ajustando-se os parâmetros do método para cada imagem, a taxa de acerto obtida foi em torno de 97% na maioria dos casos e 100% em vários deles. O método proposto, cuja principal desvantagem é a ordem de complexidade, se mostrou robusto a regiões de diferentes formas geométricas, agrupando corretamente até mesmo as desconexas.

Palavras-chave. Segmentação não supervisionada, Análise de texturas, PPM, Algoritmo aglomerativo guloso, Agrupamento.

ABSTRACT

The image segmentation problem is present in various tasks such as remote sensing, object detection in robotics, industrial automation, content based image retrieval, security, and others related to medicine. When there is a set of pre-classified data, segmentation is called supervised. In the case of unsupervised segmentation, the classes are extracted directly from the data. Among the image properties, the texture is among those that provide the best results in the segmentation process. This work proposes a new unsupervised texture segmentation method that uses as the similarity measure between regions the bit rate obtained from compression using models, produced by the Prediction by Partial Matching (PPM) algorithm, extracted from them. To segment an image, it is split in rectangular adjacent regions and each of them is assigned to a different cluster. Then a greedy agglomerative clustering algorithm, in which the two closest clusters are grouped at every step, is applied until the number of remaining clusters is equal to the number of classes (supplied by the user). In order to improve the localization of the region boundaries, the image is then split in shorter regions, that are assigned to the cluster whose PPM model results in lower bit rate. To evaluate the proposed method, three image set were used: Trygve Randen, Timo Ojala and one created by the author of this work. By adjusting the method parameters for each image, the hit rate obtained was around 97% in most cases and 100% in several of them. The proposed method, whose main drawback is the complexity order, is robust to regions with different geometric shapes, grouping correctly even those that are disconnected.

Keywords. Unsupervised segmentation, Texture analysis, PPM, Greedy agglomerative algorithm, Clustering.

SUMÁRIO

1. INTRODUÇÃO

Visão Computacional é uma área de pesquisa que tem como objetivo extrair, a partir de um conjunto de imagens, informações [1] que serão úteis para executar uma tarefa, entender melhor o ambiente ou adquirir conhecimento [2].

A cor, a forma, a textura e as relações espaciais são propriedades comumente utilizadas para descrever de forma quantitativa a informação visual [3][4]. Algumas delas, ou todas, podem ser utilizadas em conjunto [5][6][7] para analisar uma imagem, no entanto, a textura está entre as que proporcionam melhores resultados [8][9].

Apesar de ser uma propriedade importante, não há um consenso na descrição do que é, exatamente, uma textura [10]. Para Faugeras e Pratt [11], textura é um "padrão local básico que é periodicamente ou quase periodicamente repetido em alguma área". De forma semelhante, Jain e Karu [12] afirmam que textura não é caracterizada apenas pela cor do *pixel*, mas também pelo padrão encontrado nos *pixels* que o cercam. Mais discussões sobre o que é textura podem ser encontradas no trabalho de Karu, Jain e Bolle [13].

De acordo com Santamaria *et al.* [14], os algoritmos de análise de texturas podem ser divididos em quatro categorias: (i) técnicas baseadas em métodos estatísticos, nas quais a textura é caracterizada pela distribuição estatística de valores numa região de interesse; (ii) métodos geométricos, onde a textura é caracterizada por atributos primitivos e suas posições no espaço; (iii) métodos baseados em modelo, em que a textura é descrita por um modelo matemático; e (iv) métodos de processamento de sinais, nos quais um conjunto de filtros com propriedades variáveis é usado para computar os atributos da textura.

Os algoritmos de segmentação de texturas, usualmente, consistem em dividir a imagem em regiões, extrair atributos das mesmas e utilizar alguma técnica para agrupá-las [15]. Máscaras de Laws, matrizes de co-ocorrência e filtros de Gabor são exemplos de métodos de extração de atributos utilizados na literatura [16]. Para agrupar as regiões, normalmente, utilizam-se técnicas como *split-and-merge*, *pyramid node link* ou algum algoritmo de agrupamento (*clustering*) [17][18].

Segmentar uma imagem é atribuir a cada *pixel* da mesma um rótulo, formando assim regiões disjuntas e heterogêneas entre si. Quando existe um conjunto de padrões pré-

classificados, a segmentação é denominada supervisionada, caso contrário, trata-se de uma segmentação não supervisionada.

Para avaliar o resultado da segmentação, é necessário que um especialista defina manualmente a verdade terrestre (fronteiras das regiões), a qual pode ser diferente da determinada por outro especialista na mesma área. Mesmo quando há uma verdade terrestre previamente definida, a validade da mesma pode ser questionada. Por exemplo, a textura da pele de uma zebra pode ser vista como a união de duas texturas distintas, uma preta e uma branca, ou como uma única textura listrada. Portanto, na maior parte dos casos, não existe segmentação correta ou errada - a avaliação do resultado tem um caráter subjetivo [15].

Técnicas de compressão podem ser utilizadas para problemas de classificação em geral e vice-versa [19]. Dawy *et al.* [20] propõem duas medidas de distância baseadas em compressão e analisam o comportamento dessas medidas quando aplicadas ao campo da genética. Diversos algoritmos foram investigados: *Lempel-Ziv*, *Context Tree Weighting*, *Burrows Wheeler Transform*, *Prediction by Partial Matching* (PPM) e *DNACompress*.

Dawy *et al.* [20] mostraram que classificação baseada em compressão obtém bons resultados quando aplicada à filogenia. Um modelo evolucionário tem como hipótese que todos os seres herdam de um único ancestral. Uma nova espécie se forma quando um grupo de seres pára de trocar informações genéticas com os demais. Com o passar do tempo, esse grupo se divide novamente e, com isso, as espécies resultantes passam a compartilhar cada vez menos genes.

Ao classificar o DNA dos seres das espécies existentes, é possível agrupá-los sucessivamente, reconstruindo, desse modo, a filogenia. O agrupamento resultante da classificação de algumas amostras do banco NCBI [21], quando se utilizam algoritmos de compressão, está de acordo com a ordem atualmente aceita. O PPM e o *DNACompress* são os que proporcionam, de forma geral, os melhores resultados quando aplicados a seqüências genéticas [20].

Nobre Neto [22] utilizou o PPM para identificar os autores de textos da literatura brasileira. Por meio de validação cruzada, foram analisadas 112 obras no total, sendo quatro obras para cada uma das 28 classes de autores, resultando em uma taxa de acerto de 92.8% no melhor caso. Ao realizar os mesmos testes utilizando *Support Vector Machine* no lugar do PPM, a taxa de acerto caiu para 88%.

Barufaldi *et al.* [23] utilizaram o mesmo método com um propósito diferente: identificar o estilo literário de textos da literatura brasileira. Foram analisadas 37 obras, as quais pertenciam a um dos seguintes estilos: barroco, arcadismo, romantismo ou realismo. Apenas 3 obras foram classificadas incorretamente (taxa de acerto de 92%).

Bobicev [24] propôs um método para classificar textos de acordo com o conteúdo dos mesmos. A inovação, neste, caso foi utilizar para a classificação um PPM baseado em palavras em vez de caracteres. Os textos analisados foram os artigos da revista eletrônica *Evenimentul zilei*, os quais se enquadram em uma das seguintes categorias: editorial, negócios, política, investigação, cotidiano, mundo e esportes. A taxa de acerto foi de 97%.

Na área de segmentação de imagens, Batista e Meira [9] obtiveram 100% de acerto ao utilizar o algoritmo de compressão *Lempel-Ziv-Welch* (LZW) para classificar texturas de Brodatz [25]. Honório *et al.* [26] teve o mesmo êxito ao usar o PPM no lugar do LZW para classificar as texturas do mesmo banco. Uma das principais diferenças entre os métodos propostos no artigo de Honório e o apresentado no presente trabalho é que os primeiros requerem treinamento para classificar as texturas enquanto o segundo não.

O objetivo desse trabalho é propor um novo método de segmentação não supervisionada de texturas utilizando o modelo estatístico produzido pelo algoritmo de compressão de dados *Prediction by Partial Matching*.

1.1. Objetivos

O objetivo geral deste trabalho é propor um novo método de segmentação não supervisionada de imagens utilizando como atributo modelos PPM [27].

O primeiro objetivo específico é implementar o algoritmo PPM, pois, dessa forma, será mais fácil fazer as adaptações necessárias. A primeira delas é prepará-lo para a leitura dos *pixels* da imagem, que ocupam duas dimensões, enquanto que o PPM original foi projetado para trabalhar com apenas uma dimensão. Também é necessário definir como os modelos PPM serão utilizados para medir a similaridade entre duas texturas. Para medir a semelhança entre as regiões, é utilizada a razão de compressão obtida pela compressão com modelo PPM.

O segundo objetivo específico é escolher ou criar uma forma de dividir e de agrupar as diversas regiões da imagem. É necessário um método para decidir quais regiões devem ser analisadas e como estas devem ser agrupadas ou divididas de forma a encontrar as regiões finais, as quais, idealmente, correspondem às classes de textura presentes na imagem.

O último objetivo específico é a avaliação empírica dos resultados utilizando texturas de vários bancos da literatura, como o de Brodatz, o de Trygve Randen e o de Timo Ojala. Mesmo nos casos em que são conhecidas as texturas correspondentes a cada *pixel* da imagem a ser analisada, a taxa de acerto não é suficiente para garantir a qualidade dos resultados, é necessária também a avaliação por inspeção visual devido ao caráter subjetivo do tema.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, será apresentada a fundamentação teórica deste trabalho. A primeira seção (2.1) trata sobre agrupamento: definições, atributos, distância e exemplos de algoritmos utilizados na literatura. A segunda seção (2.2) apresenta alguns conceitos básicos da teoria da informação, o paradigma modelagem + codificação e, por último, o algoritmo PPM, que corresponde à base da medida de similaridade entre imagens utilizada neste trabalho.

2.1. Agrupamento

Na segmentação supervisionada, existe um conjunto de padrões pré-classificados, utilizados para extração de atributos, em uma etapa denominada *treinamento*, e o objetivo é atribuir um novo padrão, cuja classe é desconhecida, a uma das classes de treinamento.

Na segmentação não supervisionada, também chamada de agrupamento (*clustering*), o objetivo é agrupar um conjunto de padrões em classes (*clusters*) que compartilham alguma semelhança. Nesse caso, cada padrão também é atribuído a uma classe, mas estas são extraídas automaticamente dos padrões.

Seja $D \neq \varnothing$ um conjunto de padrões, $D_t \subset D$ o conjunto dos padrões utilizados no treinamento, $D_n = D - D_t$ o conjunto não vazio dos padrões de teste (a serem classificados), $D_p = 2^D$ o conjunto das partes de *D*, e $P : D_p \to 2^{D_p}$ uma função que, para todo subconjunto não vazio de *D*, gera todas as partições válidas desse subconjunto, ou seja:

$$P(D') = \left\{ P' \subset 2^{D'} : (\varnothing \notin P') \wedge \left(\bigcup_{C \in P'} C = D' \right) \wedge \left(C_1, C_2 \in P' \Rightarrow C_1 \cap C_2 = \varnothing \right) \right\}$$

Um segmentador supervisionado pode ser expresso como uma função $S_s : D_n \times P(D_t) \rightarrow 2^{D_t}$ tal que $S_s(d, P') \in P'$ para todo $d \in D_n$ e $P' \in P(D_t)$, ao passo que o não supervisionado pode ser expresso por $S_n : D_n \rightarrow P(D_n)$, quando o número de classes é inferido automaticamente, ou por $S_n : D_n \times N_C(D_n) \rightarrow P(D_n)$, quando o número de classes é um parâmetro do método, onde $|S_n(d, n)| = n$ para todo $d \in D_n$ e $n \in N_C(D_n)$, e $N_C(D_n) = \{m \in \aleph : 0 < m \leq |D_n|\}$.

2.1.1. Atributos

Ao se trabalhar com agrupamento, o primeiro problema é definir a representação dos padrões, que normalmente se dá por meio de uma função $E : D \rightarrow \Re^n$ que converte o conjunto de padrões *D* em um conjunto (denominado espaço de atributos) de vetores multidimensionais (denominados vetores de atributos) cujas coordenadas são chamadas atributos [28]. Não há um guia teórico que indique os atributos a serem utilizados [18]. Denomina-se extração de atributos o uso de transformações aplicadas aos padrões para produzir novos atributos [18].

Um extrator de atributos eficiente pode tornar trivial o trabalho do classificador, da mesma forma que um classificador eficiente pode gerar bons resultados até mesmo com atributos pouco representativos. Para diferenciar a responsabilidade de cada um, o extrator de atributos é dependente do domínio do problema, requer conhecimento específico, enquanto que o classificador processa os atributos de forma genérica, podendo, inclusive, ser utilizado em problemas de naturezas completamente distintas [29].

Em alguns casos, o número de dimensões do espaço de atributos é excessivamente elevado, tornando a execução do classificador inviável, dadas as limitações dos recursos computacionais disponíveis. Em outros, o número de dimensões é reduzido, mas a extração de um determinado atributo requer um custo muito alto. Mesmo em situações onde se utiliza um número pequeno de atributos de fácil aquisição, a presença de atributos redundantes ou inadequados pode até piorar o resultado do classificador [29]. Por esse motivo, é melhor isolar os atributos mais descritivos e discriminatórios [18], processo denominado seleção de atributos.

Os atributos podem ser classificados como quantitativos ou qualitativos [30]. Os quantitativos podem possuir valores contínuos (e.g. altura), discretos (e.g. número de patas) ou intervalares (e.g. duração de um evento), enquanto que os qualitativos podem ser nominais (e.g. cor) ou ordinais (e.g. níveis de temperatura: quente, médio, frio). Há atributos que não se enquadram nessa classificação, a exemplo dos citados nos trabalhos de Michalski, Stepp e Diday [31] e de Diday [32].

2.1.2. Distância

Para segmentar um conjunto de padrões *C*, é necessário definir uma função, que normalmente é aplicada sobre os vetores de atributos, que indique a distância entre seus elementos.

Sejam $x, y, z \in C$. Uma função distância $D: C \times C \to \Re$ precisa satisfazer três requisitos para ser chamada de métrica:

I. $D(x, y) = 0 \Leftrightarrow x = y$ (Identidade)

II. $D(x, y) = D(y, x)$ (Simetria)

III. $D(x, z) \leq D(x, y) + D(y, z)$ (Desigualdade triangular)

Uma métrica bastante utilizada é a distância Minkowski [33]. Dados dois pontos em $\Re^n$, $X = (x_1, x_2, ..., x_n)$ e $Y = (y_1, y_2, ..., y_n)$, a distância Minkowski de ordem *p* é definida como:

$$D_M(X, Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

Essa métrica pode ser vista como uma generalização de várias outras utilizadas na literatura. Dentre elas, podemos citar as seguintes:

Tabela 1

Casos particulares da distância de Minkowski

Distância	Equação	Ordem (p)
Manhattan	$D_{MM}(X,Y) = \sum_{i=1}^{n} \lvert x_i - y_i \rvert$	$p = 1$
Euclidiana	$D_{ME}(X,Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$	$p = 2$
Chebyshev	$D_{MC}(X,Y) = \max_{i=1}^{n} \lvert x_i - y_i \rvert$	$p \to \infty$

Fonte: Mohror e Karavanic [34]

A distância Manhattan tende a formar agrupamentos hiper-retangulares, enquanto que a Euclidiana, métrica mais utilizada, tende a formar classes hiperesféricas [33]. As distâncias Minkowski são ideais para padrões que possuem classes compactas ou isoladas [35], porém apresentam duas desvantagens: não funcionam bem com atributos que apresentam correlação linear (uma opção neste caso é utilizar a distância Mahalanobis) e tendem a favorecer atributos que apresentam elevada amplitude (uma solução, neste caso, é a normalização dos mesmos) [18].

As distâncias citadas até aqui são adequadas para atributos contínuos. O cálculo da métrica se torna difícil quando alguns ou todos os atributos não são contínuos. Atributos ordinais podem ser indexados e, dessa forma, tratados como discretos (com os quais é mais fácil lidar). Mas como definir a distância entre atributos intervalares e nominais? No artigo de Jain, Murty e Flynn [18], é possível encontrar diversas sugestões para lidar com esse tipo de problema.

Vale ressaltar que, em alguns casos, para medir a semelhança entre os atributos de um agrupamento, pode ser conveniente utilizar uma medida específica de similaridade que não satisfaça todos os requisitos de uma métrica.

2.1.3. Algoritmos de agrupamento

Escolhida a medida de similaridade, é necessário definir o algoritmo de agrupamento, o qual encontrará as classes presentes no conjunto a partir das distâncias entre seus elementos. Não há algoritmo de agrupamento universalmente satisfatório [33] – cada problema tem características que devem ser exploradas para que se obtenha um bom resultado. Também não há um consenso no julgamento dos resultados, mas muitos autores consideram mais satisfatória a divisão que simultaneamente maximiza a homogeneidade dentro de cada classe e a heterogeneidade entre classes distintas [33].

Nos trabalhos de Xu e Wunsch [33] e de Jain, Murty e Flynn [18], é possível encontrar uma taxonomia para os algoritmos de agrupamento. A primeira classificação, que diz respeito ao modo como os atributos são utilizados no processo, divide os algoritmos em monotéticos, os quais utilizam todos os atributos ao mesmo tempo no cálculo das distâncias, e politéticos, que utilizam os atributos de forma seqüencial. Anderberg [36] apresentou um algoritmo monotético que é inadequado para padrões com alta dimensionalidade porque gera 2^d classes, onde d é o número de dimensões.

Também é possível classificar os algoritmos de agrupamento como determinísticos ou estocásticos. Algoritmos determinísticos sempre produzem o mesmo resultado, não importa quantas vezes ele seja executado. Algoritmos estocásticos baseiam-se em variáveis que assumem valores aleatórios, portanto, normalmente produzem resultados diferentes cada vez que são executados. Estes últimos são muito utilizados em algoritmos particionais (que serão descritos a seguir), pois possibilitam uma busca otimizada num espaço de soluções muito grande, onde uma busca exaustiva seria intratável.

Algoritmos de agrupamento também podem ser classificados como particionais, hierárquicos e nebuloso (*fuzzy*), sendo que estes dois últimos não se enquadram na definição apresentada na seção 2.1. Os algoritmos hierárquicos têm como resultado uma árvore binária ou um dendrograma representando os agrupamentos e o nível de similaridade em que cada par de grupos se une. Esses algoritmos são mais versáteis e, frequentemente, computacionalmente mais complexos que os particionais [18].

Os algoritmos hierárquicos podem ser aglomerativos ou divisivos. Um algoritmo divisivo começa com um único grupo, o qual contém todos os padrões do conjunto, e então divide os grupos presentes na partição até que um critério de parada seja satisfeito. Um algoritmo hierárquico aglomerativo começa com vários grupos, um para cada padrão do conjunto, e sucessivamente agrupa os grupos até que um determinado critério de parada seja satisfeito.

Como exemplos de algoritmos hierárquicos, podemos citar o BIRCH [37], que é robusto a exceções e pode ser aplicado a padrões volumosos; o CURE [38], que evita o efeito corrente e a tendência de formar grupos hiperesféricos; e o ROCK [39], cujo objetivo é lidar com atributos qualitativos. Outros exemplos podem ser encontrados nas referências [33] e [18].

Os algoritmos particionais têm como resultado uma única partição. São úteis para grandes conjuntos de padrões, onde a construção de um dendrograma é inviável e a busca exaustiva é proibitiva. Normalmente esses algoritmos são executados várias vezes com diferentes estados iniciais e o melhor resultado obtido é escolhido [18]. Um problema comum aos algoritmos particionais é a escolha do número de grupos, tema abordado nas referências [40] e [33].

K-médias é um algoritmo particional bastante conhecido. Ele é simples e seu tempo de execução é linear em função do tamanho do conjunto de padrões. É ideal para agrupamentos compactos e hiperesféricos. Tem como desvantagens a não garantia de atingir o ótimo global e sua sensibilidade à partição inicial, a ruídos e às exceções [33] [18].

O ISODATA [41] é uma variação do K-médias que agrupa ou divide grupos de acordo com um limiar, dispensando, com isso, a necessidade de fornecer o número de grupos. O CLICK [42] utiliza teoria dos grafos para encontrar a partição. O ELBG [43] utiliza um mecanismo que trata o problema da má inicialização (partição inicial ruim). Krishna e Murty [44] propuseram uma mistura de algoritmos genéticos e K-médias que converge para o ótimo global. Outros exemplos de algoritmos particionais podem ser encontrados nas referências [33] e [18].

Os algoritmos expostos até o momento atribuem cada padrão a um único grupo. No caso da aglomeração *fuzzy*, os padrões fazem parte de todos os grupos com um determinado grau de pertinência. Esse tipo de algoritmo é útil quando os limites entre os padrões são ambíguos ou mal definidos [33]. Um dos algoritmos de agrupamento *fuzzy* mais populares é o *Fuzzy C-Means*, que possui os mesmos problemas fundamentais do K-médias [33], embora geralmente seja melhor que este último para evitar máximos locais [18].

2.2. Teoria da Informação

2.2.1. Conceitos básicos

O objetivo de um compressor é reduzir a representação da informação [45]. Na compressão sem perdas, nenhuma informação é perdida, enquanto que na compressão com perdas alguma informação é perdida para que seja possível uma maior redução na representação da mesma. Para atingir esse objetivo, técnicas de compressão exploram as características de fontes de informação específicas.

Um esquema de um compressor/descompressor genérico é apresentado na Figura 1. O compressor, ao receber uma mensagem M, produz uma mensagem codificada M_c cujo tamanho, idealmente, é menor que o de M. A mensagem codificada é transmitida através do canal, que pode ser, por exemplo, um disco rígido ou uma linha telefônica. O descompressor, ao decodificar M_c, gera uma nova mensagem M_d que, na compressão sem perdas, é idêntica à original, enquanto que, na compressão com perdas, é apenas uma aproximação de M.

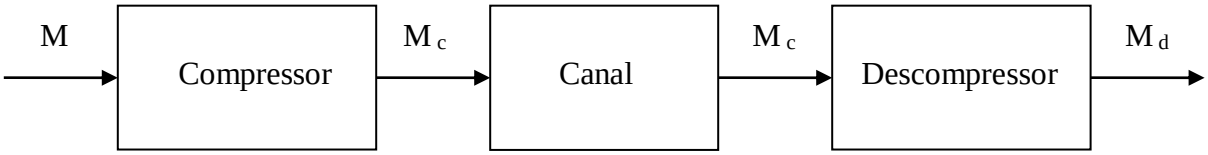

Figura 1: Compressor/Descompressor genérico

Uma medida muito utilizada para avaliar o grau de compressão obtido é a razão de compressão (RC), que corresponde à razão entre o tamanho de M e o tamanho de M_c. Na compressão com perdas, normalmente, utiliza-se também uma métrica de distorção para identificar a qualidade, que depende da importância da informação descartada da mensagem original. Em geral, maiores distorções possibilitam maiores razões de compressão [45].

2.2.2. Informação e entropia

Shannon se preocupou em descobrir quanta informação é produzida por uma dada fonte e concluiu que, ao utilizar o conhecimento estatístico sobre a mesma, é possível reduzir a capacidade necessária do canal [46].

Seja *S* uma fonte de informação que produz elementos selecionados de um conjunto $A = \{a_1, a_2, ..., a_M\}$, denominado *alfabeto* de *S*, cujos elementos são conhecidos como *símbolos*. A *auto-informação* associada à ocorrência do símbolo a_i é definida como:

$$I(a_i) = -\log_2 P(a_i) \text{ bits}$$

onde $P(a_i)$ é a probabilidade do símbolo a_i. Se os sucessivos símbolos produzidos pela fonte são independentes, então a entropia *H* é definida como:

$$H = -\sum_{i=1}^{M} P(a_i) \log_2 P(a_i) \text{ bits/símbolo}$$

A entropia é a menor quantidade (média) de bits necessária para codificar uma fonte de saída sem perda de informação [46]. Portanto, para avaliar a eficiência de um código, basta comparar seu comprimento médio com a entropia, sendo melhor aquele que apresentar a menor diferença entre essas grandezas.

2.2.3. Paradigma modelagem + codificação

Uma das abordagens utilizadas para a compressão de dados consiste em dividir a solução para o problema em duas partes [27]. A primeira delas é responsável pela extração do modelo, atribuindo a cada símbolo uma probabilidade de ocorrência, a qual pode variar no decorrer do processo. A segunda parte, denominada codificação, tem a função de transformar em bits os símbolos, codificando-os de acordo com as probabilidades atribuídas aos mesmos.

Quando se utiliza o mesmo modelo para qualquer mensagem, diz-se que ele é estático. Nesse caso, o mesmo deve ser incluído na mensagem codificada ou compartilhado entre o compressor e o descompressor. Uma desvantagem dessa abordagem é que, como o modelo nunca se modifica, apenas mensagens que se ajustam ao modelo são bem comprimidas.

Por outro lado, modelos estáticos podem ser úteis na classificação de padrões. Para decidir a que classe um padrão pertence, utilizam-se vários modelos para comprimi-lo, onde cada modelo contém as principais características de cada classe. O padrão é então comprimido por cada um dos modelos e atribuído à classe correspondente ao modelo que resultar numa maior razão de compressão. Nesse caso, o fato de o modelo não se adaptar à mensagem, já que é estático, ajuda no processo de classificação.

Existem também modelos de compressão classificados como semi-adaptativos, casos em que é necessário percorrer a mensagem inteira duas vezes, uma para extrair o modelo e outra para codificá-la. Para que o descompressor seja capaz de descobrir o modelo empregado, o mesmo deve ser incluso na mensagem codificada.

A primeira desvantagem dessa abordagem é a necessidade de percorrer a mensagem duas vezes; a segunda é o fato de ser desejado, em alguns casos, começar o envio da mensagem codificada antes de alcançar o final da mesma; e a terceira é a necessidade de incluir o cabeçalho com o modelo empregado. A vantagem de se usar modelo semi-adaptativo em vez de estático é que o primeiro pode ser utilizado numa aplicação de propósito mais geral, pois o mesmo se adapta à mensagem a ser codificada.

Por último, temos os modelos adaptativos, os quais são reconstruídos à medida que cada símbolo da mensagem é codificado. Tal abordagem pode dispensar a necessidade de cabeçalho, permite a transmissão da mensagem antes que o término da mesma tenha sido alcançado, não

exige mais de uma leitura da mensagem, pode ser utilizado numa variedade maior de aplicações, e, em geral, não apresenta resultados significativamente piores que o semi-adaptativo, podendo, inclusive, ser melhores em alguns casos [27]. Mas a necessidade de reconstruir o modelo inteiro a cada símbolo codificado pode exigir um tempo de processamento elevado ou, em alguns casos, proibitivo.

Quanto maior a probabilidade atribuída ao símbolo, menor deve ser a quantidade de bits utilizada para codificá-lo. Diz-se que um codificador é ótimo quando o mesmo atinge a entropia ou, em outras palavras, quando utiliza $-\log_2 p$ bits para cada símbolo codificado, onde p é a probabilidade atribuída a ele pelo modelo. O codificador proposto por Huffman [47] é ótimo quando as probabilidades são potências de ½, enquanto que o codificador aritmético [48] atinge desempenho ótimo em qualquer caso [49].

2.2.4. PPM

É possível reduzir o número de bits necessários para codificar uma mensagem quando a probabilidade atribuída a cada símbolo é calculada baseando-se nos símbolos que o precedem (contexto) [50]. Incluir essa informação num cabeçalho da mensagem nem sempre é viável pois o tamanho dele cresce exponencialmente com o tamanho do contexto.

Para explorar as vantagens do modelo contextual de uma forma que não exigisse a presença de um cabeçalho, Cleary e Witten [27] propuseram um método, denominado *Prediction by Partial Matching* (PPM), que é considerado, juntamente com suas variações, um dos melhores algoritmos de compressão sem perdas sob o ponto de vista da razão de compressão [51].

Ao codificar um símbolo, denotado por *s*, o algoritmo PPM identifica a lista de *t* símbolos, denotada por C_t, que o precede. Se for a primeira vez que C_t ocorre na mensagem, analisa-se a lista de t-1 símbolos anteriores a *s*, denotada por C_{t-1} . Essa redução no tamanho da lista ocorre até o momento em que for encontrado um tamanho *k* tal que C_k tenha ocorrido pelo menos duas vezes na mensagem lida até o momento.

Em seguida, o PPM verifica quantas vezes, na parte da mensagem que precede *s*, cada símbolo do alfabeto ocorreu após C_k. O PPM utiliza um símbolo especial, denominado escape, para representar aqueles que ainda não ocorreram após um dado contexto. Uma das variantes do PPM, o PPM-C, considera que o número de ocorrências do escape de cada contexto é igual ao número de símbolos distintos que ocorreram após o contexto em questão.

Se *s* ocorreu pelo menos uma vez após C_k, o PPM codifica s com probabilidade $P(s|C_k)$ e parte para a codificação do próximo símbolo. O valor de $P(s|C_k)$ é igual ao número de ocorrências de *s* após C_k dividido pelo número de ocorrências de todos os símbolos (incluindo o escape) após o mesmo contexto. Se *s* nunca ocorreu após C_k , o PPM codifica o escape, cuja probabilidade é calculada da mesma forma, e tenta codificar *s* utilizando um contexto menor, C_{k-1}, e então todo o processo se repete.

Se o tamanho do contexto atingir zero, são utilizadas as freqüências de ocorrência de cada símbolo isolado, independentemente dos contextos que os precedem. Se for a primeira vez que *s* aparece na mensagem, escape será codificado no contexto de tamanho zero, e o PPM codifica *s* com probabilidade $1/|A'|$, onde A' é o conjunto dos símbolos que ainda não ocorreram, e parte para a codificação do próximo símbolo da mensagem.

3. MATERIAIS E MÉTODOS

Neste capítulo, será descrito um método de segmentação não supervisionada de texturas que utiliza como medida de similaridade entre regiões as taxas de bits resultantes da compressão utilizando modelos, produzidos pelo algoritmo PPM, extraídos das mesmas.

3.1. Método Proposto

Sejam *C* um conjunto de níveis de cinza e $l, a \in IN$. Uma imagem pode ser definida como um conjunto $I = \{(x, y, c) \in \aleph \times \aleph \times C : 0 \le x < l \wedge 0 \le y < a\}$. Tomando como base as definições apresentadas na seção 2.1, o segmentador proposto S_p é uma função $S_p : I \times N_C(I) \to P(I)$ tal que $S_p(p,n) = S_n(p,n)$ para todo $p \in I$ e $n \in N_C(I)$.

Em resumo, o método proposto consiste em particionar a imagem inteira em regiões, atribuindo cada uma delas a uma classe diferente, as quais serão unidas por meio de uma estratégia gulosa até que o número de classes restantes seja igual ao número de texturas presentes na imagem, o qual é fornecido pelo usuário.

Em seguida, a imagem é sucessivamente dividida em regiões muito menores e cada uma delas é atribuída à classe mais próxima. A medida de similaridade entre regiões utilizada, que não é uma métrica, é uma nova função de distância baseada na taxa de bits resultante da compressão pelos modelos PPM extraídos das mesmas.

Portanto, trata-se de um método particional e determinístico, projetado para lidar com atributos discretos (*pixels* de uma imagem), e que pode ser dividido em duas etapas: uma segmentação não supervisionada seguida de uma segmentação supervisionada que utiliza como padrões pré-classificados as sementes resultantes da etapa anterior.

3.1.1. Cálculo das distâncias

Inicialmente, a imagem é dividida em regiões de tamanho *N* x *N*, onde *N* é um parâmetro. O ideal seria que uma janela *N* x *N* deslizante percorresse toda a imagem e que, para cada posição da janela, fosse extraído um modelo. Essa abordagem, porém, foi descartada por causa do tempo de processamento necessário para executá-la. Portanto, em vez de se utilizar uma janela deslizante, a imagem é dividida em regiões adjacentes de tamanho *N* x *N*. O próximo passo consiste em calcular as distâncias entre todas as regiões resultantes da divisão anterior.

O algoritmo PPM foi projetado para processar fluxos de entrada de uma dimensão, como textos por exemplo. Portanto, foi necessário adaptar o PPM à natureza 2D do problema. A solução encontrada foi ler os *pixels* da região de cima para baixo e da esquerda para a direita e, a partir dos níveis de cinza encontrados, extrair modelos PPM. Durante esse processo, o contexto é reiniciado (tamanho do contexto = 0) cada vez que ocorre uma mudança de linha ou quando dois *pixels* não adjacentes são lidos. Este procedimento é válido inclusive para regiões não retangulares, as quais provavelmente ocorrerão nas etapas seguintes do algoritmo, que serão descritas nas próximas seções.

200	41	28	113	104	9	6
117	118	98	10	11	162	13
14	15	16	17	18	19	220
211	22	23	24	25	26	127
8	29	30	31	32	188	134
35	36	37	8	39	74	41
44	20	111	67	46	47	48

(a)

15 16 17 **R** 23 24 25 **R** 30 **R** 32 **R** 36 37 **R**

(b)

Figura 2: (a) níveis de cinza de uma imagem 7 x 7 e (b) entrada do PPM ao ler a região destacada. As letras R indicam onde os contextos são reiniciados

Conforme descrito na seção 2.2.4, dado um alfabeto A, um modelo PPM é uma função $M : A \times A^* \rightarrow [0,1]$ tal que $M(s,C) = P(s \mid C)$ onde $s \in A$, $C \in A^*$ e $P(s \mid C)$ é a probabilidade de *s* ocorrer dado o contexto *C*. No caso das regiões, há ainda um símbolo $R \in A$, que não é codificado, para indicar que o contexto será reiniciado, ou seja, $M_r(R,C) = 1$ e $\sum_{s \in A - \{R\}} M_r(s,C) \leq 1$ para todo $C \in A^*$, e ainda $M_r(s, C_1 R C_2) = M_r(s, C_2)$ para todo $C_1, C_2 \in A^*$ e $s \in A$, onde $C_1 R C_2$ é a concatenação de C_1, R e C_2.

Sejam $C_s : A \rightarrow \{0,1\}$ uma função que retorna 0 quando $s = R$ e 1 em caso contrário, r_1 e r_2 duas regiões da imagem *I*, M_1 o modelo extraído de r_1, e $s_1 ... s_t$ a seqüência de símbolos lidos de r_2, conforme descrito no início desta seção. A distância entre r_1 e r_2 é definida neste trabalho como:

$$D_R(r_1, r_2) = \frac{-\sum_{i=1}^{t} \log M_1(s_i, s_1 ... s_{i-1})}{\sum_{i=1}^{t} C_s(s_i)}$$

O denominador da equação anterior tem como propósito ignorar o efeito da codificação das mudanças de linhas (representadas pelo símbolo R) no cálculo da distância. Portanto, D_R pode ser vista a quantidade média de informação dos *pixels* da região. Desse modo, as mudanças de linha interferem de forma indireta na codificação, dado que o contexto é reinicializado toda vez que elas ocorrem.

A equação anterior é aplicável a qualquer região não vazia, mas, durante a escolha das sementes, conforme detalhado na próxima seção, novas regiões, resultantes do agrupamento das anteriores, formam-se a cada iteração. Para calcular o modelo dessas novas regiões, seria necessário percorrer novamente todos os *pixels* das regiões que se agruparam e formar a árvore de contextos do PPM novamente.

O modelo resultante, contudo, seria muito semelhante aos das regiões antigas, pois apenas os *pixels* que deixaram de ser fronteira e passaram a ser internos acarretariam mudanças no modelo final. Como a quantidade de *pixels* desse tipo é desprezível quando comparada à quantidade de *pixels* da região inteira, as mudanças no modelo seriam mínimas.

Portanto, aproveitando esta característica para reduzir o tempo de processamento, a distância entre dois grupos de regiões $G_1 = \{r_{11}, r_{12}, ..., r_{1n}\}$ e $G_2 = \{r_{21}, r_{22}, ..., r_{2m}\}$ é calculada utilizando-se uma média das distâncias entre as regiões de cada grupo, sem a necessidade de recalcular o modelo, por intermédio da seguinte equação:

$$D_G(G_1, G_2) = \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} D_R(r_{1i}, r_{2j})$$

Esta medida não pode ser chamada de métrica. Ao codificar, por exemplo, uma região com o próprio modelo, será utilizada uma quantidade positiva de bits para codificar o primeiro símbolo (nível de cinza do *pixel* no canto superior esquerdo), violando dessa forma o requisito I da seção 2.1.2, o qual determina que $D(x, x) = 0$.

3.1.2. Escolha das sementes

Considera-se que esta etapa é bem sucedida quando são encontradas regiões, doravante chamadas sementes, que possuem características (nesse caso, modelo PPM) muito semelhantes às das classes verdadeiras presentes na imagem. O número de classes é um parâmetro do método.

Inicialmente, a imagem é dividida em regiões adjacentes *N* x *N* e cada uma delas é atribuída a um grupo distinto, formando assim uma partição inicial P_0. O tamanho *N* deve ser grande o suficiente para capturar as características da textura de cada região, entretanto deve respeitar um determinado limite superior, caso contrário haverá muitas regiões contendo mais de uma textura, o que pode prejudicar a escolha das sementes.

Seja C_q um conjunto qualquer, $U:\{C_q\}\rightarrow P(C_q)$ uma função que retorna uma partição onde cada grupo contém um elemento de C_q, ou seja, $|U(C_q)|=|C_q|$, e *Q* uma função que recebe uma imagem I e um $n\in IN$ e retorna um conjunto de regiões $R_{ij}=\{(x,y,c)\in I:\lfloor x/n\rfloor=i\wedge\lfloor y/n\rfloor=j\}$. Então $P_0=U(Q(I,n))$, onde:

$$Q(I,n)=\bigcup_{i=0}^{(l-1)/n}\left(\bigcup_{j=0}^{(a-1)/n}\{R_{ij}\}\right)$$

Através de uma estratégia gulosa, o par de grupos de regiões mais próximo, de acordo com a equação de distância definida na seção anterior, será fundido, formando uma nova partição:

$$P_{i+1}=\{((P_i-G_1)-G_2)\cup\{G_1\cup G_2\}\}$$

onde $G_1,G_2\in P_i$ e $D_G(G_1,G_2)\le D_G(G_3,G_4)$ para todo $G_3,G_4\in P_i$. Seja $R_{64}=Q(I,64)$ o conjunto das regiões (64 x 64) adjacentes de uma imagem *I*, e *J* uma função que recebe como parâmetros uma partição e o número de classes tal que

$$J(P_i, n) = \begin{cases} P_i & se \quad |P_i| \leq n \\ P_{i+1} & se \quad |P_i| > n \end{cases}$$

Então o algoritmo guloso de agrupamento utilizado neste trabalho pode ser visto como um segmentador não supervisionado $S'_n : R_{64} \times N_C(R_{64}) \rightarrow P(R_{64})$, conforme descrito em 2.1 (nesse caso $D_n = R_{64}$), tal que $S'_n(R_{64}, I) = J(U(R_{64}), n)$. Fazendo $P_f = S'_n(R_{64}, I)$, o conjunto de sementes é então definido como:

$$S = \bigcup_{G \in P_f} \left\{ \bigcup_{R \in G} R \right\}$$

Diferentemente da maioria dos algoritmos de agrupamento da literatura, que operam sobre um espaço de atributos em IR^n, o algoritmo guloso utilizado neste trabalho opera sobre um espaço em 2^I, diretamente extraído dos padrões (*pixels*). Portanto, como não se utilizam atributos, não faz sentido classificar o método como monotético nem politético.

Nos testes realizados, notou-se que regiões de tamanho menor que oito geram bastante ruído (considerando como ruído a união das regiões incorretamente classificadas) na imagem resultante, o que se deve ao fato de que um tamanho tão reduzido é incapaz de extrair de forma satisfatória as características de uma textura. Por outro lado, regiões de tamanho 8 x 8, 16 x 16 ou 32 x 32 requerem um elevado tempo de execução (a ordem de complexidade é discutida na seção 3.1.4).

Empiricamente, observa-se que regiões iniciais de tamanho 64 x 64 são ideais para as imagens analisadas neste trabalho. Tamanhos maiores (e.g. 128 x 128) favorecem a ocorrência de regiões contendo mais de uma textura, o que não é desejável, como foi comentado no início desta seção.

3.1.3. Refinamento das fronteiras

Após escolher o conjunto de sementes, a imagem inteira é dividida em regiões de tamanho 8 x 8 e cada uma delas é atribuída à classe correspondente à semente mais próxima. As regiões nesta etapa são menores porque o objetivo é identificar com precisão as fronteiras das texturas, enquanto que, na etapa anterior, o objetivo é identificar as características (modelos) das mesmas.

Regiões de tamanho 8 x 8 foram utilizadas porque foi verificado empiricamente que, nas imagens testadas neste trabalho, regiões menores geram muito ruído na imagem, e as delimitações das fronteiras entre as texturas não melhoram significativamente.

Fazendo $R_8 = Q(I,8)$, esta etapa do método pode ser expressa por uma função $H : R_8 \to S$, onde $D_R(R', H(R')) \leq D_R(R', S')$ para todo $R' \in Q$ e $S' \in S$. Se definirmos uma função $B : I \to R_8$, onde $p \in B(p)$ para todo $p \in I$, ou seja, B retorna a região de R_8 que contém o *pixel* p, então esta etapa do método pode ser vista como um segmentador supervisionado $S_s' : I \times P(I) \to S$ tal que $S_s'(p, S) = H(B(p), S)$.

3.1.4. Complexidade

Os tempos de inserção e de busca em uma tabela *hash* que utiliza encadeamento para tratar colisão são, respectivamente, $O(1)$ e $O(n)$, onde *n* é o tamanho da entrada [49]. Da forma como foi implementado, o PPM contém uma tabela *hash* que mapeia cada contexto para a tabela de contadores correspondente. Essa tabela de contadores, que indica quantas vezes cada símbolo ocorreu após o dado contexto, também é uma tabela *hash* que mapeia cada símbolo para o respectivo contador. Ambas usam encadeamento.

Para codificar um símbolo *s*, o PPM executa, no máximo, $t_C + 1$ buscas na tabela de contextos até achar uma tabela de contadores que contenha *s*, onde t_C é o tamanho máximo do contexto. A rigor, a busca de uma tabela de contadores requer $O(\min(n_s, 2^{t_A}))$, onde t_A é o tamanho do alfabeto, n_s é o número de símbolos antes de *s*, e 2^{t_A} é o número de contextos distintos. Porém, no caso de imagens, o número de níveis de cinza ($= t_A$) geralmente é grande o suficiente para garantir que $n_s << 2^{t_A}$, o que nos permite afirmar que a busca de uma tabela de contadores é $O(n_s)$.

Cada vez que o contexto é encontrado, também é realizada uma busca na tabela de contadores, o que requer $O(t_A)$. Com isso, a busca do contador de um símbolo é $O(t_C(t_A + n_s))$. Como a operação de inserção é $O(1)$, é possível concluir, através de um raciocínio análogo, que a atualização do modelo também é $O(t_C(t_A + n_s))$.

No caso da distância entre regiões de uma imagem, não é necessário transformar o símbolo em uma seqüência de bits de acordo com a probabilidade a ele atribuída (o que seria a codificação de fato), basta calcular a *auto-informação* (seção 2.2.2) a ele associada, operação executada em $O(1)$.

Assim, o processo completo de codificação de um único símbolo é $O(t_C(t_A + n_s))$, enquanto que a codificação de uma entrada de tamanho *n* requer:

$$\sum_{n_s=0}^{n-1} O\left(t_C\left(t_A + n_s\right)\right) = O\left(t_A t_C n + n^2\right)$$

Sejam $R_1, R_2 \subset I$ duas regiões de uma imagem *I*, a distância $D_R\left(R_1, R_2\right)$, que é baseada no algoritmo PPM, requer $O\left(t_A t_C\left(\left|R_1\right| + \left|R_2\right|\right) + \left(\left|R_1\right| + \left|R_2\right|\right)^2\right)$ visto que, nesse caso, o tamanho da entrada é $n = \left|R_1\right| + \left|R_2\right|$.

Na primeira etapa do método (escolha das sementes), a imagem é dividida em $n_R = \left\lfloor l_I / \sqrt{t_R} \right\rfloor \left\lfloor a_I / \sqrt{t_R} \right\rfloor$ regiões de tamanho t_R, onde l_I e a_I, são, respectivamente, a largura e a altura da imagem *I*.

Da forma como foi implementado, o algoritmo armazena em uma matriz $n_R \times n_R$ as distâncias entre as regiões da imagem. Como todas as regiões possuem o mesmo número de *pixels* (t_R), a complexidade necessária para calcular a distância entre duas delas é dada por $O\left(t_A t_C\left(t_R + t_R\right) + \left(t_R + t_R\right)^2\right) = O\left(t_A t_C t_R + {t_R}^2\right)$.

Considerando o tamanho da matriz, é possível afirmar que sua construção tem uma complexidade da ordem de $O\left({n_R}^2\left(t_A t_C t_R + {t_R}^2\right)\right) = O\left(t_A t_C {t_I}^2 / t_R + {t_I}^2\right)$, sendo $t_I = l_I a_I$ o tamanho da imagem.

Como as distâncias D_R entre as regiões já foram previamente calculadas e armazenadas em uma matriz, a complexidade do cálculo de D_G, de acordo com a seção 3.1.1, é $O\left(\left|G_1\right|\left|G_2\right|\right)$, onde $G_1, G_2 \subset 2^I$ são dois grupos de regiões de uma imagem *I*.

As regiões são sucessivamente agrupadas duas a duas, o que requer, em cada agrupamento, uma busca entre os $n_G\left(n_G - 1\right)$ pares, onde n_G é o número de grupos restantes, para identificar qual deles possui grupos mais próximos.

Essa busca implicaria na execução de $n_G\left(n_G - 1\right)\left|G_i\right|\left|G_j\right|$ operações, mas, se uma matriz $n_R \times n_R$ também for utilizada para os grupos, o cálculo da distância entre dois grupos será imediato, fazendo com que o número de operações diminua para $n_G\left(n_G - 1\right)$.

Após cada busca, é necessário calcular a distância entre o novo grupo, resultante da união dos grupos do par encontrado, e todos os outros restantes. Essa operação, que seria $O(|G_1||G_2|)$, pode ser executada em $O(1)$ utilizando a seguinte equação:

$$D_G(G_1 \cup G_2, G_3) = \frac{|G_1||G_3| D_G(G_1, G_3) + |G_2||G_3| D_G(G_2, G_3)}{|G_1| + |G_2| + |G_3|}$$

onde $D_G(G_1, G_2) = D_R(R_1, R_2)$ quando $|G_1| = |G_2| = 1$, com $R_1 \in G_1$ e $R_2 \in G_2$.

Portanto, em cada iteração são executadas $n_G(n_G - 1) + (n_G - 1) = {n_G}^2 - 1$ operações, sendo $n_G(n_G - 1)$ para buscar o par e $(n_G - 1)$ para calcular a distância entre o novo grupo e os demais. Seja n_C o número de classes, este processo se repete até que $n_G = n_C$, o que permite afirmar que o número total de operações executadas nos agrupamentos é:

$$\sum_{n_G = n_R}^{n_C} \left({n_G}^2 - 1 \right) = O\left({n_R}^3 \right) = O\left({t_I}^3 / {t_R}^3 \right)$$

Consequentemente, a escolha das sementes é $O\left(t_A t_C {t_I}^2 / t_R + {t_I}^3 / {t_R}^3 + {t_I}^2 \right)$. Em seguida tem início a etapa de refinamento das fronteiras, cujo procedimento inicial é a extração dos modelos das n_C sementes de tamanho $t_{Si} = t_R t_{Gi}$, sendo t_{Gi} o tamanho do grupo que deu origem à i-ésima semente. A complexidade desse cálculo inicial é dada por:

$$\sum_{i=1}^{n_C} O\left(t_A t_C \left(t_R t_{Gi} \right) + \left(t_R t_{Gi} \right)^2 \right) = \sum_{i=1}^{n_C} O\left(t_A t_C t_R t_{Gi} \right) + \sum_{i=1}^{n_C} O\left({t_R}^2 {t_{Gi}}^2 \right)$$

onde

$$\sum_{i=1}^{n_C} O\left(t_A t_C t_R t_{Gi} \right) = O\left(t_A t_C t_R \sum_{i=1}^{n_C} t_{Gi} \right) = O\left(t_A t_C t_R n_R \right) = O\left(t_A t_C t_I \right)$$

Logo, para encontrar a complexidade do pior caso da extração de modelos é necessário resolver o seguinte problema de otimização:

Maximize:

$$\sum_{i=1}^{n_C} t_i^{\,2}$$

sujeito a:

$$\sum_{i=1}^{n_C} t_i = n_R$$

$$t_i \geq 1$$

Tem-se:

$$\left(\sum_{i=1}^{n_C} t_i \right)^2 = n_R^{\,2} \Rightarrow \sum_{i=1}^{n_C} t_i^{\,2} + \sum_{i=1}^{n_C-1} \sum_{j=i+1}^{n_C} t_i t_j = n_R^{\,2} \Rightarrow \sum_{i=1}^{n_C} t_i^{\,2} = n_R^{\,2} - \sum_{i=1}^{n_C-1} \sum_{j=i+1}^{n_C} t_i t_j$$

Como $t_i, t_j \geq 1$, tem-se:

$$\sum_{i=1}^{n_C} t_i^{\,2} = n_R^{\,2} - \sum_{i=1}^{n_C-1} \sum_{j=i+1}^{n_C} t_i t_j \leq n_R^{\,2} - \sum_{i=1}^{n_C-1} \sum_{j=i+1}^{n_C} 1 \leq n_R^{\,2}$$

Então:

$$\sum_{i=1}^{n_C} O\left(t_R^{\,2} t_{Gi}^{\,2}\right) = O\left(t_R^{\,2} \sum_{i=1}^{n_C} t_{Gi}^{\,2} \right) = O\left(t_R^{\,2} n_R^{\,2}\right) = O\left(t_I^{\,2}\right)$$

Assim a ordem de complexidade da extração de modelos é $O\left(t_A t_C t_I + t_I^{\,2}\right)$. Em seguida a imagem é dividida em $n'_R = \left\lfloor l_I / \sqrt{t'_R} \right\rfloor \left\lfloor a_I / \sqrt{t'_R} \right\rfloor$ regiões de tamanho t'_R cujos *pixels* serão atribuídos à classe correspondente ao modelo mais próximo. O número de operações envolvidas nesse processo final de classificação é de ordem:

$$\sum_{i=1}^{n_C} O\left(t_A t_C \left(t_{Si} + t'_R\right) + \left(t_{Si} + t'_R\right)^2\right) = \sum_{i=1}^{n_C} O\left(t_A t_C t_{Si} + t_{Si}^{\,2}\right) = O\left(t_A t_C \sum_{i=1}^{n_C} t_{Si} + \sum_{i=1}^{n_C} t_{Si}^{\,2} \right)$$

Na equação anterior, t'_R foi descartado porque $t'_R \leq t_{Si}$. Ainda temos que:

$$\sum_{i=1}^{n_C} t_{Si} = t_I$$

$$\sum_{i=1}^{n_C} {t_{Si}}^2 \leq \max(t_{Si}) \leq {t_I}^2$$

Portanto a ordem de complexidade do processo de classificação dessas regiões menores é $O\left(t_A t_C t_I + {t_I}^2\right)$. Sendo assim, pode-se afirmar que a ordem de complexidade total do algoritmo é igual a $O\left(t_A t_C {t_I}^2 / t_R + {t_I}^3 / {t_R}^3 + {t_I}^2\right)$, da extração das sementes, mais $O\left(t_A t_C t_I + {t_I}^2\right)$, da extração dos modelos, mais $O\left(t_A t_C t_I + {t_I}^2\right)$ da classificação final, resultando num total de $O\left(t_A t_C {t_I}^2 / t_R + {t_I}^3 / {t_R}^3 + {t_I}^2\right)$, ou seja, a extração das sementes é o que processo que no pior caso demanda o maior número de operações.

Resta analisar a quantidade de memória necessária. Para simplificar, a partir deste ponto do texto até o fim desta seção, os cálculos se referem à quantidade de memória alocada, e não ao número de operações executadas.

A tabela de contextos de um modelo PPM conteria, no máximo, 2^{t_A} entradas (encadeadas ou não), mas, na prática, apenas n entradas são utilizadas pois $n << 2^{t_A}$. Cada entrada contém uma tabela de contadores com, no máximo, t_A entradas (encadeadas ou não), uma para cada símbolo do alfabeto, portanto um modelo PPM consome $O(t_A n)$.

Durante a escolha das sementes, é criada uma matriz $n_R \times n_R = O\left({t_I}^2 / {t_R}^2\right)$. Para construir essa matriz, modelos PPM de regiões de tamanho t_R são criados, um de cada vez, e aplicados às outras regiões para calcular a taxa de bits. A criação de modelos é $O(t_A t_R)$, já que é um por vez, mas o cálculo da taxa é $O(1)$ pois os modelos permanecem estáticos depois de criados.

É necessário utilizar uma matriz $n_R \times n_R$ ao agrupar as regiões, mas o cálculo de seus elementos também é $O(1)$, o que nos possibilita afirmar que a escolha das sementes requer $O\left({t_I}^2 / {t_R}^2\right) + O(t_A t_R) + O(1) + O(1) = O\left({t_I}^2 / {t_R}^2 + t_A t_R\right)$.

Durante o refinamento das fronteiras são criados n_C modelos, e todos ficam armazenados em memória para classificar cada região, de tamanho t'_R, da imagem. Como explicado anteriormente, a classificação das regiões (cálculo da taxa de bits) é $O(1)$, mas os modelos juntos consomem:

$$\sum_{i=1}^{n_C} O(t_A t_{Si}) = O\left(t_A \sum_{i=1}^{n_C} t_{Si} \right) = O(t_A t_I)$$

Portanto, a memória total consumida pelo algoritmo é $O\left(t_I{}^2 / t_R{}^2 + t_A t_I\right)$.

3.2. Materiais

Para implementar o segmentador, foi utilizada a linguagem JAVA, versão 1.6, por ser uma linguagem de programação moderna, por oferecer um sofisticado tratamento de exceções (o que facilita muito a depuração), por ser orientada a objetos (o que possibilita maior robustez e organização do código) e também por ser a linguagem com a qual o autor desse trabalho tem mais experiência. Além disso, existem diversas IDEs (que favorecem um desenvolvimento mais rápido) gratuitas que oferecem suporte à linguagem, e a mesma oferece uma biblioteca vasta, gratuita e bem documentada.

O computador utilizado para os testes foi um Intel® Pentium® D 2.8 GHz com 0,98GB de RAM, executando um sistema operacional Linux (Ubuntu 8.04). O ambiente de programação utilizado foi o Netbeans 6.0.1, que, além de gratuito, é recomendado pela própria Sun Mycrosystems (proprietária da linguagem JAVA) e pode ser facilmente integrado a diversas ferramentas como o Subversion, a qual é voltada para o controle de versão, o que torna o desenvolvimento mais seguro e organizado.

A única desvantagem de se utilizar JAVA é o aumento no tempo de execução do programa devido ao fato desta linguagem ser interpretada em vez de compilada. Porém o objetivo deste trabalho não é propor um sistema de tempo real ou um sistema crítico, portanto esse acréscimo não acarreta conseqüências graves. Além disso, implementar esse algoritmo em outra

linguagem, como C ou Assembly, diminuiria o tempo de execução por um fator constante, mas a ordem de complexidade do algoritmo continuaria a mesma, e, em contrapartida, o tempo de desenvolvimento provavelmente aumentaria, tendo em vista a pouca experiência do autor deste trabalho com estas linguagens.

Para avaliar o método, foram criados quatro mosaicos (mostrados na Figura 3) a partir de texturas do banco de Brodatz [25], o qual se tornou um padrão de fato na comunidade científica devido ao seu fácil acesso e sua diversidade. O método também foi aplicado a dois outros bancos: o de Timo Ojala e o de Trygve Randen, os quais serão apresentados no próximo capítulo.

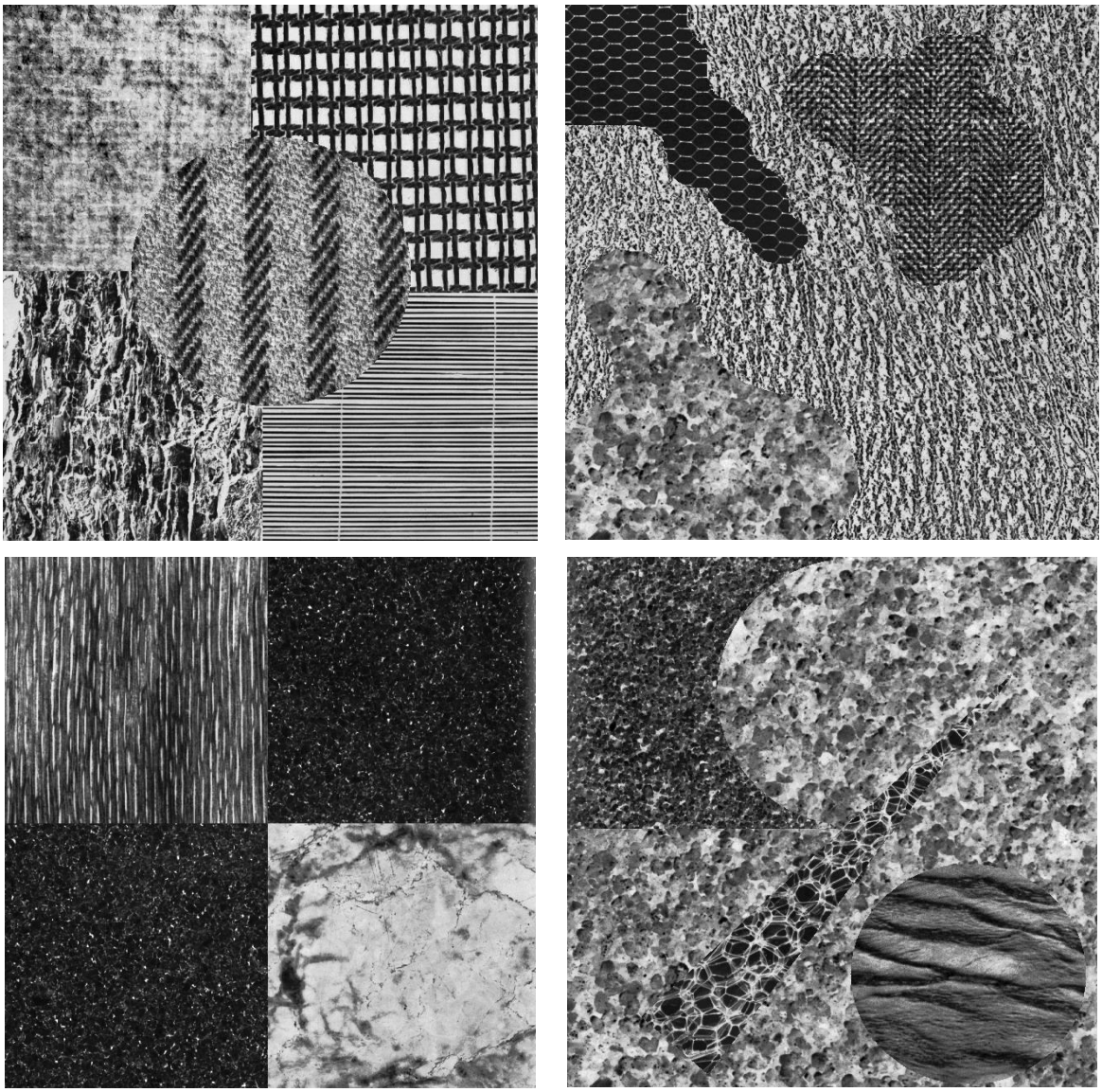

Figura 3: Mosaicos criados. Na primeira linha, mosaicos A e B, na segunda, mosaicos C e D.

Tabela 2

Relação das texturas de Brodatz utilizadas nos mosaicos criados

Mosaico	Texturas utilizadas
A	D19.gif, D20.gif, D11.gif, D12.gif e D49.gif
B	D34.gif, D24.gif, D17.gif e D28.gif
C	D68.gif, D32.gif e D63.gif
D	D29.gif, D28.gif, D111.gif e D37.gif

Fonte: Pesquisa própria

A Tabela 2 indica as texturas utilizadas em cada mosaico criado. O mosaico A contém texturas de diferentes naturezas: as do lado direito são homogêneas, as do lado esquerdo são heterogêneas e a central (circular) é listrada, o que pode confundir um segmentador. O mosaico B apresenta regiões com formas mais complexas, principalmente a do canto inferior esquerdo, cujo contorno se distancia da borda da imagem e então se aproxima novamente, o que provavelmente confundiria um segmentador que se baseasse apenas em regiões grandes.

O mosaico C é simples, tanto pelas formas das regiões quanto pelo tipo de texturas nele contidas, com exceção de algumas áreas da textura do canto inferior esquerdo que, por serem muito escuras, podem ser confundidas com a textura do canto inferior esquerdo (que é a mesma do canto superior direto). O mosaico D apresenta uma região cuja forma mistura linhas retas com curvas, resultando em pontas finas, o que as tornam difíceis de serem detectadas. Vale destacar a região central (triângulo) que, por ser estreita e inclinada, também dificulta a correta detecção de suas fronteiras.

Foi realizada uma busca em órgãos conceituados na área acadêmica (IEEE por exemplo) por artigos na área de segmentação não supervisionada de texturas. No entanto, tendo em vista a dificuldade para conseguir os mosaicos testados nos mesmos e/ou suas respectivas verdades terrestres, o método proposto neste trabalho só pode ser comparado com os trabalhos de Ojala e Pietikäinen [17] e de Randen e Rusoy [52], cujas imagens analisadas são mostradas no capítulo seguinte.

4. RESULTADOS E DISCUSSÃO

Foram realizados testes com o tamanho inicial da região (t_R) igual a 64 porque valores menores acarretam um tempo de execução elevado, o que condiz com a equação de complexidade do método, e valores maiores aumentam o número de regiões contendo mais de uma textura, o que geralmente prejudica o processo de segmentação.

Nas imagens testadas, as taxas de erro são sempre mais altas quando o tamanho final da região (t'_R) é diferente de 8. Se $t'_R > 8$ as regiões finais não são capazes de detectar com precisão os contornos das texturas. Se $t'_R < 8$ não é possível capturar as características das texturas, o que acarreta uma ocorrência elevada de ruído (regiões incorretamente classificadas) na imagem segmentada, conforme mostrado na Figura 4.

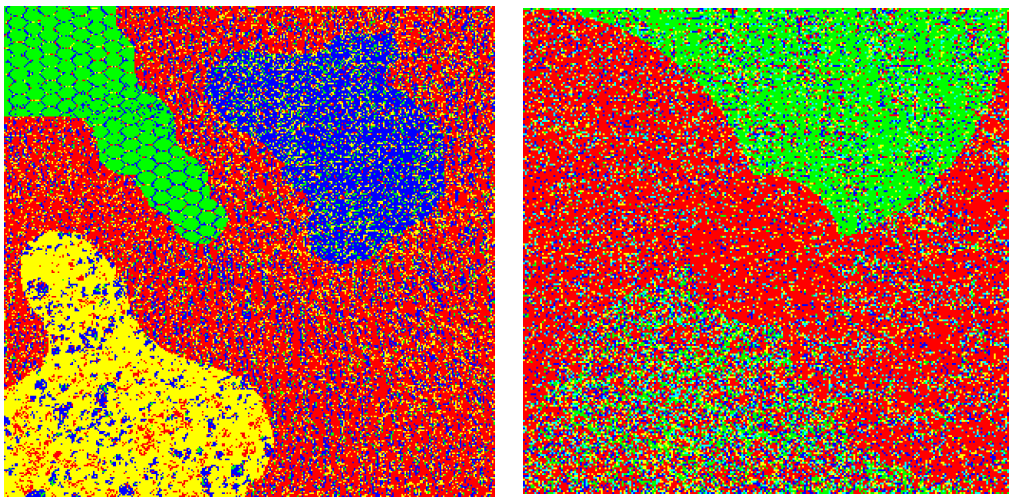

Figura 4: Exemplos do que acontece quando o tamanho da região final é menor que 8.

Os valores testados para o tamanho máximo do contexto (t_C) do PPM foram apenas 0, 1 e 2, visto que, nos bancos utilizados, tamanhos maiores aumentam a taxa de erro. Isso provavelmente acontece porque, dessa forma, a probabilidade dos contextos coincidirem diminui, acarretando um aumento no número de escapes codificados fazendo com que a taxa de bits não seja representativa.

As taxas de erro mostradas nesta seção são calculadas a partir da matriz de confusão da seguinte forma: procuram-se os mapeamentos “nível de cinza da região da verdade terrestre” x “nível de cinza da região da imagem segmentada” que resultam na menor taxa de erro total, a qual é escolhida como a taxa de erro do processo de segmentação.

É possível que uma falha no processo de segmentação resulte em uma imagem segmentada cujo número de regiões (n) seja diferente do número de regiões na verdade terrestre (m). Nestes casos, as regiões da verdade terrestre que não forem mapeadas para nenhuma região da imagem segmentada, por convenção, terão seu índice correspondente mapeados para o índice $n+1$.

Um mapeamento é um par ordenado (i, j), onde i é o índice de uma região da verdade terrestre e j é o índice de uma região da imagem segmentada ou $n+1$ se a imagem segmentada não tiver uma região associada à i-ésima região da verdade terrestre. Um conjunto de mapeamentos é dado por

$$M = \bigcup_{i=1}^{m} \{(i, r_i)\}$$

com as seguintes restrições:

- $r_i \in \{r \in \aleph : 1 \le r \le n+1\}$
- $i \ne j \wedge r_i \le n \wedge r_j \le n \Rightarrow r_i \ne r_j$

A última condição significa que duas regiões distintas da verdade terrestre não podem ser mapeadas para a mesma região da imagem segmentada. No entanto, é possível que duas regiões distintas da verdade terrestre não tenham regiões correspondentes na imagem segmentada.

Seja C uma matriz de confusão com m linhas e $n+1$ colunas, onde $C[i][j]$ representa o número de acertos (em *pixels*) se a i-ésima região da verdade terrestre for mapeada para a j-ésima região da imagem segmentada. Conforme citado anteriormente, quando $j = n+1$, temos que a i-ésima região da verdade terrestre não foi mapeada para nenhuma região da imagem segmentada.

Portanto, $C[i][n+1]=0$ para todo $1 \le i \le m$. A taxa de acerto *A* obtida para um conjunto *M* de mapeamentos é definida neste trabalho como

$$A = \sum_{(i,j) \in M} C[i][j]$$

Seja *T* o número de *pixels* da imagem segmentada e $M_{\max}$ o conjunto de mapeamentos que resultam na maior taxa de acerto, denominada $A_{\max}$. A taxa de erro *E* utilizada para avaliar os resultados deste trabalho pode ser então definida como

$$E = \frac{T - A_{\max}}{T}$$

Tendo em vista que, em todas as imagens analisadas neste trabalho, o número de regiões das imagens segmentadas foi idêntico ao das verdades terrestres, a última coluna (de índice $n+1$, cujos valores são todos iguais a zero) das matrizes de confusão foi omitida para facilitar a visualização.

O primeiro banco de imagens testado foi o de Ojala [17]. A Tabela 3 mostra as taxas de erro obtidas pelo método de Ojala [17] no mesmo banco e a Tabela 4 mostra as taxas de erro obtidas pelo método proposto neste trabalho ao variar o tamanho do contexto.

Tabela 3

Relação das taxas de erro (%) obtidas pelo método de Ojala no banco de Ojala [17].

Mosaico	Erro (%)
#1	1,7
#2	1,2
#3	1,9
#4	1,4
#5	2,1

Fonte: Ojala [17]

Tabela 4

Relação das taxas de erro (%) e dos tempos de execução obtidos no banco de Ojala pelo método proposto quando $t_R = 64$ e $t'_R = 8$.

t_C	Mosaico	Erro (%)	Tempo (s)
0	#1	53.95	172
0	#2	42.18	209
0	#3	10.61	144
0	#4	1.08	13
0	#5	16.58	69
1	#1	2.32	180
1	#2	43.17	233
1	#3	8.98	146
1	#4	25.69	14
1	#5	13.89	73
2	#1	5.38	214
2	#2	44.19	266
2	#3	18.71	159
2	#4	26.93	14
2	#5	13.89	74

Fonte: Pesquisa própria

A Figura 5 mostra, para cada mosaico do banco de Ojala, a imagem segmentada (pelo método proposto neste trabalho) que apresentou a menor taxa de erro, juntamente com a respectiva imagem original, sementes e verdade terrestre. A Figura 6, por sua vez, mostra as matrizes de confusão dos resultados da Figura 5 que apresentaram as maiores taxas de erro.

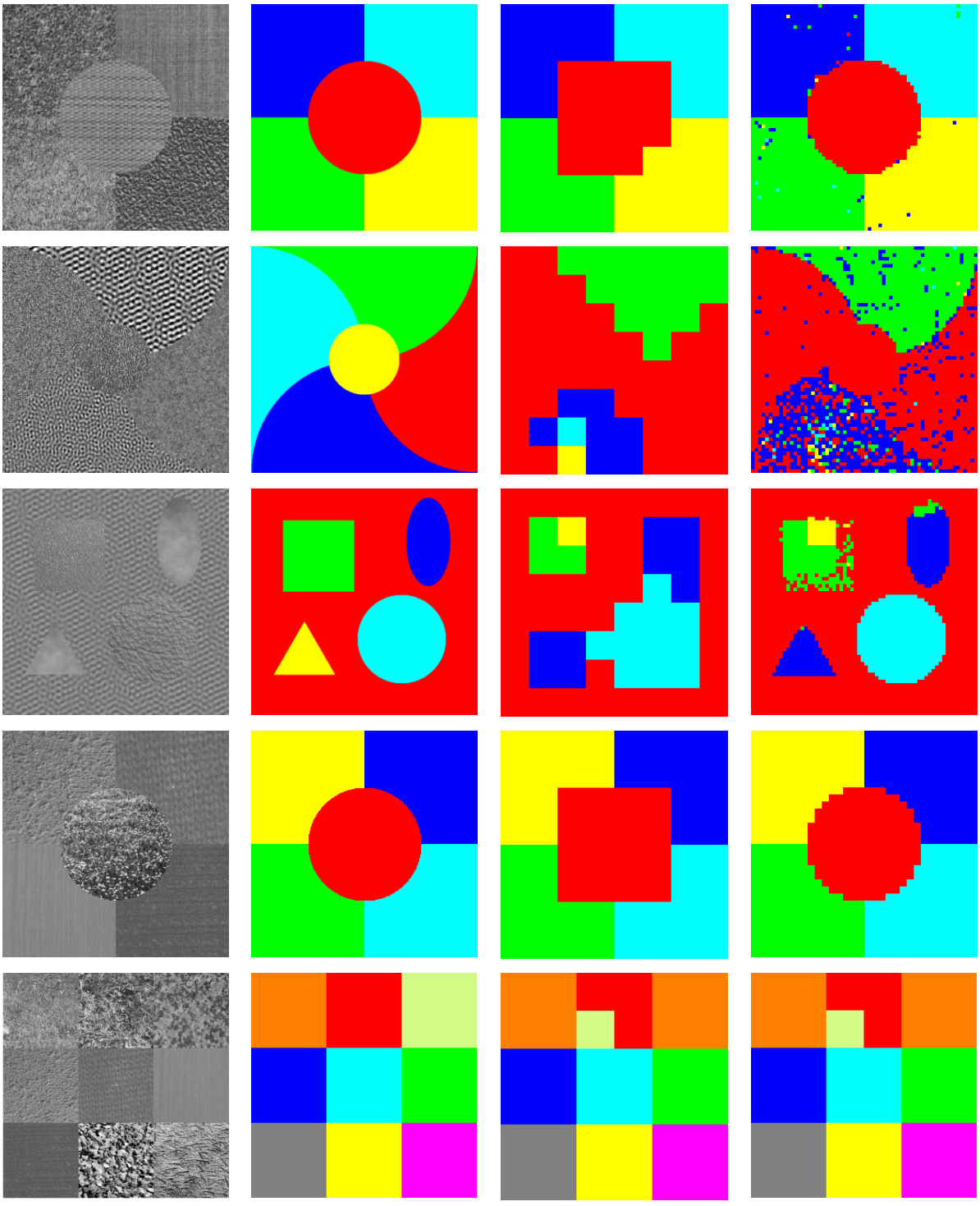

Figura 5: imagens originais do banco de Ojala na primeira coluna, verdades terrestres do mesmo banco na segunda, a terceira coluna contém as sementes encontradas e a quarta coluna, as imagens segmentadas. A primeira linha corresponde ao mosaico #1,a segunda ao mosaico #2 e assim sucessivamente.

	38340	3456	1152	16352	1472
	4672	54960	253	175	200
	1981	258	64	52652	0
	7444	378	3	58195	56
	171	148	0	19762	0

	13662	1272	0	767	0
	0	16256	0	5889	3776
	0	0	31292	125	0
	1642	139	1348	177399	192
	8184	61	0	140	0

	16384	0	0	0	0	0	0	0	0
	0	16384	0	0	0	0	0	0	0
	0	0	16384	0	0	0	0	0	0
	0	0	0	16384	0	0	0	0	0
	0	0	0	0	0	0	0	16384	0
	0	0	0	0	4096	12288	0	0	0
	0	0	0	0	0	0	16384	0	0
	0	0	0	0	0	0	0	16384	0
	0	0	0	0	0	0	0	0	16384

Figura 6: matrizes de confusão de alguns resultados da Figura 5 – mosaico #2 (matriz superior), mosaico #3 (matriz central) e mosaico #5 (matriz inferior) As linhas correspondem às cores da verdade terrestre e as colunas, às do resultado.

É fácil perceber que, sempre que as sementes estão bem posicionadas, a segmentação é bem sucedida (taxa de erro < 3%). No caso do mosaico #2, o método sempre confunde as regiões laterais com a região central porque as regiões localizadas na parte inferior da imagem são muito distantes entre si, sob o ponto de vista da medida de distância proposta em 3.1.1.

Dessa forma, independentemente dos parâmetros do método, durante a escolha das sementes, as regiões da parte superior são corretamente agrupadas, mas as da parte inferior do mosaico #2 são as últimas a serem agrupadas, resultando no indesejado agrupamento das regiões

central, lateral esquerda e lateral direita. Portanto, a medida de distância utilizada não é adequada para esta imagem.

Durante a segmentação do mosaico #3, a região triangular e a elíptica foram atribuídas a um mesmo grupo, o que demonstra que o método proposto também identifica regiões desconexas. Essa característica, embora tenha prejudicado a segmentação nesse caso em particular, pode ser útil em outros.

A Figura 7 mostra qual seria o resultado se a verdade terrestre do mesmo mosaico fosse modificada. Nesse caso, a taxa de erro seria 4,69% com contexto zero, enquanto que outro método que não oferecesse suporte a regiões desconexas, caso determinasse com precisão máxima todos os contornos, apresentaria um erro de, no mínimo, 3,20% visto que as regiões citadas seriam atribuídas a grupos diferentes.

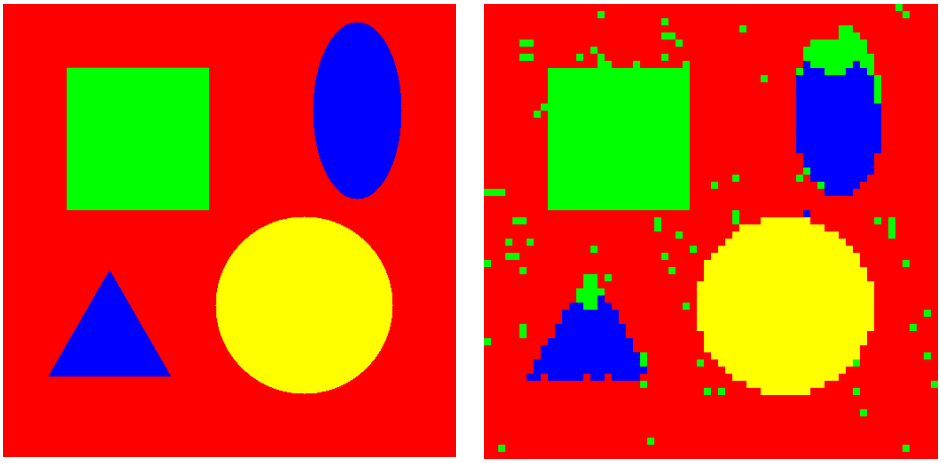

Figura 7: nova verdade terrestre à esquerda e, à direita, novo resultado da segmentação do mosaico #3.

A taxa de erro resultante da segmentação do mosaico #5 deve-se unicamente à escolha equivocada de algumas sementes. Contudo, apesar desta falha, os contornos das demais regiões foram corretamente determinados.

O segundo banco testado (Figuras 9, 10 e 11) está disponível na Internet [53]. Dois mosaicos deste banco, Nat-10.png e D004D084.png (Figura 9), foram analisados por Trygve Randen [52]; a segunda resultou numa taxa de erro de 1,3% enquanto que a primeira resultou na imagem da Figura 8, cuja taxa de erro não foi indicada, mas é obviamente maior que zero. Em ambos os casos, o método proposto neste trabalho obteve 0% (zero) de erro.

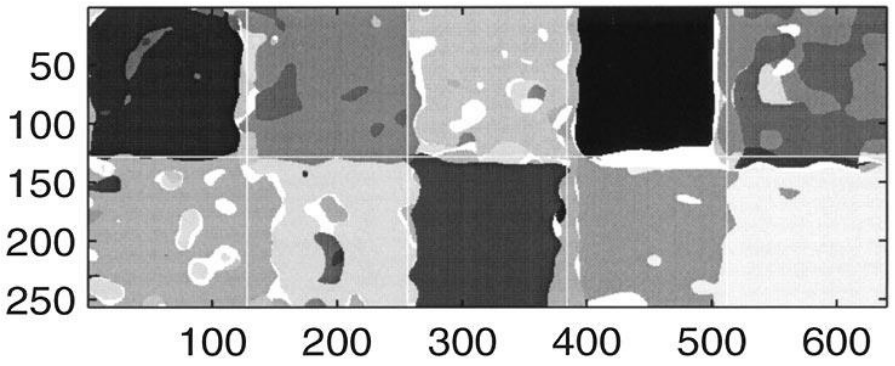

Figura 8: Nat-10.png segmentada pelo método proposto por Trygve Randen [52].

As Tabelas 5 e 6 juntas mostram o tempo de execução obtido pelo método proposto quando aplicado a todas as imagens do banco de Trygve Randen. A Tabela 5 mostra apenas os tempos de execução porque os resultados de todas as imagens nela contidas apresentaram uma taxa de erro de 0%. As demais imagens do banco constam na Tabela 6, juntamente com suas respectivas taxas de erro (positivas). Em ambas as Tabelas $t_R = 64$ e $t'_R = 8$.

Tabela 5

Relação dos tempos de execução necessários para segmentar
as imagens do banco de Trygve Randen que resultaram em taxa de erro zero.

t_C	Mosaico	Tempo(s)
0	D004D084.png	53
0	D005D092.png	54
0	D008D084.png	51
0	D012D017.png	55
0	Nat-10.png	115
0	Nat-10v.png	119
1	D004D084.png	56
1	D005D092.png	54
1	D008D084.png	51
1	D012D017.png	57
1	Nat-10.png	119
1	Nat-10v.png	120
2	D004D084.png	63
2	D005D092.png	60
2	D008D084.png	55
2	D012D017.png	63
2	Nat-10.png	126
2	Nat-10v.png	127

Fonte: Pesquisa própria

Tabela 6

Relação das taxas de erro e dos tempos de execução obtidos ao segmentar as imagens do banco de Trygve Randen que apresentaram taxa de erro positiva.

t_C	Mosaico	Erro (%)	Tempo (s)
0	D17D55.gif	1.34	13
0	Nat-16b.gif	35.76	227
0	Nat-16c.png	2.91	301
0	Nat-16v.png	3.04	301
0	Nat-5b.gif	23.36	15
0	Nat-5c.png	2.97	19
0	Nat-5m.png	2.73	19
0	Nat-5v.png	3.29	19
0	Nat-5v2.png	3.25	19
0	Nat-5v3.png	3.31	19
1	D17D55.gif	3.49	13
1	Nat-16b.gif	35.14	217
1	Nat-16c.png	14.46	306
1	Nat-16v.png	12.71	306
1	Nat-5b.gif	33.43	15
1	Nat-5c.png	16.46	20
1	Nat-5m.png	14.01	18
1	Nat-5v.png	14.72	19
1	Nat-5v2.png	14.82	19
1	Nat-5v3.png	13.51	19
2	D17D55.gif	5.39	16
2	Nat-16b.gif	40.35	248
2	Nat-16c.png	19.11	320
2	Nat-16v.png	19.16	311
2	Nat-5b.gif	34.68	18
2	Nat-5c.png	17.90	22
2	Nat-5m.png	17.61	21
2	Nat-5v.png	17.80	21
2	Nat-5v2.png	17.80	21
2	Nat-5v3.png	17.90	21

Fonte: Pesquisa própria

A Figura 9 mostra as imagens que constam na Tabela 5 e suas respectivas verdades terrestres, as quais são idênticas aos resultados visto que a taxa de erro foi 0%. As sementes selecionadas também não são mostradas aqui porque, coincidentemente, foram idênticas às verdades terrestres.

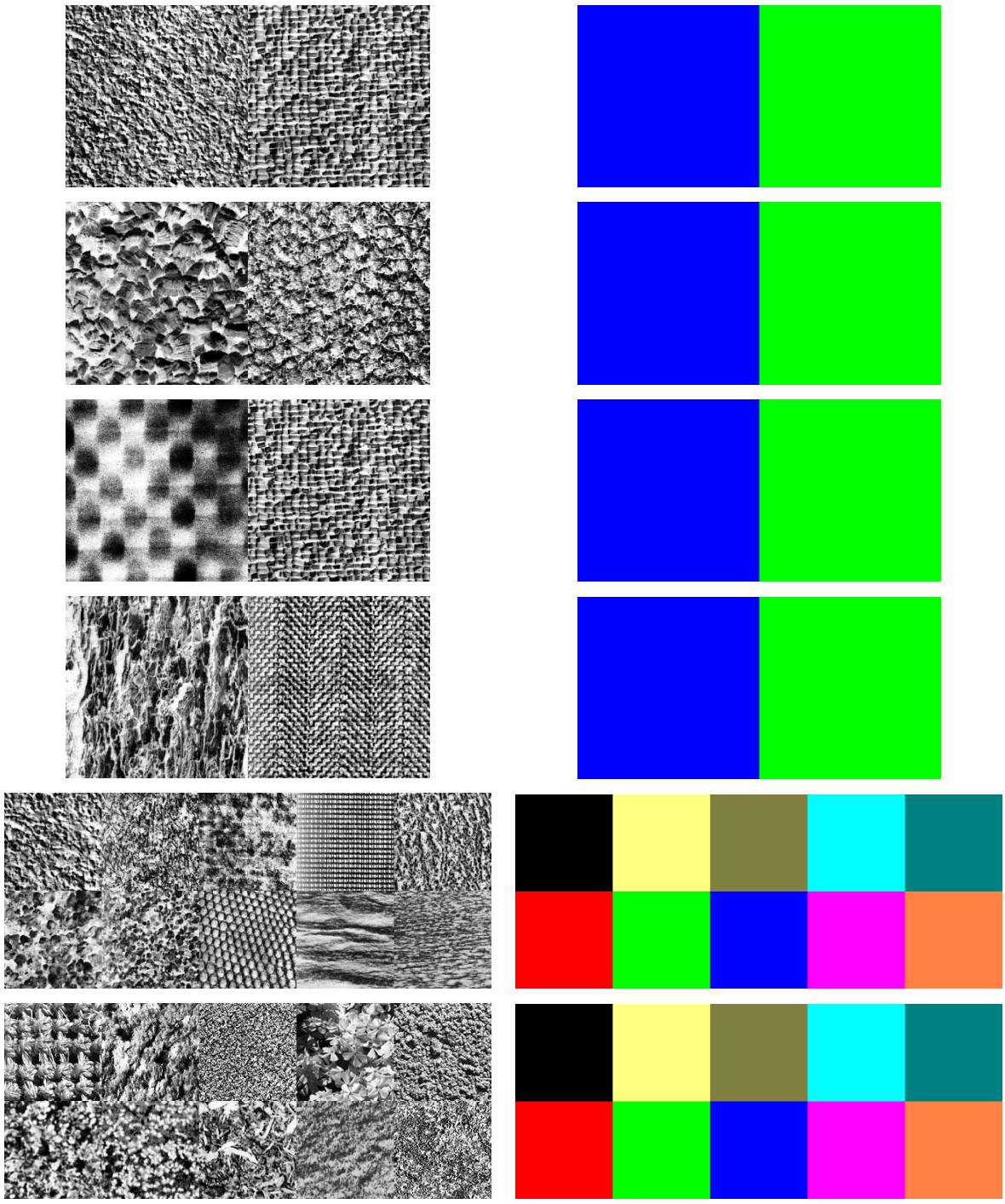

Figura 9: imagens que resultaram em taxa de erro zero (primeira coluna) e suas respectivas verdades terrestres (segunda coluna). De cima para baixo: D004D084.png, D005D092.png, D008D084.png, D012D017.png, Nat-10.png e Nat-10v.png

A igualdade entre as sementes e a verdade terrestre não é uma condição necessária para atingir 100% de acerto. Também não se trata de uma condição suficiente, como exemplifica a Figura 5. No entanto, é provável que a taxa de acerto seja alta nesses casos visto que o modelo que representa cada semente será mais fidedigno.

Os resultados da Tabela 6 obtidos quando $t_C = 0$, que são os melhores sob o ponto de vista da taxa de erro, são mostrados nas Figuras 10 e 11 juntamente com as verdades terrestres e sementes escolhidas correspondentes.

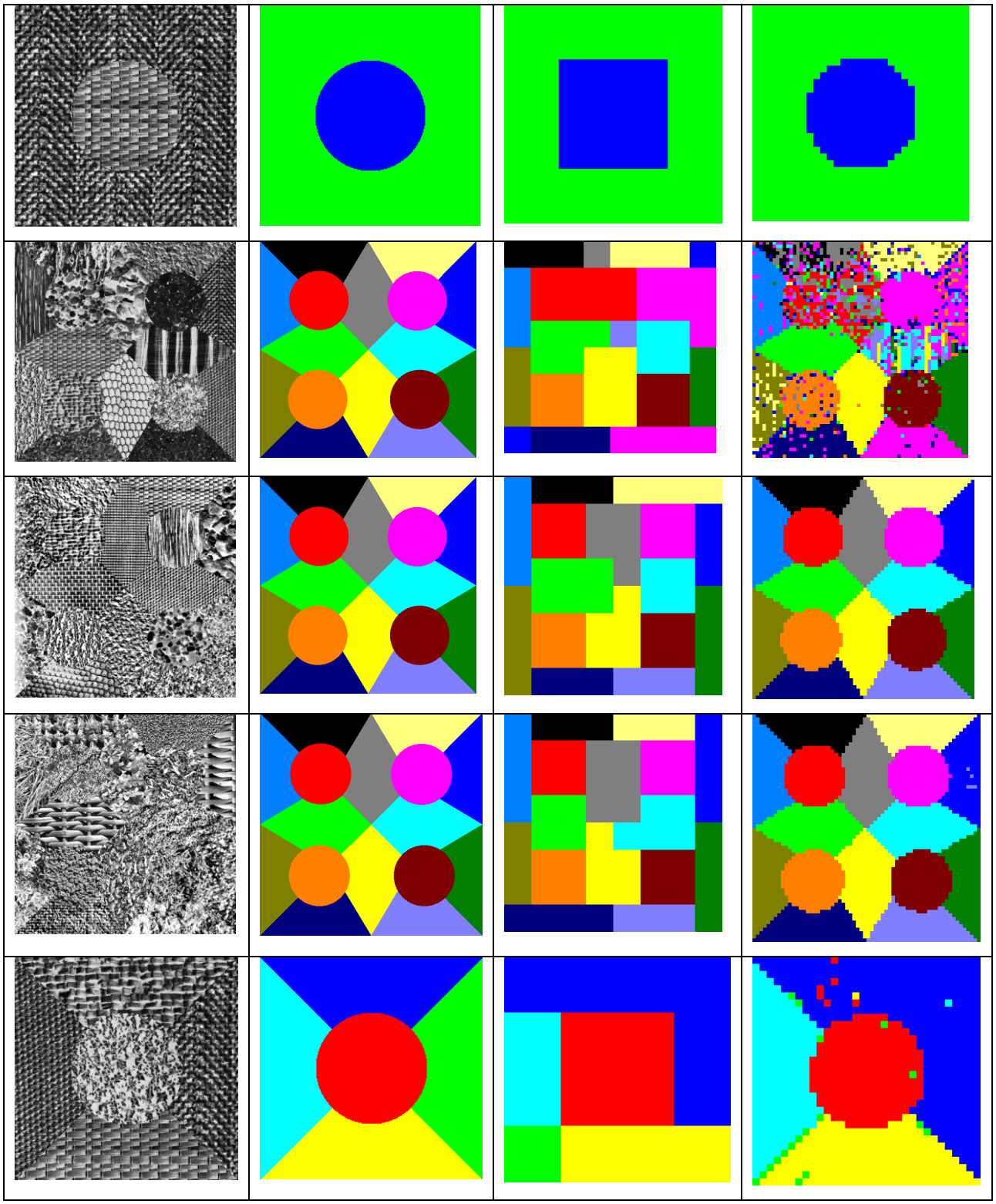

Figura 10: imagens D17D55.gif, Nat-16b.gif, Nat-16c.png, Nat-16v.png e Nat-5b.gif na primeira coluna e, nas colunas seguintes, as verdades terrestres, as sementes e os resultados.

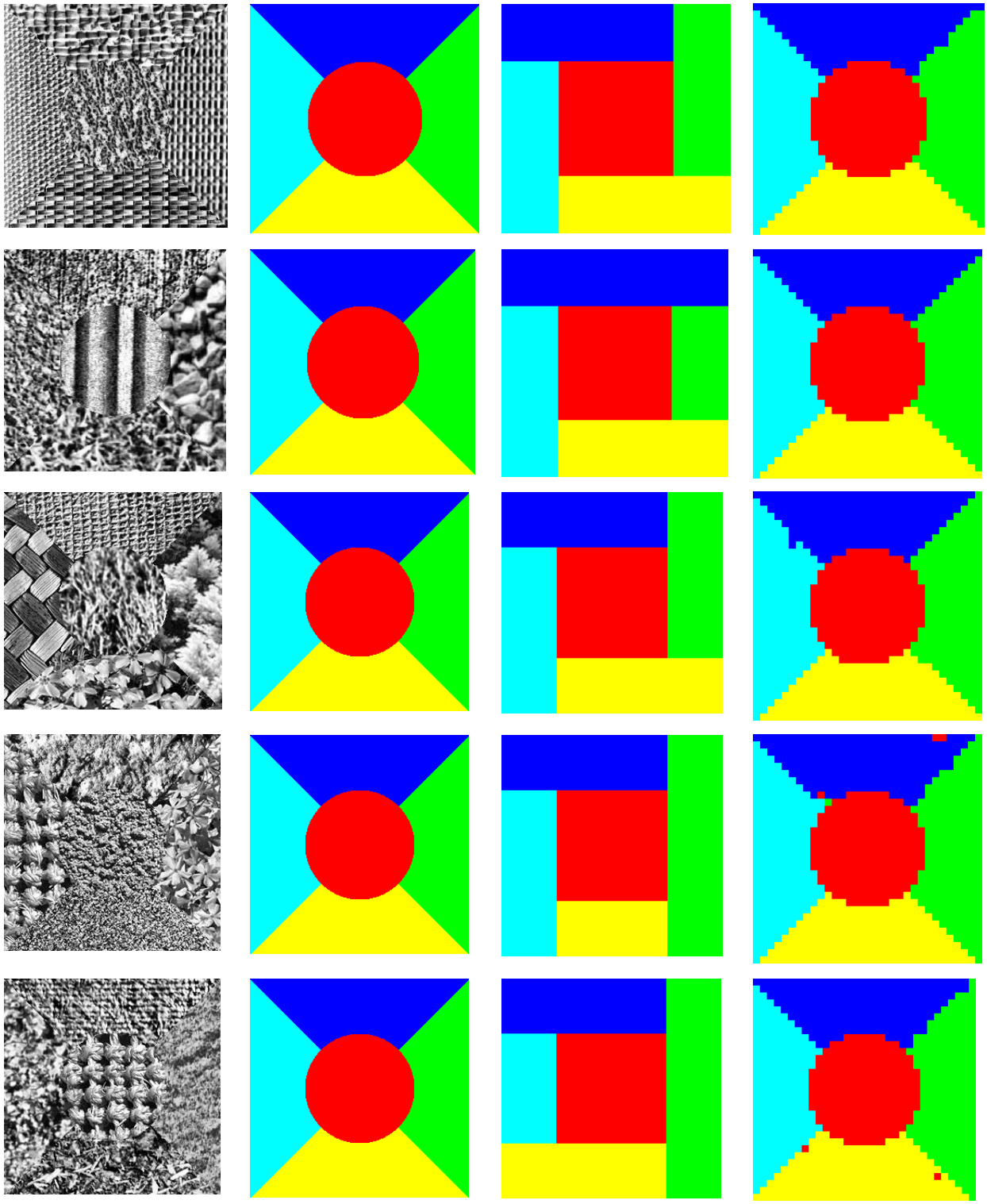

Figura 11: imagens Nat-5c.png, Nat-5m.png, Nat-5v.png, Nat-5v2.png e Nat-5v3.png na primeira coluna e, nas colunas seguintes, as verdades terrestres, as sementes e os resultados.

Conforme mostrado na Figura 12, a verdade terrestre original da D17D55.gif foi corrigida porque contém, ao redor do círculo central, uma pequena quantidade de ruído, o qual é difícil de ser percebido visualmente por causa da semelhança entre a cor dos *pixels* ruidosos e a dos vizinhos. A análise dos resultados foi baseada na verdade terrestre corrigida.

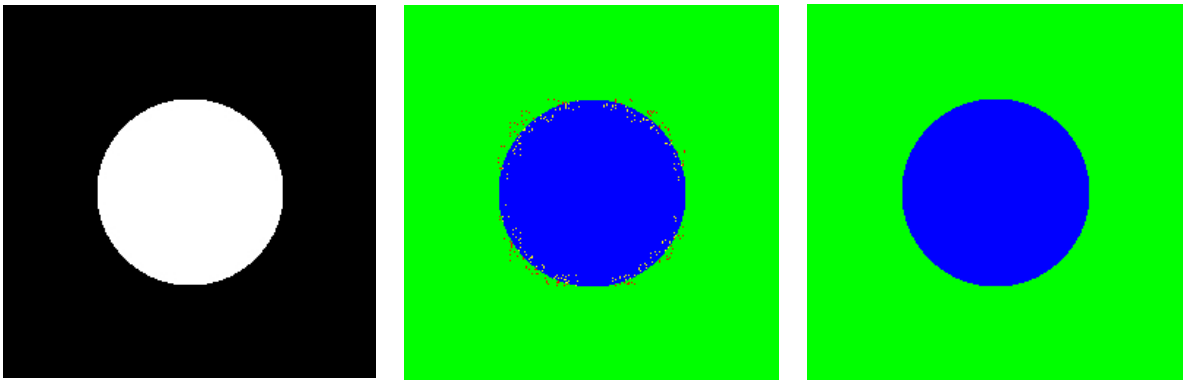

Figura 12: correção da verdade terrestre da imagem D17D55.gif. À esquerda, verdade terrestre original; ao centro, mudança nas cores para destacar o ruído; e, à direita, verdade terrestre corrigida.

As matrizes de confusão dos resultados das Figuras 10 e 11 que apresentaram taxas de erro muito acima da média (Nat-16b.gif e Nat-5b.gif) são mostradas na Figura 13.

	12705	72	0	541	64
	12466	72	64	154	288
	224	476	12468	121	10
	77	159	55	12518	40
	0	373	21	106	12462

	5574	181	284	150	28	28	895	0	0	5551	26	54	2331	138	0	0
	0	12573	345	38	0	139	128	64	28	0	0	1	1088	478	147	28
	241	82	2619	74	0	0	607	0	0	0	0	0	9281	2370	47	0
	47	14	0	14229	0	0	23	153	0	0	11	0	180	41	0	0
	240	66	36	64	13540	108	0	0	0	43	0	42	1116	35	0	20
	0	81	36	0	326	18742	7	56	133	54	0	0	69	73	235	2
	81	712	257	1465	0	92	6533	86	0	384	3365	46	3408	356	2867	0
	64	112	128	315	0	192	9	13694	47	405	0	72	222	26	154	0
	0	0	835	0	0	143	0	42	11089	0	0	0	0	82	0	2816
	1	999	92	106	192	2137	104	64	18	4695	1825	7550	1071	92	0	557
	23	562	0	173	0	0	163	53	0	0	0	38	14051	0	64	0
	73	949	64	0	441	1100	247	19	0	2480	965	7450	1333	83	0	236
	79	489	122	0	0	12	277	0	2	18	16	95	14286	23	1	20
	0	475	615	90	65	115	64	139	78	512	0	63	509	12604	77	34
	41	242	25	0	0	17	196	30	17	43	0	0	47	175	19448	0
	0	127	1262	0	64	23	27	0	2220	23	0	13	288	0	0	11327

Figura 13: matrizes de confusão dos resultados das imagens Nat-16b.gif e Nat-5b.gif da Figura 10. As linhas correspondem às cores da verdade terrestre e as colunas, às do resultado.

O terceiro banco testado foi o apresentado na seção 3.2. A Tabela 7 mostra as taxas de erro e os tempos de execução obtidos pelo método proposto quando aplicado ao referido banco ao passo que a Figura 14 mostra, para cada imagem do mesmo, a verdade terrestre, as sementes selecionadas e o resultado final quando $t_C = 1$, caso que proporcionou, na média, as menores taxas de erro.

Tabela 7

Relação das taxas de erro e dos tempos de execução obtidos ao segmentar as imagens do banco da seção 3.1.

t_C	Mosaico	Erro (%)	Tempo (s)
0	A	5.28	482
0	B	4.18	452
0	C	4.44	450
0	D	15.34	482
1	A	1.32	463
1	B	1.91	441
1	C	1.08	432
1	D	3.24	402
2	A	6.93	500
2	B	13.39	505
2	C	0.76	462
2	D	9.36	466

Fonte: Pesquisa própria.

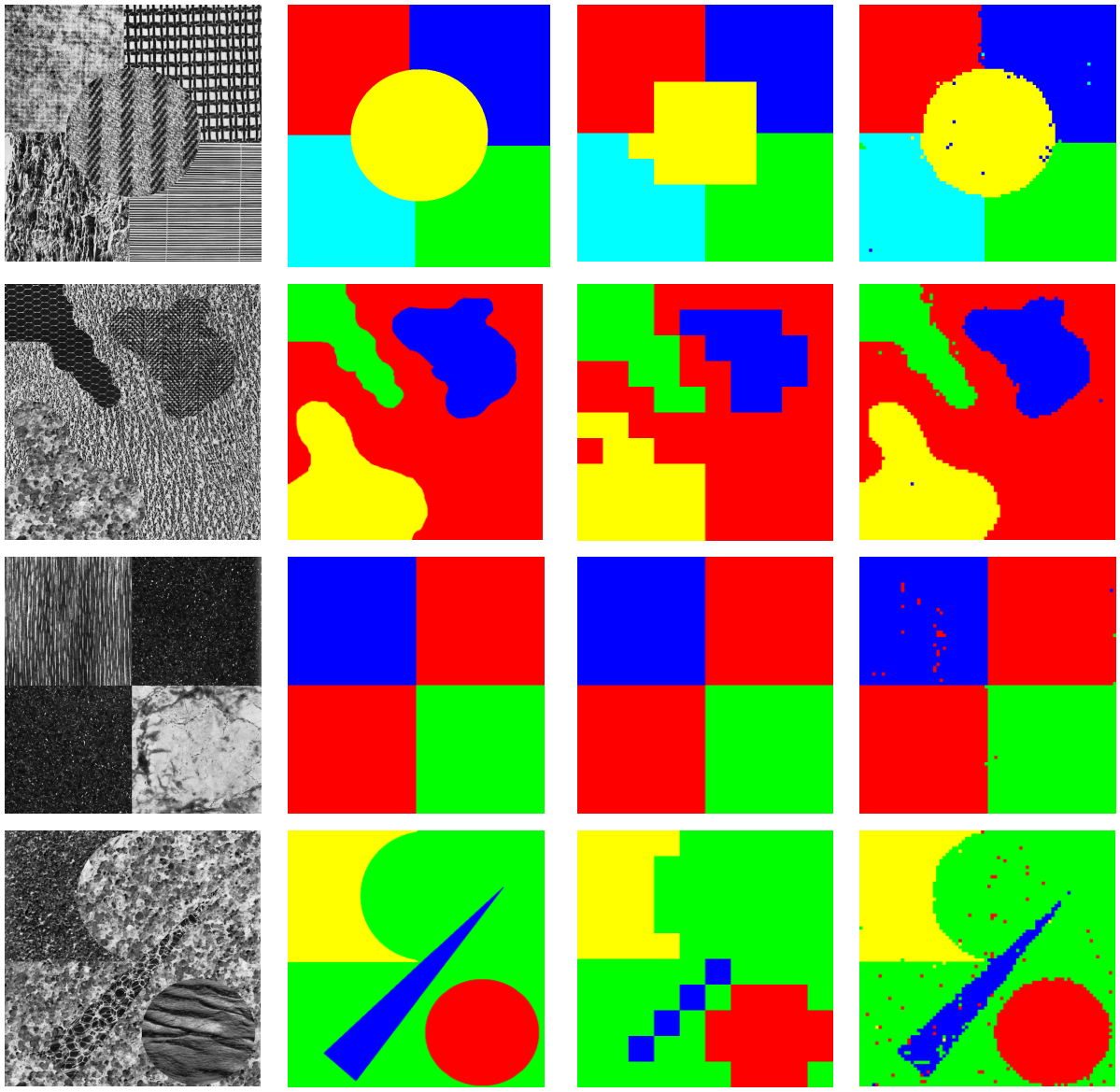

Figura 14: Mosaicos A, B, C e D na primeira coluna, verdades terrestres na segunda, sementes na terceira e resultados na quarta.

No mosaico A, o valor de t_R possibilitou a identificar as listras do círculo central como uma textura única. Esse é um dos exemplos em que entra em questão a subjetividade, pois há pessoas que identificam duas texturas distintas no círculo central enquanto outras reconhecem apenas uma textura listrada.

Como o círculo central do mosaico A foi formado a partir de uma única textura de Broadtz (D11.gif), a verdade terrestre considerada consta na Figura 14. Para identificar duas

texturas na região central, é necessário utilizar um valor menor de t_R para minimizar as chances de uma região inicial estar localizada na fronteira entre elas.

A Figura 15 mostra o resultado da segmentação do mosaico A quando $t_R = 32$ e o número de classes é definido como 6. Neste caso, não é possível medir de forma objetiva a taxa de erro visto que o círculo central, que está sendo interpretado como uma junção de duas texturas distintas, é, na verdade, uma única textura de Brodatz.

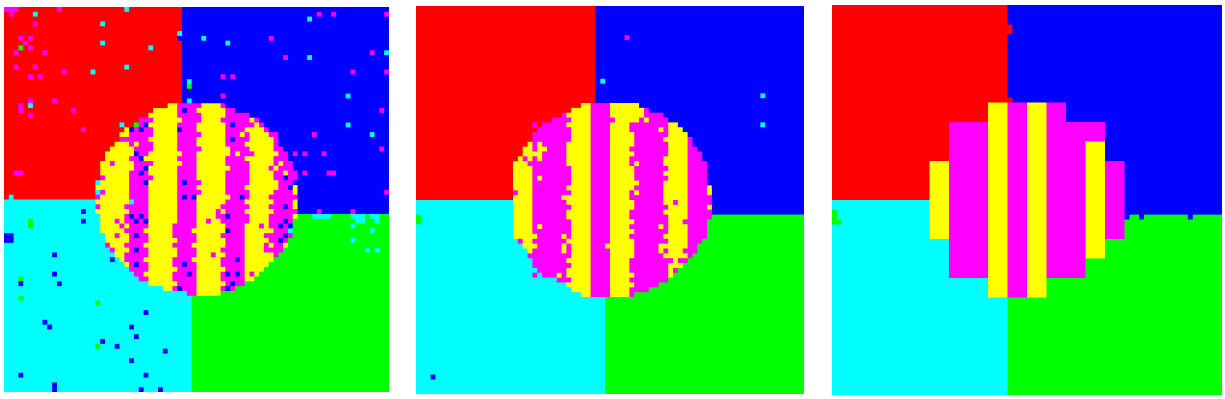

Figura 15: Resultados da segmentação do mosaico A quando o tamanho da região inicial é definido como 32 e o número de classes é definido como 6. O tamanho do contexto é 0 na imagem da esquerda, 1 na central e 2 na da direita.

Durante a segmentação do mosaico B, o fato de o método proposto aceitar regiões desconexas, mais uma vez, possibilitou melhores resultados, pois, dessa forma, foi possível identificar que, dentro da semente amarela, havia parte da textura representada pela semente vermelha, propiciando uma detecção mais precisa do contorno da região amarela.

O mosaico C, por apresentar regiões maiores, resultou nas menores taxas de erro. O mosaico D, apesar das formas pouco convencionais (como o triângulo fino e inclinado), resultou em uma taxa de erro próxima a 3%. O fato de utilizar o tamanho 8 x 8 no processo final da segmentação permite que até mesmo as extremidades pontiagudas das regiões (amarela e azul) possam ser detectadas de forma satisfatória.

5. CONCLUSÕES

Neste trabalho foi apresentado um método de segmentação não supervisionada que utiliza como medida de distância entre regiões as taxas de bits obtidas pelos modelos PPM extraídos das mesmas. Este era o objetivo geral deste trabalho.

Conforme descrito na seção 1.1, o primeiro objetivo específico deste trabalho era implementar um compressor PPM e adaptá-lo à natureza do problema. Este objetivo foi alcançado, pois um PPM próprio foi implementado e adaptado à leitura de regiões de imagens conforme descrito na seção 3.1.1.

O segundo objetivo específico era decidir como dividir e agrupar as regiões da imagem. Um algoritmo guloso de agrupamento, que une as regiões mais próximas a cada iteração, foi utilizado para extrair as sementes, as quais, idealmente, possuem as principais características das texturas presentes na imagem.

Para identificar com maior precisão as fronteiras das texturas, a imagem é então dividida em regiões menores, as quais são atribuídas aos modelos das sementes mais próximas.

Nenhuma informação espacial é utilizada ao agrupar ou classificar as regiões. Por esse motivo, o método é robusto a diferentes formas (contornos) de regiões, agrupando corretamente até mesmo regiões desconexas. O método proposto só utiliza informação espacial ao calcular a distância entre regiões, pois o modelo PPM utiliza o contexto (no caso, *pixels* vizinhos).

O terceiro e último objetivo específico era avaliar empiricamente os resultados. As avaliações subjetiva (discussão) e objetiva (taxas de acerto) das imagens segmentadas são apresentadas no capítulo 4. Dessa forma, pode-se dizer que todos os objetivos do presente trabalho foram alcançados.

Um trabalho futuro é identificar automaticamente o número ótimo de classes para cada imagem. A principal desvantagem do método ainda é a ordem de complexidade, a qual poderá ser reduzida se for utilizado PPM binário para calcular as distâncias entre as regiões.

O método de segmentação proposto neste trabalho foi testado com vários mosaicos da literatura, obtendo uma taxa de erro inferior a 3,5% na maioria deles e atingindo, em alguns casos, 0% de erro. A análise dos casos que resultaram em uma taxa de erro acima da média será objeto de investigações futuras.

REFERÊNCIAS

[1] ZISSERMAN, A.; HARTLEY, R. **Multiple view geometry in computer vision**. Cambridge: Cambridge University Press, 2000.

[2] SEBE, N. et al. Machine Learning in Computer Vision. **Springer**, 2005.

[3] BASKARAN, R.; DEIVAMANI, M.; KANNAN, A. A Multi Agent Approach for Texture Based Classification and Retrieval (MATBCR) using Binary Decision Tree. **International Journal of Computing & Information Sciences**, v. 2, n.1, abr. 2004.

[4] PAYNE, J. S.; HEPPLEWHITE, L.; STONHAM, T. J. **Applying perceptually-based metrics to textural image retrieval methods**. In: PROC SPIE ELECTRONIC IMAGING, 2000, San Jose. San Jose: [s.n.], 2000. v. 3959, p. 423-433.

[5] LIAPIS, S.; TZIRITAS, G. Color and Texture Image Retrieval Using Chromaticity Histograms and Wavelet Frames. **IEEE Transactions on multimedia**, v. 6, p. 676-686, 2004.

[6] FLICKNERM, M. et al. Query by image and video content: the qbic system. **IEEE Computer**, set. 1995.

[7] OGLE, V.; STONEBRAKER, M. Chabot: Retrieval from a relational database of images. **IEEE Computer**, set. 1995.

[8] PAYNE, J.S.; HEPPLEWHITE, L.; STONHAM, T.J. **Texture, human perception, and information retrieval measures**. In: PROCEEDINGS OF THE ACM SIGIR MF/IR WORKSHOP, 2000.

[9] BATISTA, L. V.; MEIRA, M. M. Texture Classification Using the Lempel-Ziv-Welch Algorithm. **Lecture Notes in Computer Science**, Berlin, v. 3171, p. 444 - 453, 2004.

[10] RANGUELOVA, E. **Segmentation of Textured Images on Three-Dimensional Lattices**. Disponível em: <homepages.cwi.nl/~ely/chapter2.pdf> Acesso em: 15 mar. 2010.

[11] FAUGERAS O.; PRATT, W. Decorrelation Methods of texture feature extraction, **IEEE Trans. Pattern Anal. Machine Intell**, v. 2, p. 323-332, jul. 1980.

[12] JAIN, A. K.; KARU, K. Learning texture discrimination masks. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v.18, p.195-205, 1996.

[13] KARU, K.; JAIN, A. K.; BOLLE, R. M. Is there any texture in the image?, **Pattern Recognition**, v.29, n. 9, p.1437-1446, 1996.

[14] SANTAMARIA, C.; BOBER, M.; SZAJNOWSKI, W. **Texture analysis using level-crossing statistics**. In: PROCEEDINGS INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION, Cambridge, UK, 2004.

[15] WU. J. **Rotation Invariant Classification of 3D Surface Texture Using Photometric Stereo**. Heriot-Watt University. Disponível em: <http://www.macs.hw.ac.uk/texturelab/publications/phds_mscs/JW/chapter2.pdf> Acesso em: 15 mar. 2010.

[16] PUIG, D.; GARCIA, M. A. Pixel-based texture classification by integration of multiple feature extraction methods evaluated over multisized windows, **International Journal of Pattern Recognition and Artificial Intelligence**, v. 21, n.7, p.1159-1170, 2007.

[17] OJALA T.; PIETIKÄINEN, M. Unsupervised Texture Segmentation Using Feature Distributions, **Pattern Recognition**, v.32, p.477-486, 1999.

[18] JAIN, A. K.; MURTY, M. N.; FLYNN, P. J. Data clustering: a review. **ACM Computing Surveys (CSUR)**, v. 31 n. 3, p. 264-323, set. 1999.

[19] FEDER, M.; MERHAV, N. Relations between entropy and error probability. **IEEE Transactions on Information Theory**, 1994.

[20] DAWY Z. et al. **Mutual Information Based Distance Measures for Classification and Content Recognition with Applications to Genetics**. [S.l.]: ICC, 2005.

[21] NATIONAL center for biotechnology information - NCBI. Disponível em: <http://www.ncbi.nlm.nih.gov/> Acesso em: 14 ago. 2010.

[22] NOBRE NETO, F. D. **Atribuição Automática de Autoria de Obras da Literatura Brasileira**. 2010. Dissertação (Mestrado em Ciência da Computação) – CCEN, UFPB João Pessoa.

[23] BARUFALDI, B. et al. **Classificação Automática de Textos por Período Literário Utilizando Compressão de Dados Através do PPM-C**. In: BRAZILIAN SYMPOSIUM IN INFORMATION AND HUMAN LANGUAGE TECHNOLOGY, 7, 2009, São Carlos.

[24] BOBICEV, V. Text Classification Using Word-Based PPM Models. **Computer Science Journal of Moldova**, v.14, n. 2, 2006.

[25] BRODATZ, P. **Textures:** A Photographic album for artists and designers. New York: Dover, 1966.

[26] HONÓRIO, T. C. S.; BATISTA, L. V.; DUARTE, R. C. M. **Texture Classification Using Prediction by Partial Matching Models**. In: WORKSHOP DE VISÃO COMPUTACIONAL, 5, 2009.

[27] CLEARY, J. G.; WITTEN, I. H. Data compression using adaptative coding and partial string matching, **IEEE Trans. Commun**., v. 32, n. 4, p. 396-402, abr. 1984.

[28] DUDA, R. O.; HART, P. E. **Pattern Classification and Scene Analysis**. New York: John Wiley and Sons, Inc., 1973.

[29] DUDA, R. O.; HART, P. E.; STORK, D. G. **Pattern Classification**. [S.l.]: Wiley Interscience. 2000.

[30] GOWDA, K. C.; DIDAY, E. Symbolic clustering using a new dissimilarity measure. **IEEE Trans. Syst. Man Cybern**, v. 22, p. 368-378, 1992.

[31] MICHALSKI, R.; STEPP, R. E.; DIDAY, E. Automated construction of classifications: conceptual clustering versus numerical taxonomy. **IEEE Trans. Pattern Anal. Mach. Intell,** PAMI-5, set. 1983.

[32] DIDAY, E. **The symbolic approach in clustering**. Classification and Related Methods, Ed. Amsterdam: North-Holland Publishing Co. 1988.

[33] XU R.; WUNSCH, D.I.I. Survey of clustering algorithms. **IEEE Trans. Neural Networks**, p. 645–678, 2005.

[34] MOHROR, K.; KARAVANIC, K. L. Evaluating Similarity-based Trace Reduction Techniques for Scalable Performance Analysis. **Proc. ACM/IEEE** SC09 Conference Portland, OR, USA, nov. 2009.

[35] MAO, J. and JAIN, A. K. A self-organizing network for hyperellipsoidal clustering (HEC). **IEEE Trans. Neural Netw.** v. 6, p. 296-317, 1996.

[36] ANDERBERG, M. R. **Cluster Analysis for Applications**. New York: Academic Press, Inc., 1973.

[37] ZHANG, T.; RAMAKRISHNAN, R.; LIVNY, M. BIRCH: An efficient data clustering method for very large databases, **Proc. ACM SIGMOD Conf. Management of Data**, p. 103–114, 1996.

[38] GUHA, S.; RASTOGI, R.; SHIM, K. CURE: An efficient clustering algorithm for large databases, **Proc. ACM SIGMOD Int. Conf. Management of Data**, p. 73–84, 1998.

[39] GUHA, S.; RASTOGI, R.; SHIM, K. ROCK: A robust clustering algorithm for categorical attributes. **Inf. Syst.**, v. 25, n. 5, p. 345–366, 2000.

[40] DUBES, R. C. How many clusters are best? – an experiment. **Pattern Recogn**. 1987.

[41] BALL, G.; HALL, D. A clustering technique for summarizing multivariate data, **Behav. Sci.**, v. 12, p. 153–155, 1967.

[42] SHARAN, R.; SHAMIR, R. CLICK: A clustering algorithm with applications to gene expression analysis, In: PROC. INT. CONF. INTELLIGENT SYSTEMS FOR MOLECULAR BIOLOGY, 8, 2000. p. 307–316.

[43] PATANÈ, G.; RUSSO, M. The enhanced-LBG algorithm, **Neural Netw.**, v. 14, n. 9, p. 1219–1237, 2001.

[44] KRISHNA, K.; MURTY, M. Genetic K-means algorithm, **IEEE Trans. Syst., Man, Cybern. B, Cybern.**, v. 29, n. 3, p. 433–439, jun. 1999.

[45] BATISTA, L. V. **Compressão de Sinais Eletrocardiográficos Baseada na Transformada Cosseno Discreta**. 2002. Tese (Doutorado em Engenharia Elétrica) - UFCG, Campina Grande.

[46] SHANNON, C. E., A mathematical theory of communication. **Bell Systems Technical Journal**, v. 27, p. 379-423, 1948.

[47] HUFFMAN, D.A. A method for the construction of minimum-redundancy codes. **Proc. Inst. Electr. Radio Eng**, v. 40, set. 1952.

[48] PASCO, R. **Source coding algorithms for fast data compression**. 1976. Tese (Doutorado em Engenharia Elétrica), Stanford Univ. Stanford, CA.

[49] WITTEN, I.; NEAL, R.; CLEARY, J. Arithmetic coding for data compression. **CACM**, v.30, n. 6, p. 520-541, jun. 1987.

[50] MOFFAT, A. Implementing the PPM data compression scheme. **IEEE Trans. Commun.** v. 11. nov. 1990.

[51] SKIBINSKI, Pr. PPM with the Extended Alphabet, **Inform. Sci.** v. 176, 2006.

[52] RANDEN, T.; HUSOY, J. H. Texture Segmentaton Using Filters With Optimized Energy Separation, **IEEE Transactions on Image Processing**, v. 8, n. 4, 2009.

[53] RANDEN, T. Banco de imagens. Disponível em: <http://www.ux.uis.no/~tranden/> Acesso em: 15 mar. 2010.