

**UNIVERSIDADE DE SÃO PAULO**  
**ESCOLA DE ENGENHARIA DE SÃO CARLOS**

CARLOS EDUARDO DE OLIVEIRA CHIERICI

**Classificação de texturas com diferentes orientações  
baseada em descritores locais**

São Carlos  
2015



**Carlos Eduardo de Oliveira Chierici**

**Classificação de texturas com diferentes orientações  
baseada em descritores locais**

Dissertação apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para a obtenção do título de Mestre em Ciências, Programa de Engenharia Elétrica.

Área de Concentração: Processamento de Sinais e Instrumentação.

Orientador: Prof. Dr. Adilson Gonzaga.

São Carlos  
2015

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Chierici, Carlos Eduardo de Oliveira  
C532c Classificação de texturas com diferentes orientações baseada em descritores locais / Carlos Eduardo de Oliveira Chierici; orientador Adilson Gonzaga. São Carlos, 2015.

Dissertação (Mestrado) - Programa de Pós-Graduação em Engenharia Elétrica e Área de Concentração em Processamento de Sinais e Instrumentação -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2015.

1. Local Fuzzy Pattern. 2. LFP. 3. análise de texturas. 4. descritor de texturas. 5. classificação de texturas. 6. texturas rotacionadas. I. Título.

## FOLHA DE JULGAMENTO

Candidato: Bacharel **CARLOS EDUARDO DE OLIVEIRA CHIERICI**.

Título da dissertação: "Classificação de texturas com diferentes orientações baseada em descritores locais".

Data da defesa: 25/09/2015

Comissão Julgadora:

Resultado:

Prof. Associado **Adilson Gonzaga (Orientador)**  
(Escola de Engenharia de São Carlos/EESC)

APROVADO

Dr. **Lúcio André de Castro Jorge**  
(Empresa Brasileira de Pesquisa Agropecuária/EMBRAPA)

APROVADO

Prof. Dr. **Celso Aparecido de França**  
(Universidade Federal de São Carlos/UFSCar)

APROVADO

Coordenador do Programa de Pós-Graduação em Engenharia Elétrica:  
Prof. Associado **Luis Fernando Costa Alberto**

Presidente da Comissão de Pós-Graduação:  
Prof. Associado **Paulo César Lima Segantini**



Dedico este trabalho aos meus pais, Rosa e José, por todo amor e educação que recebi, assim como à minha esposa Ivana, pelo seu carinho, compreensão e incentivo ao longo deste trabalho.



## **AGRADECIMENTOS**

Ao Prof. Dr. Adilson Gonzaga, meu orientador, meu agradecimento pela confiança, pelo imenso conhecimento compartilhado e pela permissão de participar de uma equipe tão valorosa, criativa e empenhada como é a equipe do Laboratório de Visão Computacional, o LAVI.

Aos meus colegas de LAVI, excelentes parceiros de estudo, pelo apoio na construção desse trabalho.

Aos meus pais, Rosa e José, que, mesmo diante de sua simplicidade, sempre lutaram pela construção de valores nobres junto ao meu caráter, entre eles a educação. Os agradeço imensamente pelo seu amor e perseverança, combustível que me fez chegar até aqui.

Ao meu irmão, Roberto Chierici, pelo seu exemplo de superação.

À minha querida esposa Ivana, pelo seu amor, apoio e exemplo de superação.

Por último, mas não menos importante, a todos os meus amigos, de infância, trabalho e da academia, por terem contribuído para minha caminhada através de suas boas companhias.



## RESUMO

CHIERICI, C. E. O. **Classificação de texturas com diferentes orientações baseada em descritores locais**, 2015. 179 f. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2015.

Diversas abordagens vêm sendo empregadas para a descrição de texturas, entre elas a teoria dos conjuntos *fuzzy* e lógica *fuzzy*. O *Local Fuzzy Pattern* (LFP) é um descritor de texturas diferente dos demais métodos baseados em sistemas *fuzzy*, por não utilizar regras linguísticas e sim números *fuzzy* que são usados na codificação de um padrão local de escala de cinza. Resultados anteriores indicaram o LFP como um descritor eficaz para a classificação de texturas a partir de amostras rotacionadas ou não. Este trabalho propõe uma análise mais abrangente sobre sua viabilidade para aplicação em cada um desses problemas, além de propor uma modificação a este descritor, adaptando-o para a captura de padrões em multiresolução, o *Sampled LFP*. A avaliação da performance do LFP e do Sampled LFP para o problema de classificação de texturas foi feita através da aplicação de uma série de testes envolvendo amostras de imagens rotacionadas ou não das bases de imagens Outex, álbum de Brodatz e VisTex, onde a sensibilidade obtida por esses descritores foi comparada com um descritor de referência, a variante do *Local Binary Pattern* (LBP) melhor indicada para o teste em execução. Os resultados apontaram o *LFP* como um descritor não indicado para aplicações que trabalhem exclusivamente com amostras não rotacionadas, visto que o LBP mostrou maior eficácia para este tipo de problema. Já para a análise de amostras rotacionadas, o Sampled LFP se mostrou o melhor descritor entre os comparados. Todavia, foi verificado que o Sampled LFP somente supera o LBP para resoluções de análise maiores ou iguais a 32x32 pixels e que o primeiro descritor é mais sensível ao número de amostras usadas em seu treinamento do que o segundo, sendo, portanto, um descritor indicado para o problema de classificação de amostras rotacionadas, onde seja possível trabalhar com imagens a partir de 32x32 pixels e que o número de amostras utilizadas para treinamento seja maximizado.

Palavras-chave: Análise de texturas, descritor de texturas, classificação invariante à rotação, texturas rotacionadas, *Local Fuzzy Pattern*.



## ABSTRACT

CHIERICI, C. E. O. **Classification of texture with different orientations based on local descriptors** 2015. 179 f. Dissertação (Mestrado) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2015.

Several approaches have been employed for describing textures, including the fuzzy sets theory and fuzzy logic. The Local Fuzzy Pattern is a texture descriptor different from other methods based on fuzzy systems, which use linguistic rules to codify a texture. Instead, fuzzy numbers are applied in order to encode a local grayscale pattern. Previous results indicated the LFP as an effective descriptor employed to characterize statically oriented and rotated textures samples. This paper proposes a more comprehensive analysis of its feasibility for use in each of these problems, besides proposing a modification to this descriptor, adapting it to capture patterns in multiresolution, the Sampled LFP. The LFP and Sampled LFP performance evaluation when applied to the problem of texture classification was conducted by applying a series of tests involving images samples, rotated or not, from image databases such as Outex, the Brodatz album and Vistex, where the sensitivity obtained by these descriptors were compared with a reference descriptor, the variant Local Binary Pattern (LBP) best suited to running the test. The results indicated the LFP as a descriptor not suitable for applications who work exclusively with non-rotated samples, since the LBP showed greater efficacy for this problem kind. As for rotated samples analysis, the Sampled LFP proved the best descriptor among those compared. However, it was determined that the Sampled LFP only overcomes the LBP when the analysis resolutions are greater or equal to 32x32 pixels, besides that, the first descriptor is more sensitive to the number of training samples than the latter, therefore, this descriptor is indicated for the problem of rotated samples classification, where it is possible to work with resolution from 32x32 pixels while maximizing the number of samples used for training.

Keywords: texture analysis, texture descriptor, rotated textures, *Local Fuzzy Pattern*



## LISTA DE FIGURAS

Figura 2.1 - Exemplo de classificação e segmentação de texturas .....	29
Figura 2.2 - Imagem em nível de cinza .....	36
Figura 2.3 - (a) Imagem em nível de cinza. (b) <i>Texture Unit</i> .....	36
Figura 2.4 - Computação do código LBP .....	37
Figura 2.5 - Imagem rotacionada com o <i>LBP</i> original .....	38
Figura 2.6 - Conjunto de vizinhos simetricamente circulares .....	39
Figura 2.7 - Operação de <i>bitshifting</i> do <i>LBP ri</i> .....	40
Figura 2.8 - Possíveis padrões do <i>LBP ri</i> .....	40
Figura 2.9 - (a) Operador <i>LBP</i> clássico. (b) Operador <i>MB-LBP</i> 9x9.....	43
Figura 2.10 - (a) Imagem em nível de cinza. (b) <i>FTUs</i> .....	46
Figura 2.11 - Função de pertinência sigmoide .....	49
Figura 2.12 - Cálculo do código <i>LFP</i> .....	50
Figura 3.1 - Ordem de disposição da vizinhança circular.....	55
Figura 3.2 - Vizinhança circular utilizada no <i>Sampled LFP</i> .....	56
Figura 3.3 - Separação de imagens para conjuntos de treino e teste .....	58
Figura 3.4 - Processo de classificação .....	59
Figura 3.5 - Processo de parametrização do <i>Sampled LFP</i> .....	61
Figura 3.6 - Amostras de texturas do banco de imagens Outex.....	62
Figura 3.7 - Pixels utilizados para cálculo do grau de pertinência do pixel central, para uma vizinhança 5x5 pixels: (a) <i>LFP-s</i> ; (b) <i>LFP-q</i> .....	66
Figura 3.8 - Amostras das 16 texturas usadas no experimento I do álbum de Brodatz .....	67
Figura 3.9 - Amostras das 15 texturas usadas no experimento II do álbum de Brodatz .....	69
Figura 3.10 – Exemplos de texturas do VisTex .....	73
Figura 3.11 – Amostras para treino da textura “Bark 0000”, não rotacionada .....	74
Figura 3.12 – Amostras para teste da Textura “Bark 0000”, rotacionada em 10° .....	75
Figura 4.1 - Sensibilidade versus valor de $\beta$ para a suíte Outex_TC_00000 .....	77
Figura 4.2 - Sensibilidade versus valor de $\beta$ para a suíte Outex_TC_00010 .....	78
Figura 4.3 - Sensibilidade versus valor do número de <i>bins</i> para a suíte TC_00000, com $\beta = 1,6$ .....	78
Figura 4.4 - Sensibilidade versus valor do número de <i>bins</i> para a suíte TC_00010, com $\beta = 1,02$ .....	79
Figura 4.5 - Melhor $\beta$ para o experimento III de Brodatz .....	87
Figura 4.6 - Sensibilidade no Outex_TC_00010, para amostras de 64x64 pixels .....	88
Figura 4.7 - Sensibilidade no Outex_TC_00010, para amostras de 128x128 pixels .....	88
Figura 4.8 - Sensibilidade no Outex_TC_00012, para amostras de 64x64 pixels .....	89
Figura 4.9 - Sensibilidade no Outex_TC_00012, para amostras de 128x128 pixels .....	90
Figura 4.10 - Sensibilidade conforme número de amostras de treinamento, para imagens do VisTex, usando amostras de 64x64 pixels e vizinhança 8x1 .....	91
Figura 4.11 - Diferença de sensibilidade dos descritores ( <i>Sampled LFP</i> - <i>LBP riu2</i> ) conforme número de amostras de treinamento para imagens do VisTex, usando amostras 64x64 pixels e vizinhança 8x1 .....	92
Figura 4.12 - Sensibilidade conforme número de amostras de treinamento, para imagens do VisTex, usando amostras de 64x64 pixels e vizinhança 16x2 .....	93



## LISTA DE TABELAS

Tabela 2.1 - Número máximo de padrões gerados pelo <i>LBP</i> .....	41
Tabela 3.1 - Suítes de teste do Outex selecionadas para experimento .....	63
Tabela 3.2 - Configuração do experimento I de Brodatz .....	68
Tabela 3.3 - Configuração do experimento II de Brodatz .....	70
Tabela 3.4 - Configuração do experimento III de Brodatz .....	70
Tabela 3.5 - Configuração do Outex_TC_00010.....	71
Tabela 3.6 - Configuração do Outex_TC_00012.....	72
Tabela 4.1 - Melhor configuração do <i>LFP</i> para cada suíte de teste .....	79
Tabela 4.2 - Comparação entre o <i>LFP-s</i> e o <i>LBP</i> melhor indicado .....	80
Tabela 4.3 - Comparação entre o <i>LFP-s</i> e o <i>LBP riu2</i> .....	81
Tabela 4.4 - Médias dos melhores parâmetros para o <i>LFP-s</i> .....	82
Tabela 4.5 - Sensibilidade do <i>LFP-s</i> parametrizado com $\beta$ e número de <i>bins</i> médios .....	82
Tabela 4.6 - Diferença entre as sensibilidades obtidas pelo <i>LFP-s</i> calculado com parâmetros médios e o <i>LBP riu2</i> .....	83
Tabela 4.7 - Valores de $\beta$ para o experimento I de Brodatz .....	84
Tabela 4.8 - Sensibilidade obtida no experimento I de Brodatz .....	85
Tabela 4.9 - Sensibilidade por ângulo – experimento II de Brodatz .....	85
Tabela 4.10 - Sensibilidade por resolução de amostragem – experimento II de Brodatz.....	86
Tabela 4.11 - Sensibilidade por resolução de amostragem – experimento III de Brodatz.....	86



## LISTA DE SÍMBOLOS E ABREVIATURAS

$A(k, l)$	Pixels da vizinhança
ALBPS	<i>Adaptative Local Binary Pattern With Oriented Standard Deviation Term</i>
$b(q_p, q_c)$	Código do descritor C-LBP
$B$	Número de bins
$c$	Valor limiar
CLBP	Completed Local Binary Pattern
$d_P$	Diferença entre o pixel central e outro pixel da vizinhança
$E_i$	Valor binário ou ternário assumido por um pixel da vizinhança
$f_{g(i,j)}$	Função de pertinência aplicada para cada pixel vizinho
$FE_i$	Uma Fuzzy Texture Unit Box para o pixel $i$
FN	Falsos negativos
FTU	Fuzzy Texture Unit Box
FUNED	Fuzzy Number Edge Detector
$g(i,j)$	Grau de pertinência do pixel central para o pixel vizinho de posição $i, j$
$i$	Número inteiro
$j$	Número inteiro
LBP	Local Binary Pattern
LBP ri	Local Binary Pattern rotarion invariant
LBP riu	Local Binary Pattern rotarion invariant uniform
LBP riu2	Local Binary Pattern rotarion invariant uniform, de até 2 transições de pixels
LFP	Local Fuzzy Pattern
LFP-q	Local Fuzzy Pattern de vizinhança quadrada
LFP-s	Local Fuzzy Pattern Sigmoidal
M	Modelo de textura
$M_b$	Probabilidades do <i>bin b</i> no modelo
$m_P$	Magnitude de $d_P$
MB-LBP	<i>Multi-scale Block Local Binary Patterns</i>
$N_{LBP}$	Código LBP
$N_{TU}$	<i>Texture Unit Number</i> (código da unidade de textura)

NAC	Número de amostras por classe
NATES	Número de amostras para teste
NATRE	Número de amostras para treinamento
NC	Número de classes de texturas
NT	Número de testes disponíveis para a suíte de teste indicada
NTA	Número total de amostras
$P$	Número de pixel que compõem uma vizinhança
$P(k, l)$	Matriz de pesos
$q_c$	Pixel central
$q_p$	Pixel da vizinhança
$R$	Raio do perímetro circular de uma vizinhança
RO	Ângulo de rotação da textura
S	Amostra de imagem
$S_b$	Probabilidades do <i>bin b</i> na amostra
$Sn$	Sensibilidade
$s_P$	Sinal de $d_P$
SVM	Support Vector Machine
TA	Tamanho das amostras
TIL	Tipo de iluminação utilizado na captura de texturas
TU	Unidade de textura
$V_0$	Valor do nível de cinza do pixel central
$V_i$	Valor do nível de cinza de um pixel
VP	Verdadeiros positivos
$w$	Número inteiro
$x$	Número binário
$X(S, M)$	Distância chi-quadrado
$\beta$	Inclinação da curva sigmoide
$\mu_i(V_j)$	Grau de pertinência do pixel $V_j$ para o conjunto fuzzy $i$

## SUMÁRIO

<b>1.</b>	<b>Introdução .....</b>	<b>23</b>
1.1.	Objetivos.....	24
1.2.	Motivação e contribuições .....	25
1.3.	Estrutura do trabalho .....	25
1.4.	Publicações relacionadas .....	26
<b>2.</b>	<b>Fundamentação teórica .....</b>	<b>27</b>
2.1.	Considerações iniciais .....	27
2.2.	Descrição de texturas .....	30
2.2.1.	Abordagem estatística .....	31
2.2.2.	Abordagem estrutural.....	32
2.2.3.	Abordagem espectral .....	33
2.2.4.	Abordagem baseada em modelos .....	34
2.3.	O Local Binary Pattern (LBP) .....	35
2.3.1.	Descritores baseados no LBP.....	42
2.4.	Descritores fuzzy .....	45
2.4.1.	Local Fuzzy Pattern .....	47
2.5.	Considerações finais .....	51
<b>3.</b>	<b>Metodologia .....</b>	<b>53</b>
3.1.	Considerações iniciais .....	53
3.2.	Sampled Local Fuzzy Pattern.....	54
3.3.	O processo de classificação .....	57
3.3.1.	Parametrização do Sampled LFP .....	60
3.4.	Avaliação do LFP para classificação de texturas .....	61
3.5.	Classificação de amostras de texturas rotacionadas e e em multiresolução..	65
3.5.1.	Álbum de Brodatz.....	66
3.5.2.	Outex .....	71
3.5.3.	VisTex .....	72
3.6.	Considerações finais .....	75
<b>4.</b>	<b>Resultados .....</b>	<b>77</b>
4.1.	Avaliação do LFP para classificação de texturas .....	77
4.1.1.	Considerações .....	83
4.2.	Classificação de texturas invariantes à rotação e multiresolução .....	84
4.2.1.	Álbum de Brodatz.....	84
4.2.2.	Outex .....	87
4.2.3.	VisTex .....	90
4.3.	Considerações finais .....	94
<b>5.</b>	<b>Conclusões .....</b>	<b>96</b>
5.1.	Sugestões para trabalhos futuros .....	97
	<b>Referências bibliográficas.....</b>	<b>99</b>
	<b>Apêndice A – Código fonte comum para teste e treino com o álbum de Brodatz, Outex e VisTex.....</b>	<b>109</b>
	<b>Apêndice B – Código fonte exclusivo para treino com o álbum de Brodatz e Outex 125</b>	

<b>Apêndice C – Código fonte exclusivo para teste com o álbum de Brodatz e Outex 139</b>	
<b>Apêndice D – Código fonte exclusivo para treino com o VisTex .....</b>	<b>151</b>
<b>Apêndice E – Código fonte exclusivo para teste com o VisTex .....</b>	<b>163</b>
<b>Apêndice F – Código fonte exclusivo para treino e teste com o VisTex .....</b>	<b>175</b>
<b>Apêndice G – Parâmetros do Sampled LFP sintonizados para os experimentos com o álbum de Brodatz e Outex .....</b>	<b>177</b>
<b>Apêndice H – Parâmetros sintonizados e sensibilidade do Sampled LFP para o experimentos com o VisTex .....</b>	<b>178</b>
<b>Apêndice I –Sensibilidade do LBP para o experimento com o VisTex .....</b>	<b>180</b>

## 1. INTRODUÇÃO

Entre todos os sentidos humanos, a visão é um dos mais importantes por prover a um espectador farta informação sobre o ambiente ao seu redor. Essa fonte de informações também é de interesse das máquinas, sendo que a ciência e a tecnologia envolvidas na construção de sistemas artificiais, destinadas a incorporar a percepção de um ambiente a uma máquina, por meio do processamento de informações visuais, constitui o campo de estudo da visão computacional.

Conforme apresentado por Szelisk (2011), sistemas de visão computacional são utilizados hoje em dia em uma variedade de aplicações, tais como:

- Reconhecimento ótico de caracteres: tal como leitura de escritos feitos à mão e reconhecimento automático de placas de automóveis;
- Inspeção automática: pela utilização de câmeras especiais, a qualidade de artefatos ou a busca por defeitos é realizada;
- Construção de modelos 3D (fotogrametria): a construção de modelos 3D é automatizada a partir de fotografias aéreas, tais como as utilizadas pelo Google Maps;
- Análise de imagens médicas: onde as informações extraídas das imagens permitem realizar diagnósticos sobre os pacientes;
- Segurança automotiva: detectando obstáculos não esperados, como pedestres na rua;
- Captura de movimentos: muito utilizado nos filmes cheios de efeitos especiais produzidos pela indústria de Hollywood, o movimento de atores é capturado por várias câmeras e posteriormente aplicado a um avatar eletrônico;
- Vigilância: como monitoramento de intrusos ou análise de tráfego rodoviário;
- Biometria: onde a autenticação de um indivíduo é feita a partir da análise de características físicas ou comportamentais do mesmo, como uma digital ou seu modo de caminhar.

Entre as diversas técnicas disponíveis para a construção dessas aplicações, a análise do padrão de texturas tem papel de destaque na literatura, sendo considerada uma técnica de baixo nível, fundamental no campo de visão computacional e de reconhecimento de padrões (SHADKAM et al., 2012). Porém, a maior parte das abordagens para classificação de texturas assume, tanto implicitamente como explicitamente, que as amostras desconhecidas a serem classificadas são idênticas ao conjunto de amostras utilizados para treinar o classificador, com relação à escala espacial, orientação e propriedades da escala de cinza (OJALA; PEITIKAINEN; MÄENPÄÄ, 2002). Todavia, texturas do mundo real podem ocorrer em escalas espaciais e rotações arbitrárias, além de sofrer com as variações das condições de iluminação, proporcionadas pelo ambiente natural.

Neste contexto, diversos descritores foram desenvolvidos nos últimos anos, entre eles o *Local Fuzzy Pattern - LFP* (VIEIRA et al., 2012a), o qual consiste em um descritor que emprega números fuzzy para lidar com as incertezas dos níveis de cinza dos pixels, sendo desenvolvido pela equipe do Laboratório de Visão Computacional (LAVI), do Departamento de Engenharia Elétrica e de Computação/EESC, da Universidade de São Paulo.

### **1.1. Objetivos**

Este trabalho tem como objetivo geral analisar a viabilidade do descritor *Local Fuzzy Pattern* aplicado ao problema de classificação de texturas. As seguintes perguntas em relação a este descritor são propostas e respondidas:

- Pode ser aplicado com eficácia em problemas de classificação de amostras de texturas rotacionadas ou não?
- Apresenta resultados melhores que o descritor *Local Binary Pattern - LBP*?
- O número de amostras por classe de textura utilizadas no processo de classificação, bem como seu tamanho, são fatores primordiais para a sensibilidade obtida pelo descritor?
- A metodologia empregada pelo *LFP* o habilita para emprego em aplicações do mundo real?

Em específico, os seguintes objetivos são estabelecidos:

- Avaliar se o *Local Fuzzy Pattern (LFP)* é um descritor invariante à rotação;
- Propor um descritor de texturas, baseado na técnica de números *fuzzy* utilizada pelo *LFP*, apto à tarefa de classificação de padrões em multiresolução;
- Validar o desempenho da metodologia realizando a análise de imagens contidas em bases de texturas amplamente usadas pela comunidade acadêmica;
- Determinar a melhor configuração do *LFP* para sua aplicação em problemas de classificação de texturas.

## **1.2. Motivação e contribuições**

A redução dos preços de componentes eletrônicos, bem como o aumento do poder de processamento de sistemas computacionais, favorece cada vez mais a popularização de aplicações de visão computacional.

Uma vez que algumas dessas aplicações são baseadas na análise de texturas e que as mesmas ocorrem em escalas e rotações arbitrárias numa cena natural, é desejado que um descritor seja robusto a essas variações. O *LFP* apresentou resultados promissores neste sentido, conforme relatado por Vieira et al. (2012b) e Vieira et al. (2012c), contudo, por ser uma metodologia recentemente desenvolvida, ainda não está claro quais são suas limitações, nem se há alguma forma de melhoria de sua capacidade de discriminação.

Assim, este trabalho é motivado pela necessidade de investigação sobre os limites do *Local Fuzzy Pattern*, colaborando para sua evolução e direcionamento para futuras aplicações.

## **1.3. Estrutura do trabalho**

Este trabalho está dividido em 5 capítulos, estruturado conforme a seguir:

O primeiro capítulo apresenta uma introdução sobre os aspectos gerais do assunto tratado neste trabalho, bem como os objetivos gerais, motivação e sua estruturação.

O segundo capítulo apresenta uma revisão mais detalhada das várias definições para textura, uma revisão bibliográfica dos principais métodos utilizados para caracterização e classificação de texturas e, devido a sua importância na literatura, uma seção específica é destinada ao descritor *Local Binary Pattern* (OJALA; PIETIKÄINEN; HARWOOD, 1996) e suas derivações.

O terceiro capítulo apresenta a metodologia desenvolvida em duas partes, a primeira descreve a forma de avaliação do desempenho do descritor *LFP* para a classificação de amostras de texturas. A segunda parte descreve a metodologia destinada especificamente à classificação de texturas com rotação e como os experimentos foram construídos.

No quarto capítulo os resultados obtidos são apresentados e discutidos.

E por fim, no quinto capítulo são apresentadas as conclusões finais do trabalho.

#### **1.4. Publicações relacionadas**

CHIERICI, C. E. O.; VIEIRA, R.T.; FERRAZ, C. T.; TRAVAINI, J. N.; GONZAGA, A. A new approach for analyzing rotated textures. In: Anais do IX Workshop de Visão Computacional, Rio de Janeiro, 2013.

## 2. FUNDAMENTAÇÃO TEÓRICA

### 2.1. Considerações iniciais

Conforme Haralick, Shanmugam e Dinstein (1973), “textura é uma propriedade inata a virtualmente todas as superfícies. Ela contém informações relevantes sobre o arranjo estrutural das superfícies e sua relação com o ambiente ao redor”. Ainda que textura seja uma característica facilmente reconhecida, ela é de difícil definição (TUCERYAN, 1993). Esta dificuldade levou vários pesquisadores a fornecerem sua própria definição para textura.

Para Hawkins (1969):

[...] a noção de textura parece depender de três ingredientes: (i) uma ordem local é repetida ao longo de uma região que é grande em comparação ao tamanho da ordem local, (ii) a ordem local consiste no arranjo não aleatório de peças elementares, e (iii) as partes são entidades grosseiramente uniformes com aproximadamente as mesmas dimensões em qualquer lugar dentro da região texturizada.

Entre algumas outras definições catalogadas por Coggins (1982), destaca-se a de Richards e Polit (1974), onde os mesmos a definem como:

[..] o atributo de um campo sem componentes que pareçam enumeráveis. A relação de fase entre os componentes não é aparente. Nem o campo deve conter um gradiente óbvio. O objetivo desta definição é direcionar a atenção do observador para as propriedades globais do display, por exemplo, aspereza geral, o relevo ou a suavidade.

Já Tamura, Mori e Yamawaki (1978) consideraram textura como o componente que constitui uma região macroscópica, sendo que sua estrutura é atribuída a padrões repetitivos nos quais elementos ou primitivas são organizados conforme determinada regra. Sklansky (1978) afirma que uma região em uma imagem tem uma textura constante somente se um conjunto de estatísticas ou propriedades locais se mantém constante, com pouca variação ou for aproximadamente periódica. Uma definição mais abrangente foi feita por Haralick (1979):

[...] a imagem de textura que consideramos é não figurativa e celular.... Uma imagem de textura é descrita pelo número e tipos das suas primitivas e da organização espacial ou leiaute de suas primitivas.... Uma característica fundamental da textura: não pode ser analisada sem um quadro de referência da primitiva tonal seja explícita ou implícita. Para toda superfície lisa em tons de cinza, existe uma escala tal que quando a superfície é examinada, ela não tem textura. Então, quando a resolução cresce, ela ganha uma textura suave e então uma textura mais grosseira.

Eles ainda propuseram que imagens de texturas poderiam ser avaliadas com base em propriedades como coeficiente de uniformidade, densidade, aspereza, regularidade, intensidade, dentre outras características. Neste mesmo trabalho, os autores caracterizaram textura como um conceito bidimensional, onde uma dimensão contém as propriedades primitivas da tonalidade e a outra corresponde aos relacionamentos espaciais entre elas. Eles indicaram que os conceitos de tonalidade e textura não são independentes, tal que em algumas imagens a tonalidade é dominante e, em outras, a textura é dominante.

Zucker e Kant (1981) também reconheceram textura como um conceito paradoxal, onde seu uso é comum em pré-processamento da informação visual, especialmente para propósitos de classificação, porém, sem que exista uma definição de textura aceita por todos. Desse modo, eles indicaram que a solução deste paradoxo dependeria de um modelo mais rico, representado por vários níveis de abstração.

Em 1990, o IEEE Standard (1990) proveu uma definição, ainda que simplória, indicando que textura é um atributo que representa o arranjo espacial dos níveis de cinza dos pixels em uma região de imagem.

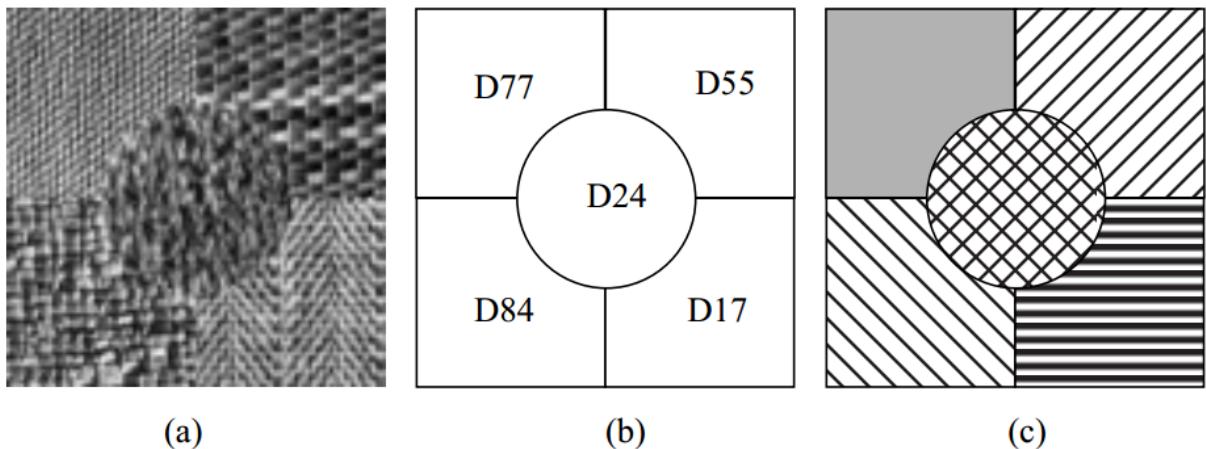
A falta de uma definição clara torna o problema de análise de texturas mais difícil do ponto de vista matemático, visto que os vários métodos de análise não podem ser comprovados como certos ou errados, fazendo com que sua avaliação seja conduzida de maneira empírica (MÄENPÄÄ, 2003); ainda assim, independente da definição utilizada, é importante ressaltar que uma região de textura fornece interpretações diferentes conforme a distância e o grau de atenção visual para seu observador (HARALICK, 1979; CHAUDHURI; SARKAR, 1995).

Não obstante sua imprecisa descrição, a análise de suas propriedades é considerada uma técnica de baixo nível, fundamental no campo de visão

computacional e de reconhecimento de padrões (SHADKAM et al., 2012), sendo utilizada em processos que fundamentam as aplicações finais, utilizadas por usuários.

Tuceryan (1993) elenca 3 aplicações básicas, sendo a primeira a classificação de texturas.

Por exemplo, na Figura 2.1 (a), é possível identificar cinco texturas diferentes. Considerando suas características, o sistema de classificação deve produzir um mapa de classificação da imagem de entrada, onde cada região de textura uniforme é identificada com a classe a qual esta textura pertence Figura 2.1 (b).



**Figura 2.1 - Exemplo de classificação e segmentação de texturas**  
Fonte: Tuceryan e Jain (1993, p. 2)

A segunda aplicação é a de segmentação de texturas, ou seja, a definição dos limites entre diferentes texturas, mesmo quando não é possível classificá-las. O objetivo da segmentação de texturas é obter o mapa de contorno como mostrado na Figura 2.1 ©. A terceira aplicação refere-se à síntese de texturas, que é fundamental em computação gráfica, onde o objetivo é renderizar superfícies de objetos tão realistas quanto possível.

Outras aplicações foram apontadas em outros trabalhos, como por exemplo, tarefas de recuperação de imagens por conteúdo (SMEUDERS et al., 2000; LIU et al., 2007) e construção de formas a partir de textura (LOBAY; FORSYTH, 2006).

Em geral, as técnicas para classificação de texturas funcionam muito bem enquanto as amostras de texturas usadas para treino e teste tiverem orientações idênticas ou similares. Porém, as rotações de imagens de texturas capturadas do mundo real irão variar arbitrariamente, afetando a performance do método (GUO;

ZHANG; ZHANG, 2010a). Tal restrição leva à necessidade de desenvolvimento de métodos mais robustos com a variação de orientação ou rotação.

Como indicado previamente, este trabalho se concentrará no problema de caracterização e classificação de texturas.

## **2.2. Descrição de texturas**

Devido à subjetividade inerente à definição de textura, vários descritores-foram desenvolvidos nos últimos anos, cada um deles seguindo uma abordagem conforme a definição assumida.

Tradicionalmente, essas técnicas são enquadradas em três principais categorias (GONZALEZ; WOODS, 2008): estatística, estrutural e espectral. Porém, diferentes autores fazem outra categorização, como Nixon e Aguado (2008), que categorizam os métodos de análise de textura em estatística, espectral e abordagem combinatória. Outra classificação é dada por Xie (2008), que considera as abordagens estatística, estrutural, baseada em filtros e baseada em modelos. Shinivasan e Shobha (2008) consideram a análise de textura como uma classe de procedimentos e modelos matemáticos que caracterizam variações espaciais dentro de uma imagem e, desta maneira, apontam somente duas categorias: estatística e sintática. Zhang e Tan (2002) dividem os métodos em estatísticos, baseados em modelos e estruturais; enquanto que Materka e Strzelecki (1998) adotam as categorias: estrutural, estatística, baseada em modelos e em transformadas. Já Tuceryan (1993), propõe quatro abordagens: métodos estatísticos, geométricos, baseado em modelos e em processamento de sinais. Essa diversidade de taxonomias para os métodos de análise de textura reflete as dificuldades em se estabelecer uma definição de textura mais objetiva e consensual.

Laws (1980) identificou as seguintes propriedades como tendo importante papel na descrição de texturas: uniformidade, densidade, aspereza, rugosidade, regularidade, linearidade, direcionamento, frequência e fase, sendo que algumas dessas qualidades não são independentes, como por exemplo, a frequência não é independente da densidade. O fato da textura possuir tão variadas dimensões é uma

das principais razões para a inexistência de um método único de representação, adequado para análise de todas as variedades possíveis de texturas.

Nas seções a seguir, as abordagens estatística, estrutural, espectral e baseada em modelos serão individualmente descritas.

### **2.2.1. Abordagem estatística**

Nos métodos estatísticos, a textura é descrita por uma coleção de estatísticas sobre a distribuição e relação entre os níveis de cinza de uma imagem.

Conforme Zhang e Tan (2002), muitos desses métodos são baseados nos trabalhos de Julesz (1975), sobre como o sistema de visão humano utiliza recursos estatísticos para discriminação de textura dependendo do número de pixels que definem a característica local. Esses métodos podem ser classificados como estatística de primeira ordem (um pixel), segunda ordem (dois pixels) e alta ordem (três ou mais pixels) (OJALA; PIETIKÄINEN, 2013). A diferença básica é que a estatística de primeira ordem estima propriedades (por exemplo, média e variância) de pixels individuais, ignorando a interações entre pixels, enquanto que nas estatísticas de segunda e alta ordem, estimam propriedades de dois ou mais valores de pixels ocorrendo em locais específicos relativos ao pixel analisado (analisa-se o posicionamento espacial de cada pixel em relação a sua vizinhança). Weszka, Dyer e Resenfeld (1976) demonstraram que métodos baseados em estatística de segunda ordem obtém maior taxa de discriminação que métodos estruturais e espetrais.

Um dos métodos estatísticos mais conhecidos proposto por Haralick (1979) é o da aplicação da matriz de co-ocorrência dos níveis de cinza. Este método é classificado como uma medida estatística de segunda ordem e fornece a probabilidade de ocorrência de níveis de cinza entre dois pixels vizinhos, um chamado de pixel referência e o outro de pixel vizinho, separados espacialmente por um vetor de distância fixo  $d = (\Delta_x, \Delta_y)$ . No mesmo trabalho, Haralick ainda sugeriu 14 medidas que poderiam ser extraídas dessas matrizes: variância, segundo momento angular, energia, contraste, correlação, homogeneidade, momento inverso da diferença, soma da média, soma da variância, soma da entropia, diferença da variância, diferença da entropia, medida de informação da correlação e coeficiente de correlação máxima.

Haralick (1979) ainda apontou oito abordagens estatísticas para a medição e caracterização de textura: funções de correlação, transformadas ópticas, transformadas digitais, bordas texturais, elementos estruturantes, probabilidade de co-ocorrência espacial de níveis de cinza, comprimento de corrida de nível de cinza (*run length*) e modelos auto regressivos.

### **2.2.2. Abordagem estrutural**

Este enfoque introduz o conceito de *texel* ou *texton*, um termo que representa uma unidade primitiva de discriminação de textura (JULESZ, 1981). Nesta abordagem, a estrutura espacial de uma textura é enfatizada (MÄENPÄÄ, 2003), sendo descrita a partir de um vocabulário de *textons* e de uma descrição da relação entre esses elementos, que pode ser expressa em termos de adjacência, distância mais próxima ou periodicidade. Os próprios *texels* podem ser definidos por seu nível de cinza, forma ou homogeneidade de alguma propriedade local (LEE, 2004).

De acordo com Faugeras e Pratt (1980), a abordagem determinística considera textura um padrão local básico que é repetido periodicamente ou quase periodicamente sobre uma região. Cross e Jain (1983) assumem textura como uma composição de primitivas de determinada forma e tamanho variado, que seguem determinada regra de posicionamento com relação a cada um desses elementos.

Modelos de textura estruturais funcionam bem com macro texturas de construções bem definidas (MÄENPÄÄ, 2003; POPESCU; DOBRESCU; ANGELESCU, 2008). Materka (1998) afirma que a vantagem desta abordagem é que ela fornece uma boa descrição simbólica da imagem, ainda que ressalte que essa característica é mais útil em tarefas de síntese do que em análise de textura. Huang, Chang e Zhang (2003) indicam que a capacidade para reconhecimento de texturas randômicas não é alta. Texturas naturais usualmente contêm tanto componentes estruturais como estatísticos, portanto, trabalhar com um único tipo de característica não seja suficiente para descrevê-las completamente.

### 2.2.3. Abordagem espectral

Os métodos de análise baseados no domínio espacial referem-se ao próprio plano da imagem, sendo, portanto baseados na análise direta dos pixels, enquanto que a abordagem espectral é baseada nas propriedades de espectros de frequência, comumente obtidos pela aplicação de transformadas, como a de Fourier, Gabor e *wavelets*.

Na transformada de Fourier, imagens de domínio espacial são transformadas para o domínio da frequência, decompondo uma imagem em suas componentes seno e cosseno. A saída da transformada representa a imagem no domínio da frequência, onde cada ponto significa cada frequência contida no domínio espacial da imagem (a imagem de entrada). Dessa maneira, o sistema de coordenadas do plano do domínio de frequência proporciona uma interpretação relacionada com as características de textura, como frequência e tamanho. Assim, ao se analisar o plano das frequências, a identificação dos picos, com sua proeminência e localização, fornece a direção principal das texturas e o período espacial fundamental dos padrões, respectivamente. Padrões não periódicos na imagem podem ser descritos com a aplicação de filtros para eliminação dos elementos periódicos e posterior aplicação de técnicas estatísticas. O espectro de uma imagem real é simétrico em torno da origem, de maneira que apenas metade do plano das frequências necessita ser considerado. Portanto, para propósito de análise, cada padrão periódico é associado a apenas um pico no espectro, ao invés de dois.

Gabor (1946) demonstrou que a capacidade de caracterizar um sinal simultaneamente nos domínios temporal (ou espacial) e das frequências, é limitada pela relação de incerteza conjunta, também conhecida como “princípio de Heisenberg”. Gabor então determinou uma família de funções que atingem este limite inferior de incerteza conjunta, sendo essencialmente composta por uma função que descreve uma onda senoidal com frequência  $\omega$  modulada por um envelope Gaussiano com duração (ou extensão)  $\sigma$  (Equação 2.1):

$$f(t) = \exp \left[ -\frac{1}{2} \left( \frac{t}{\sigma} \right)^2 + i\omega t \right] \quad (2.1)$$

Desse modo, enquanto as análises baseadas no espectro de Fourier não revelam as distribuições locais que caracterizam texturas (GONZALEZ; WOODS, 2008), o filtro de Gabor representa tanto o tempo quanto a frequência de um sinal, indicando respectivamente local e característica da textura, permitindo a manipulação de diversos parâmetros, como frequência, orientação, excentricidade e simetria. Utilizando-se estas combinações são formados os bancos de filtros de Gabor (JU; CHANG; HUNG, 2004).

Comparado com a transformada de Gabor, as transformadas *wavelets* (MALLAT, 1989; DO; VETTERLI, 2002; JI et al., 2013) apresentam várias vantagens, tais como permitir a variação da resolução espacial, habilitando a representação de texturas numa escala mais adequada. Existe uma ampla gama de escolhas para a função de *wavelet*, assim é possível escolher a mais adequada para uma aplicação específica, além de apresentar melhor precisão do que métodos baseados no domínio espacial, visto que elas extraem melhor as características de uma imagem, devido à informação no domínio da frequência ser geralmente mais estável do que o domínio espacial (TOU; TAY; LAU, 2009).

#### **2.2.4. Abordagem baseada em modelos**

Métodos de análise de textura baseados em modelos são fundamentados na construção de um modelo de imagem, geralmente usando modelos estocásticos e geradores, não apenas para descrever uma textura, mas também para sintetizá-las. Os parâmetros desses modelos são usados para caracterizar as principais qualidades da textura (TUCERYAN; JAIN, 1993). O principal problema desses métodos é como estimar esses coeficientes e como escolher o modelo correto adequado para a textura selecionada (ZHANG; TAN, 2002; MIRMEHDI; XIE; SURI, 2008).

Entre os vários métodos desta abordagem, estão os campos aleatórios de Markov, fractais, modelos auto regressivos e campos aleatórios de Gibbs (SIVAKUMAR; GOUTSIAS, 1999).

Campos aleatórios de Markov, ou *Markov Random Fields* (*MRFs*), são populares por seu uso na modelagem de imagens. Eles são capazes de capturar a informação contextual local (espacial) de uma imagem, supondo que a intensidade de cada pixel na imagem depende somente da intensidade dos pixels vizinhos. Modelos

MRFs são aplicados a várias áreas de processamento de imagem, tais como a síntese e classificação texturas, além de segmentação e restauração de imagens, entre outras.

A geometria fractal foi proposta por Mandelbrot (1983), o primeiro a notar a sua existência no mundo natural. Esse modelo é baseado na característica de que muitas superfícies têm uma qualidade estatística de rugosidade e de auto semelhança em diferentes escalas, muito útil na modelagem de texturas naturais. Duas de suas características mais populares são a dimensão fractal e a lacunaridade. No entanto, este modelo geralmente não é considerado adequado para representar estruturas locais de uma imagem.

Esses descritores podem ainda ser combinados com outros, de modo a oferecer melhor poder de descrição, como demonstraram Li, Prasad e Fowler (2014) em seu descritor *GMM-MRF*, que combinou um descritor baseado em modelo de misturas de gaussianas (*GMM*) com outro baseado em campos aleatórios de Markov.

### **2.3. O Local Binary Pattern (LBP)**

Loris, Alessandra e Sheryl (2012) afirmam que o *Local Binary Pattern* é um dos descritores mais empregados para classificação de texturas. Entre os motivos para sua ampla adoção estão sua resistência a mudanças de condições de iluminação, baixa complexidade computacional e habilidade para descrever detalhes finos. Foi introduzido por Ojala, Pietikäinen e Harwood (1996) e nota-se forte influência da metodologia *Texture Unit*, apresentada por He e Wang (1990).

Na abordagem da *Texture Unit*, textura é considerada como um conjunto de pequenas unidades chamadas *Texture Unit (TU)*, que caracterizam a informação de textura local para um pixel e sua vizinhança. A assinatura de uma textura é configurada pelo seu espectro de textura, ou seja, pela distribuição de frequência de todas as unidades de textura encontradas na imagem. Nessa metodologia, a informação de textura é extraída de cada pixel e de sua vizinhança 3x3, sendo que o padrão dessa imagem é denotado por um conjunto de nove elementos, dado como  $V = \{V_0, V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8\}$ , sendo que  $V_0$  representa o valor do nível de cinza do pixel central e  $V_i$  ( $1 \leq i \leq 8$ ) os valores dos pixels vizinhos (Figura 2.2).

$V_1$	$V_2$	$V_3$
$V_8$	$V_0$	$V_4$
$V_7$	$V_6$	$V_5$

**Figura 2.2 - Imagem em nível de cinza**

A  $TU$  é extraída dessa vizinhança e é definida como  $TU = \{E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8\}$ , sendo  $E_i$  dado pela Equação 2.2:

$$E_i = \begin{cases} 0, & \text{se } V_i < V_0 \\ 1, & \text{se } V_i = V_0 \\ 2, & \text{se } V_i > V_0 \end{cases} \quad (2.2)$$

Como exemplo, a Figura 2.3 (a) apresenta uma amostra de imagem em nível de cinza e a Figura 2.3 (b) apresenta sua *Texture Unit*.

113	110	108
114	113	120
160	150	130

(a)

1	0	0
2		2
2	2	2

(b)

**Figura 2.3 - (a) Imagem em nível de cinza. (b) Texture Unit.**

Visto que uma  $TU$  é composta por 8 elementos e cada um deles pode assumir 3 possíveis valores (0, 1 ou 2), o número total de possíveis *Texture Units* é  $3^8 = 6561$ . Essas  $TUs$  podem ser rotuladas pela multiplicação dos valores da  $TU$  pela matriz de pesos, com seus resultados sendo somados. A esse código é dado o nome de *Texture Unit Number* ( $N_{TU}$ ) e seu cálculo é feito pela Equação 2.3. Como exemplo, para a  $TU$  apresentada na Figura 2.3 (b), o valor de  $N_{TU}$  é de 6.528.

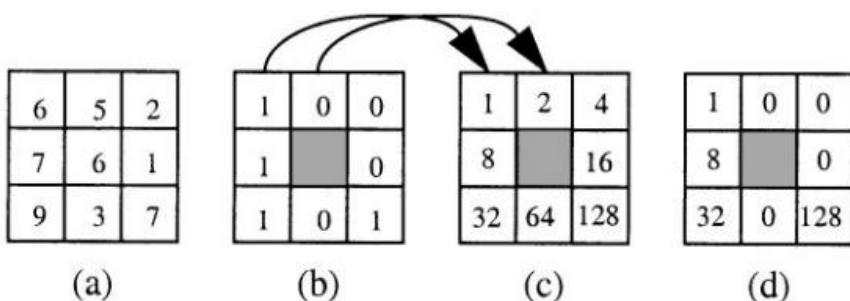
$$N_{TU} = \sum_{i=1}^{i=8} E_i (3^{i-1}) \quad (2.3)$$

O espectro de textura consiste em transformar uma imagem em uma matriz de *TUs* e avaliar a frequência de ocorrência dos vários *TUs* encontrados, comumente executado pela composição de um histograma dos *TUs*.

Tomando por base a metodologia *Texture Unit*, Ojala, Pietikäinen e Harwood (1996) apresentaram o *Local Binary Pattern (LBP)*, um poderoso descritor de texturas, capaz de descrever a relação entre um pixel e sua vizinhança imediata. Foi concebido a partir da ideia de que textura pode ser descrita por duas medidas complementares, micro padrões e contraste.

Uma de suas características mais importantes é sua invariância com relação às variações monotônicas de nível de cinza causadas por, por exemplo, variações de iluminação. Este método pode ser visto como uma abordagem unificadora dos métodos estatísticos e estruturais para descrição de textura (PIETIKÄINEN et al., 2011).

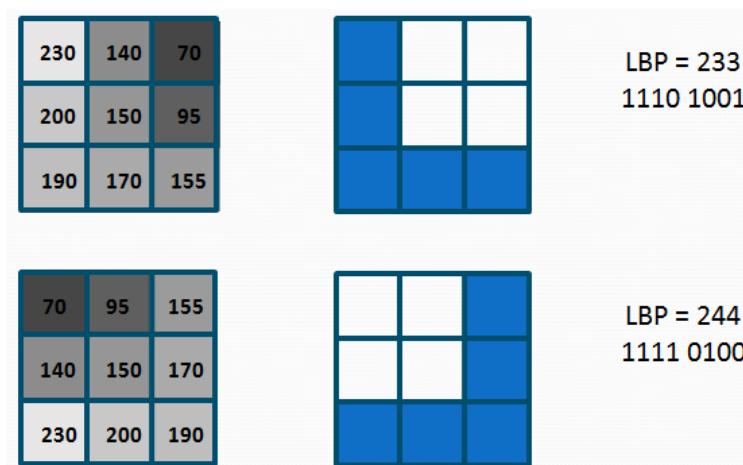
A primeira versão do operador trabalha em uma janela 3x3 pixels, usando o valor do nível de cinza do pixel central como limiar. Para cada pixel da imagem, um código *LBP* é gerado pela multiplicação dos valores de nível de cinza dos pixels da vizinhança (valores já limiarizados) por uma matriz de pesos sendo seus resultados somados. Visto que nesta versão do *LBP*, a vizinhança do pixel central consiste de 8 pixels, o número de códigos *LBP* possíveis é igual a  $2^8 = 256$ . A Figura 2.4 exemplifica esse processo, onde uma vizinhança 3x3 pixels (Figura 2.4a) é limiarizada pelo valor do pixel central. Os valores dos pixels na vizinhança limiarizada (Figura 2.4b) são multiplicados pela matriz de pesos (Figura 2.4c), resultando numa matriz de valores (Figura 2.4d). Os resultados dessa matriz de valores são então somados, formando o número *LBP* de valor igual a 169.



**Figura 2.4 - Computação do código LBP**

Fonte: Ojala e Pietikäinen (1999, p. 478)

Apesar de seu poder para descrição de texturas, o método original apresenta o inconveniente de não ser robusto a variações de rotação, devido ao uso da matriz de pesos baseada em potência de dois (PIETIKÄINEN; OJALA; XU, 2000). A Figura 2.5 exemplifica o problema, exibindo uma imagem de 3x3 pixels sendo rotacionada em 90º à esquerda. Apesar de seu micro-padrão se manter, o código *LBP* gerado é alterado conforme o ângulo de rotação.

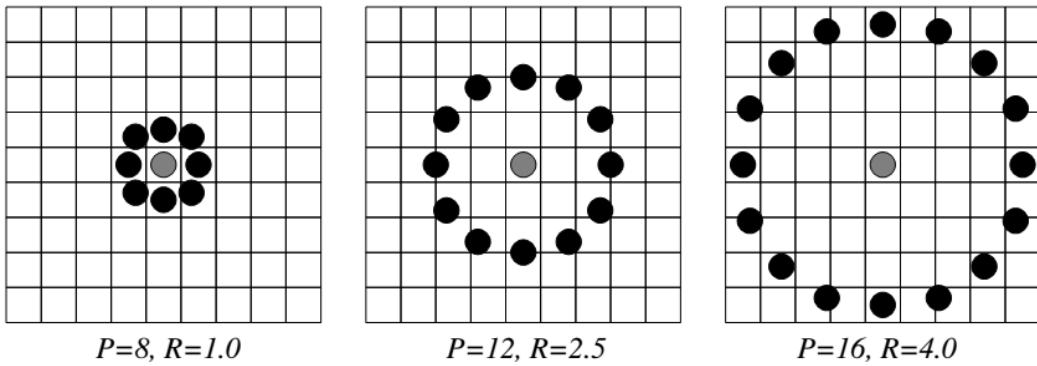


**Figura 2.5 - Imagem rotacionada com o *LBP* original**

Este problema foi contornado posteriormente por Ojala, Pietikäinen e Mäenpää (2002), quando introduziram duas variações ao método *LBP* original: o *LBP ri (rotation invariant)* e o *LBP riu (rotation invariant uniform)*.

Tanto o *LBP ri* quanto o *riu* são muito similares ao *LBP* original, porém, com duas diferenças básicas. Primeiro, a vizinhança do pixel central é indexada circularmente e depois os pixels da vizinhança em posição diagonal são obtidos por meio de interpolação. O pixel central ainda é utilizado como limiar do nível de cinza e os pixels da vizinhança são utilizados da mesma forma para a caracterização da unidade de textura.

Uma característica desses operadores é que tanto o número de vizinhos quanto o valor do raio da vizinhança podem ser definidos conforme o problema a que se destinam. A Figura 2.6 ilustra a composição da vizinhança para vários valores de *P* e *R*, número de pixels vizinhos e raio, respectivamente.



**Figura 2.6 - Conjunto de vizinhos simetricamente circulares**

Fonte: Mäenpää (2003, p. 34)

Para ambos descritores, o processo se inicia com a determinação do código *LBP*, que pode assumir 256 valores diferentes, considerando  $P = 8$  e  $R = 1$ . Considerando  $q_c$  o pixel central,  $q_p$  um pixel da vizinhança, esse código é dado pela Equação 2.4 :

$$LBP(P, R) = \sum_{p=0}^{P-1} s(q_p - q_c) 2^P \quad (2.4)$$

em que:

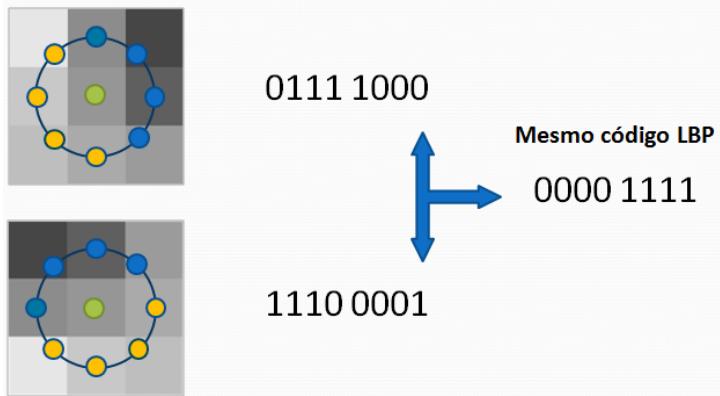
$$s(x) = \begin{cases} 1, & \text{se } x \geq 0 \\ 0, & \text{se } x < 0 \end{cases}$$

Em seguida uma operação de deslocamento de bits (*bit shifting*) circular é operada no código *LBP*. O novo código *LBP* será o menor valor encontrado entre todos os valores iterados e o código *LBP* invariante à rotação é definido pela Equação 2.5:

$$LBP_{P,R}^{ri} = \min\{ROR(LBP_{P,R}, i) \mid i = 0, 1, \dots, P - 1\} \quad (2.5)$$

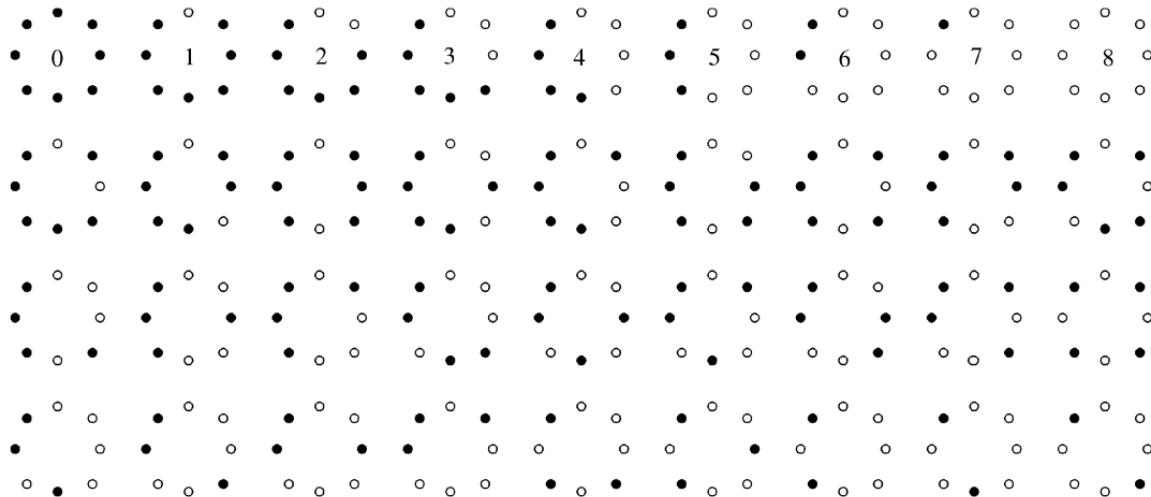
sendo que  $ROR(x, i)$  executa  $i$  vezes a operação de *bit shifting* circular à direita no número binário  $x$ .

A Figura 2.7 apresenta o mesmo caso da Figura 2.5, mas com o código *LBP* gerado a partir da técnica do *bit shifting*.



**Figura 2.7 - Operação de *bitshifting* do *LBP ri***

A este operador é dado o nome de *LBP ri* e, para  $P = 8$ , 36 valores podem ser gerados, representando padrões únicos e invariantes à rotação. A Figura 2.8 ilustra os 36 padrões possíveis do *LBP ri* (invariantes à rotação), para o caso de o número de pixels vizinhos ser igual a oito ( $P = 8$ ). Círculos negros e brancos correspondem aos valores dos bits zero e um na saída de 8 bits do descritor.



**Figura 2.8 - Possíveis padrões do *LBP ri***

Fonte: Ojala e Pietikäinen (2002, p. 974)

O *LBP ri* quantifica a ocorrência estatística desses padrões individuais, que correspondem a certas micro características de imagem, como pontos de brilho, pontos negros, áreas planas e bordas.

Porém, de acordo com Ojala e Pietikäinen (2002), a prática demonstrou que o *LBP ri* não proveu uma discriminação muito boa devido à frequência de ocorrência

dos 36 padrões individuais variar muito, além da quantização "crua" do espaço angular em intervalos de 45 graus.

Eles observaram que alguns dos 36 padrões funcionam como propriedades fundamentais da textura, sendo responsáveis por até 90% de todos os padrões presentes em uma imagem. A esse conjunto de padrões fundamentais foi dado o nome de "*uniform*" e eles estão representados pela primeira linha da Figura 2.8 (o número no centro de cada círculo informa o código *LBP* de cada padrão). A característica desses padrões é que eles possuem uma estrutura circular com até 2 transições de valor (0 e 1).

Uma das formas de contornar o problema da quantização "crua" do espaço angular, quando utilizados 8 pixels de vizinhança em intervalos de 45 graus, está na análise de um número maior de amostras, visto que a quantização do espaço angular é definida por  $(360 / P)$ . Para evitar a introdução de amostras redundantes, é estabelecida uma relação entre  $P$  e  $R$ , sendo a relação seguinte sugerida pelos autores do *LBP*: ( $P=8, R=1$ ), ( $P=16, R=2$ ) e ( $P=24, R=3$ ).

O *LBP riu2* se diferencia do *LBP ri* por trabalhar somente com 10 padrões, sendo 9 deles formados pelos padrões do tipo "uniform", mais um para os casos onde há mais de duas transições de valores dos pixels da vizinhança.

Com o emprego das técnicas de *bit shifting* e de restrição por padrão uniforme, foi possível reduzir o número de códigos gerados, conforme a Tabela 2.1.

**Tabela 2.1 - Número máximo de padrões gerados pelo *LBP***

	$P = 8, R = 1$	$P = 16, R = 2$	$P = 24, R = 3$
<i>LBP</i>	256	65536	16777216
<i>LBP ri</i>	36	4116	699252
<i>LBP riu2</i>	10	18	26

Esta redução do número de padrões tem reflexo direto na performance do sistema classificador, visto que a comparação entre as várias texturas se dá com menores vetores de características.

A introdução do *LBP* na literatura gerou enorme interesse sobre seu poder de caracterização e simplicidade, levando outros autores à proposição de novas extensões sobre os descritores originais. Ainda assim, recentes estudos ainda

conferem destaque à performance de caracterização obtida pelo *LBP*, mesmo quando comparado a novos descritores (GUO; ZHANG; ZHANG, 2010a; CHEN et al., 2013).

### 2.3.1. Descritores baseados no *LBP*

Devido a sua simplicidade e ao seu poder de caracterização, mesmo diante de variações de iluminação e escala, o *LBP* despertou muito interesse da comunidade, que apresentou diversas variações do descritor original, geralmente afirmando algum tipo de ganho sobre seu antecessor. Porém, o *LBP* ainda hoje é considerado um dos mais poderosos descritores disponíveis na literatura (MAANI; KALRA; YANG, 2013). A seguir, alguns descritores baseados no *LBP* serão apresentados.

O *Completed Local Binary Pattern* (*CLBP*) (GUO; ZHANG; ZHANG, 2010b), descritor invariante a rotação, se diferencia do *LBP* original por utilizar o sinal e a magnitude da diferença entre o pixel central e um pixel de sua vizinhança. Considerando  $q_c$  o pixel central,  $q_P$  um pixel na sua vizinhança circular de  $P$  elementos, a diferença  $d_P$  entre o pixel central e um pixel vizinho é decomposta conforme Equação 2.6:

$$d_P = s_P * m_P \quad (2.6)$$

na qual  $s_P$  é o sinal de  $d_P$  e  $m_P$  é a magnitude de  $d_P$ :

$$s_P = \begin{cases} 1, & \text{se } d_P \geq 0 \\ 0, & \text{se } d_P < 0 \end{cases} \quad (2.7)$$

$$m_P = |d_P|$$

O *CLBP* também considera a intensidade do pixel central. Assim, 3 operadores são definidos: o *CLBP\_S*, que considera o sinal componente de  $d_P$  e é o mesmo operador usado pelo *LBP* original, *CLBP\_M*, que considera a magnitude de  $d_P$  e *CLBP\_C*, que considera a intensidade do pixel central.

A definição do operador *CLBP\_M* é dada pela Equação 2.8:

$$CLBP\_M_{P,R} = \sum_{p=0}^{P-1} t(m_p, c) 2^p \quad (2.8)$$

sendo:

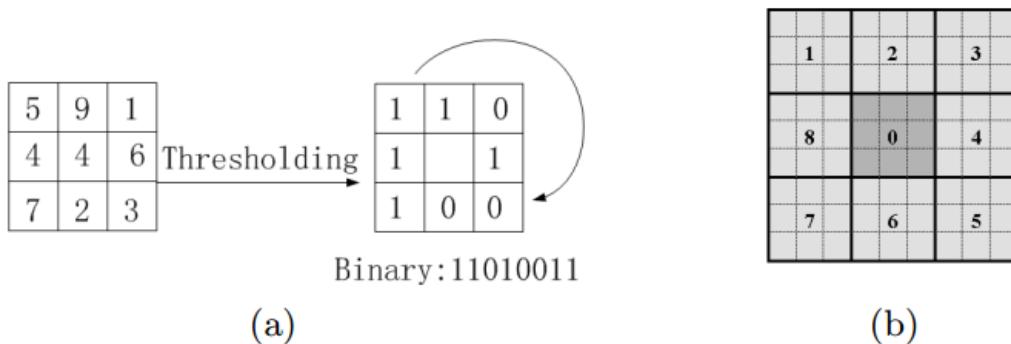
$$t(x, c) = \begin{cases} 1, & c \leq x \\ 0, & c > x \end{cases}$$

e  $c$  um valor limiar a ser determinado adaptativamente.

O último operador,  $CLBP\_C$  é definido como  $CLBP\_C = t(g_c, c_l)$ , sendo o valor limiar  $c_l$  definido como a média do nível de cinza de toda a imagem.

Esses 3 operadores,  $CLBP\_S$ ,  $CLBP\_M$  e  $CLBP\_C$  são então combinados em um histograma 3D para formar o mapa de características da imagem original.

Liao et al. (2007) propõem uma abordagem multiresolução diferente da empregada pelo  $LBP$ . Em seu método *Multi-scale Block Local Binary Patterns (MB-LBP)*, a comparação entre pixels é substituída pela comparação entre a média de valores de cinza de sub-regiões da imagem (Figura 2.9 (b)). Cada sub-região é um bloco quadrado contendo uma vizinhança de pixels. Considerando  $s$  como o tamanho do filtro em uma imagem medindo  $s \times s$  pixels, cada amostra da imagem é dividida em 9 desses blocos. Jia et al. (2013) aplicaram este método com sucesso no problema de detecção de falsas impressões digitais, sendo que o *MB-LBP* superou todos os outros descritores comparados.



**Figura 2.9 - (a) Operador *LBP* clássico. (b) Operador *MB-LBP 9x9***  
Fonte: Liao et al. (2007, p. 830)

No *LBP* original, a informação de magnitude da diferença entre os níveis de cinza do pixel central e dos pixels da vizinhança é descartada.

Com um nome similar ao do *CLBP* de Guo, Zhang e Zhang (2010b), o *C-LBP* apresentado por Ahmed et al. (2011) usa um código de 2 bits para codificar as propriedades de textura da imagem, de modo a capturar essa informação. O primeiro bit representa o sinal da diferença dos valores de cinza do pixel central e seus vizinhos, enquanto que o segundo é usado para codificar a magnitude da diferença dos pixels vizinhos com respeito a um valor limiar  $M_{avg}$ . Esse limiar é definido como a média das diferenças entre os valores de cinza do pixel central e dos pixels vizinhos. Considerando  $q_c$  como o pixel central e  $q_p$  um pixel da vizinhança, o código de 2 bits pode ser obtido por:

$$b(q_p, q_c) = \begin{cases} 00, & \text{se } q_p - q_c < 0 \text{ e } |q_p - q_c| \leq M_{avg} \\ 01, & \text{se } q_p - q_c < 0 \text{ e } |q_p - q_c| > M_{avg} \\ 10, & \text{se } q_p - q_c \geq 0 \text{ e } |q_p - q_c| \leq M_{avg} \\ 11, & \forall \text{ outro caso} \end{cases} \quad (2.9)$$

O *C-LBP* gera um código de 16 bits por pixel, para uma vizinhança de 8 elementos. A este código são aplicados os processos de *bit shifting* e de obtenção de código uniforme, de modo a reduzir o número de *bins* dos histogramas, que posteriormente são comparados.

Já Schaeffer e Doshi (2012) propõem uma técnica que combina descritores *LBP* invariantes à rotação em várias escalas multiresolução, chamada *LBP* multidimensional. Nesse processo, para cada pixel são gerados vários códigos *LBP*s em diferentes escalas e um histograma 3D é gerado, onde cada *bin* representa um código *LBP* em várias escalas.

García-Olalla et al. (2013) propõem um método híbrido para classificação de texturas invariante à rotação usando uma combinação de descritores de escopo local e global. Esse método tem o nome de *Adaptative Local Binary Pattern With Oriented Standard Deviation Term (ALBPS)* e se propõe a uma melhor descrição de texturas pela adição de informação de desvio da orientação padrão da textura ao operador *LBP* e do emprego de um classificador baseado em máquina de vetores de suporte (*SVM*, do inglês “*Support Vector Machine*”).

## 2.4. Descritores fuzzy

Dentre outras abordagens possíveis para a descrição de texturas está o emprego de conjuntos e sistemas *fuzzy*. A premissa para esta aplicação está no fato de que existem ambiguidades nas propriedades de textura inerentes à natureza da imagem, sejam elas introduzidas pelo processo de aquisição, digitalização ou pela presença de ruído, o que torna a tarefa de análise de texturas uma oportunidade para o emprego dessas técnicas (HANMANDLU; MADASU; VASIKALRA, 2004).

Como exemplo de abordagem *fuzzy* para a classificação de texturas o *Fuzzy Texture Spectrum* (BARCELÓ; MONTSENY; SOBREVILLA, 2005) é aqui apresentado. Neste método, os autores avaliam que o espectro de textura de uma imagem natural raramente apresenta *TUs* compostas por valores iguais a 1 (valor do nível de cinza do pixel central igual a algum pixel vizinho). Isso faria com que os possíveis valores de *TU* fossem reduzidos de  $3^8$  para  $2^8$ , com o espectro nunca sendo totalmente coberto e, portanto, “*misused*”. A ideia básica da solução constitui em atribuir os valores 0, 1 e 2 à *Texture Unit* conforme sua similaridade com o pixel central. Como este conceito de similaridade não é claramente definido, mas uma ideia vaga, um sistema *fuzzy* é empregado para este fim.

Assim, considerando uma amostra de imagem de tamanho 3x3 pixels como sendo  $V = \{V_0, V_1, V_2, V_3, V_4, V_5, V_6, V_7, V_8\}$ , com  $V_0$  representando o valor do nível de cinza do pixel central e  $V_i$  ( $1 \leq i \leq 8$ ) sendo os valores dos pixels vizinhos, na composição de uma *Texture Unit Fuzzy* cada pixel é representado por uma *Fuzzy Texture Unit Box (FTU)*  $FE_i$ , composta por 3 valores ao mesmo tempo:

$$FE_i = \{\mu_0(V_i), \mu_1(V_i), \mu_2(V_i)\}; \quad 1 \leq i \leq 8 \quad (2.10)$$

Na Equação 2.10,  $\mu_0(V_i)$ ,  $\mu_1(V_i)$  e  $\mu_2(V_i)$  são os graus de pertinência de  $V_i$  para os conjuntos *fuzzy* 0, 1 e 2, respectivamente, calculados a partir das funções dadas nas Equações 2.11, 2.12 e 2.13.

$$\mu_0 = \begin{cases} 1, & \text{se } x \leq -4 \\ -\frac{(x+1)}{3}, & \text{se } -4 < x < -1 \\ 0, & \text{se } x \geq -1 \end{cases} \quad (2.11)$$

$$\mu_1 = \begin{cases} 0, & se\ x \leq -4 \\ \frac{(x+4)}{3}, & se\ -4 < x < -1 \\ 1, & se\ -1 \leq x \leq 1 \\ \frac{(4-x)}{3}, & se\ 1 < x < 4 \end{cases} \quad (2.12)$$

$$\mu_2 = \begin{cases} 0, & se\ x \leq 1 \\ \frac{(x-1)}{3}, & se\ 1 < x < 4 \\ 1, & se\ x \geq 4 \end{cases} \quad (2.13)$$

Como exemplo, a Figura 2.10 apresenta os *FTUs* associados a uma imagem:

92	87	83
93	92	96
90	89	95

(a)

$\mu_1 = 1$	$\mu_0 = 1$	$\mu_0 = 1$
$\mu_1 = 1$		$\mu_2 = 1$
$\mu_1 = 2/3$ $\mu_0 = 1/3$	$\mu_0 = 1$	$\mu_2 = 2/3$ $\mu_1 = 1/3$

(b)

**Figura 2.10 - (a) Imagem em nível de cinza. (b) FTUs.**

Considerando todas as combinações de valores de similaridade, as *Fuzzy Texture Units (FTUs)* são determinadas. Multiplicando esses valores pela mesma matriz de pesos empregada no método *Texture Unit* e posteriormente somando-os, obtém-se os *Texture Unit Numbers (NTU)* correspondentes. Para o exemplo da Figura 2.10, as seguintes *FTUs* foram apuradas:

$$\begin{aligned} FTU1 &= \left\{ 1_1, 0_1, 0_1, 2_1, 2_{\frac{2}{3}}, 0_1, 1_{\frac{2}{3}}, 1_1 \right\}; N_{TU1} = 3133 \\ FTU2 &= \left\{ 1_1, 0_1, 0_1, 2_1, 1_{\frac{1}{3}}, 0_1, 1_{\frac{2}{3}}, 1_1 \right\}; N_{TU2} = 3052 \\ FTU3 &= \left\{ 1_1, 0_1, 0_1, 2_1, 2_{\frac{2}{3}}, 0_1, 0_{\frac{1}{3}}, 1_1 \right\}; N_{TU3} = 2404 \\ FTU4 &= \left\{ 1_1, 0_1, 0_1, 2_1, 1_{\frac{1}{3}}, 0_1, 0_{\frac{1}{3}}, 1_1 \right\}; N_{TU4} = 2323 \end{aligned} \quad (2.14)$$

O peso que cada *FTU* tem é determinado com a multiplicação dos oito valores apresentados pelas *FTUs*. Desse modo, as *FTUs* apresentadas na Equação 2.14 possuem os valores de pertinência dados na Equação 2.15:

$$\begin{aligned}\mu(N_{TU1}) &= \frac{4}{9} ; & \mu(N_{TU3}) &= \frac{2}{9} ; \\ \mu(N_{TU2}) &= \frac{2}{9} ; & \mu(N_{TU4}) &= \frac{1}{9} ;\end{aligned}\tag{2.15}$$

O espectro de textura é então montado como um histograma a partir de todas as *FTUs* (já balanceados pelos seus pesos) encontradas.

Apesar desse método propor uma melhor representação de uma textura, usando uma banda maior para a análise do espectro de texturas, ele ainda peca por trabalhar com uma matriz de pesos herdada do método *Texture Unit*, sendo, portanto, sensível à rotação das imagens.

#### **2.4.1. Local Fuzzy Pattern**

Em Boaventura e Gonzaga (2007), uma nova metodologia para a detecção de bordas em imagens baseada no conceito de números *fuzzy* é apresentada – a *Fuzzy Number Edge Detector (FUNED)*. Analisando seu potencial para descrição de regiões, VIEIRA et al. (2012a) apresentaram o *Local Fuzzy Pattern (LFP)* como uma metodologia para análise de texturas baseadas em números *fuzzy*.

Considerada como uma abordagem baseada em micro padrões, isto é, baseado na análise de estruturas de pixels em níveis de cinza com a capacidade de descrever um contexto espacial de uma imagem, difere dos demais métodos de análise de texturas *fuzzy* por não fazer uso de regras para obtenção de uma característica. Ao invés disso, analisa a vizinhança de pixels e, usando números *fuzzy*, determina uma relação com propriedades descritivas.

Tomando uma janela de  $W \times W$  pixels, a metodologia assume que cada distribuição dos níveis de cinza dos pixels vizinhos ao pixel central é um conjunto *fuzzy*, devido ao grau de incerteza gerado pelo processo de aquisição da imagem, pré-processamento e ao ruído produzido por diversas fontes. Cada pixel central possui

um identificador que indica a relação de seu valor de cinza para com os valores da vizinhança. O grau de pertinência do pixel central  $g(i,j)$  para a vizinhança  $WxW$  é dada pela média ponderada dos graus de pertinência dos pixels vizinhos, conforme a Equação 2.16:

$$\hat{\mu}_{g(i,j)} = \frac{\sum_{k=1}^W \sum_{l=1}^W (f_{g(i,j)} P(k, l))}{\sum_{k=1}^W \sum_{l=1}^W P(k, l)} \quad (2.16)$$

na qual  $f_{g(i,j)}$  é a função de pertinência aplicada para cada pixel vizinho e  $P(k, l)$  é uma matriz de pesos de mesmo tamanho que a vizinhança  $WxW$ . Esta formulação permite que outros descritores possam ser derivados dessa equação, aplicando-se uma função de pertinência e uma matriz de pesos específicos. Como exemplo, o *Local Binary Pattern*, pode ser derivado da Equação 2.16, adotando-se para a função de pertinência  $f_g(i,j)$  a função *degrau de Heaviside*<sup>1</sup> (Equação 2.17):

$$f_g(i,j) = H[A(k, l) - g(i, j)], \quad (2.17)$$

em que:

$$H[A(k, l) - g(i, j)] = \begin{cases} 0, & \text{se } H[A(k, l) - g(i, j)] < 0, \\ 1, & \text{se } H[A(k, l) - g(i, j)] \geq 0. \end{cases} \quad (2.18)$$

A matriz de pesos utilizada na derivação do *LBP*, para uma vizinhança  $3x3$  pixels é dada por:

$$P(k, l) = \begin{bmatrix} 1 & 2 & 4 \\ 128 & 0 & 8 \\ 64 & 32 & 16 \end{bmatrix} \quad (2.19)$$

Neste caso, o código *LBP* ( $N_{LBP}$ ) pode ser obtido pela Equação (2.20):

$$N_{LBP} = \hat{\mu}_{g(i,j)} \sum_{k=1}^W \sum_{l=1}^W P(k, l) \quad (2.20)$$

---

<sup>1</sup> Oliver Heaviside (1850-1925)

Já o *Local Fuzzy Pattern* determina o grau de pertinência de cada pixel vizinho em relação ao pixel central usando uma função sigmoide (Figura 2.11), determinada pela Equação 2.21:



**Figura 2.11 - Função de pertinência sigmoide**

$$f_{g(i,j)} = \frac{1}{1 + e^{\frac{-[A(k,l)-g(i,j)]}{\beta}}} \quad (2.21)$$

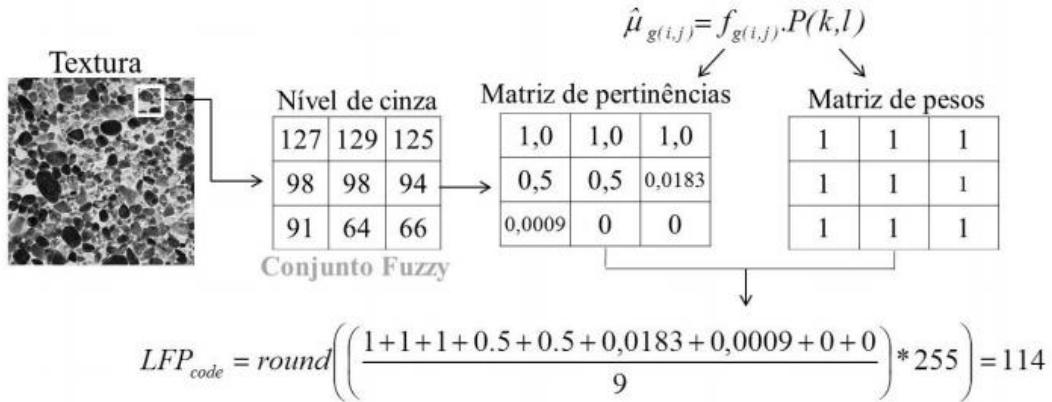
na qual  $\beta$  é a inclinação da curva sigmoide,  $A(k, l)$  são os pixels da vizinhança e  $g(i, j)$  é o pixel central. Vieira et al. (2012b) chamam o *LFP* baseado na função sigmoide como *LFP-s*.

O *LFP-s* considera que todos os pixels da vizinhança têm o mesmo peso na composição do grau de pertinência do pixel central, assim, para uma vizinhança de 3x3 pixels, a matriz de pesos  $P(k, l)$ , utilizada no *LFP-s*, é dada por:

$$P(k, l) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (2.22)$$

Após a multiplicação dos graus de pertinência dos pixels da vizinhança pelos seus respectivos pesos, uma média ponderada de seus valores é estabelecida. Este valor, que pode variar dentro da faixa de 0 a 1, é convertido para uma escala de cinza (0 – 255).

A Figura 2.12 apresenta um exemplo do cálculo do *LFP-s* para uma amostra de 3x3 pixels.



**Figura 2.12 - Cálculo do código LFP**  
Fonte: Vieira (2012, p. 61)

A partir dos códigos *LFP*-s já convertidos para escala de cinza, um histograma é montado, representando a identidade da textura analisada. O processo de comparação e classificação de amostras a ser executado pode ser o mesmo empregado no *LBP*.

Nos trabalhos apresentados pelos autores (VIEIRA et al., 2012a; VIEIRA et al., 2012b; VIEIRA et al., 2012c; CHIERICI et al., 2013), o *LFP*-s apresentou resultados superiores ao *LBP* na classificação de amostras não rotacionadas, quando foi aplicado ao banco de imagens de Brodatz (1966). Nesses estudos, o *LFP*-s também foi aplicado a um problema composto por amostras rotacionadas de texturas, também obtendo resultados superiores ao *LBP*.

Porém, é importante registrar o modo como os experimentos foram compostos. Na composição do conjunto de amostras para sintonização do  $\beta$ , havia a possibilidade de inclusão de amostras que também fariam parte do conjunto de teste. Há também a falta de diversificação das texturas utilizadas, ou seja, do banco de imagens empregado, pois o *LFP*-s foi experimentado basicamente em um banco de imagens, fato que também pode levantar dúvidas sobre sua empregabilidade a outros bancos de texturas.

Vale ainda ressaltar que o *LFP* é um descritor de texturas intrinsecamente invariante à rotação, devido ao emprego de uma matriz de pesos idênticos; e que é um método paramétrico, onde a definição da função de pertinência empregada (sigmoide), bem como sua parametrização é realizada de modo heurístico.

## 2.5. Considerações finais

A abordagem *fuzzy* permite uma maior tolerância a incertezas e presença de ruídos introduzidos pelo processo de aquisição de imagens. O método *Local Fuzzy Pattern* se diferencia dos demais disponíveis na literatura por fazer uso de números *fuzzy* para codificar as características de uma textura. Os demais métodos são baseados em regras *fuzzy* para codificar as características de uma textura, geralmente como uma abordagem híbrida, mapeando variáveis *fuzzy* em uma codificação do tipo do *LBP*.

O próximo capítulo introduz uma modificação do *LFP*, habilitando-o para a captura, em amostras rotacionadas de textura, de padrões em várias resoluções, além de apresentar uma metodologia para avaliação do potencial do descritor para problemas de classificação de amostras de texturas rotacionadas e estaticamente orientadas, ou seja, quando as amostras de texturas usadas para treino e teste tiverem a mesma orientação.



### 3. METODOLOGIA

#### 3.1. Considerações iniciais

Nas seções a seguir, uma variante do *LFP*-s é apresentada, o *Sampled Local Fuzzy Pattern* (*Sampled LFP*), adicionando suporte para análise multiresolução e melhor quantização do vetor de características. Os processos de classificação de texturas e de parametrização dos descritores baseados no *LFP* são definidos na sequência.

Apesar do *LFP*-s ter obtido bons resultados em trabalhos anteriores, mais análises devem ser conduzidas de modo a concluir pela sua viabilidade como um descritor de texturas.

Assim, o primeiro passo é determinar se o *LFP*-s é um descritor invariante à rotação ou não. A classificação de texturas invariante à rotação está mais ligada a problemas de processamento de imagens do mundo real, porém, na literatura, o caso mais investigado é o de classificação de amostras de texturas não rotacionadas (amostras estaticamente orientadas), tal qual nos estudos anteriores dos autores do *LFP*-s. Essa primeira análise é feita na seção 3.4 – *Avaliação do LFP para classificação de texturas*.

O segundo experimento (seção 3.5.1 – *Álbum de Brodatz*), com imagens do álbum de Brodatz (1966), submete o *Sampled LFP* a uma série de casos de teste especificados pelo Outex (OJALA et al., 2002), um arcabouço público, disponibilizado pela Universidade de Oulu, da Finlândia. O propósito aqui é o de avaliar o descritor em uma outra base de imagens e colher algumas informações a respeito de seu comportamento com relação à variação do tamanho das amostras, número de amostras que compõem o conjunto de treinamento e ângulo de rotação das amostras.

A seção 3.5.1 apresenta o experimento feito com casos de teste do Outex específicos para descritores invariantes à rotação, sendo um dos casos de teste composto por imagens capturadas sob diferentes condições de iluminação. O objetivo novamente é o de avaliar o descritor em uma outra base de imagens e colher algumas informações a respeito de seu comportamento com relação à variação do tamanho

das amostras, número de amostras que compõem o conjunto de treinamento e ângulo de rotação das amostras.

Por fim, a seção 3.5.2 apresenta uma análise do descritor quando aplicado a um problema de classificação de texturas do VisTex (VISTEX; 1995), um banco de imagens criado pelo Media Lab do Massachusetts Institute of Technology. Tem o propósito de verificar o comportamento do *Sampled LFP* aplicado a um terceiro banco de imagens e o de analisar o comportamento do descritor diante da variação do número de amostras disponíveis para treinamento.

### **3.2. Sampled Local Fuzzy Pattern**

O *Local Fuzzy Pattern Sísmoide* trabalha utilizando uma janela de análise medindo 3x3 pixels, o que acaba limitando o tamanho dos padrões detectáveis. Porém, padrões podem ser formados em diferentes resoluções e ainda é possível a existência de padrões distintos conforme a resolução: é possível existir um determinado padrão numa janela de 3x3 pixels e outro distinto, numa janela expandida de 5x5 pixels. A escolha da resolução empregada na descrição de textura é feita de maneira heurística, visto que determinada característica pode ser mais evidente em determinada resolução.

Enquanto o *LFP*-s pode se adaptar à análise multiresolução pelo incremento de sua janela de  $W \times W$  pixels, o *Sampled Local Fuzzy Pattern (Sampled LFP)* compõe a vizinhança de um pixel central a partir de uma vizinhança de simetria circular de raio  $R$ . A configuração da análise multiresolução é dada pelo número de vizinhos, ou amostras, e pelo valor do raio da vizinhança.

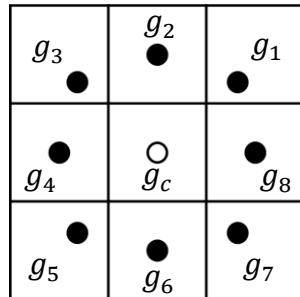
A escolha da abordagem de vizinhança circular não é por acaso. Conforme Cross e Jain (1983), "o nível de brilho em um ponto de uma imagem é altamente dependente do nível de brilho dos pontos vizinhos, a menos que a imagem seja simplesmente ruído". Assim, essa dependência entre o nível de cinza do pixel central e de seus vizinhos deveria levar em conta a distância entre os pixels analisados – pixels mais distantes do pixel central tem menor capacidade de influenciá-lo. Dessa maneira, ao ser utilizada uma vizinhança circular e simétrica, todos os pixels da

vizinhança terão a mesma distância e, portanto, o mesmo poder de influência sobre o pixel central.

Ojala et al. (2000) apontaram outro argumento em favor do emprego da vizinhança circular e do emprego da análise multiresolução. De acordo com os autores, a sensibilidade do descriptor invariante à rotação  $LBP_8^{riu^2}$  é prejudicada devido a quantização “crua”, em  $45^\circ$ , do espaço angular. Para contornar este problema, o espaço angular foi quantizado a uma resolução mais fina a partir de uma vizinhança circular simétrica. Este mesmo espaço de amostragem foi utilizado em Porter e Canagarajah (1997), quando alguns descritores de textura baseados em *wavelets* e campo aleatório de Markov foram apresentados como invariantes à rotação.

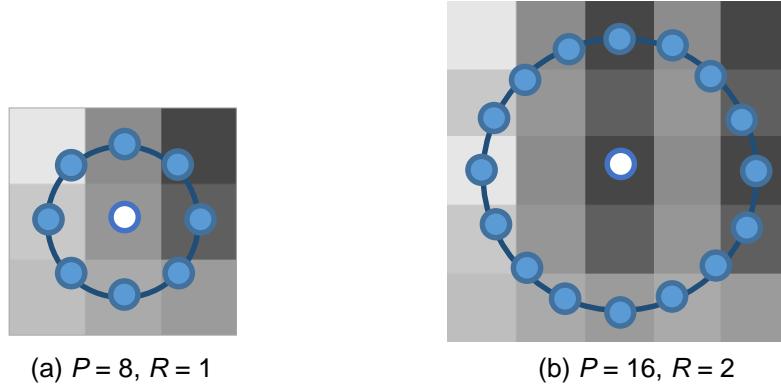
Assim, sendo  $P$  o número de pixels que compõem o conjunto de pixels vizinhos,  $g_p$  corresponde aos valores de nível de cinza de cada pixel  $P$ , igualmente espaçados em um círculo de raio  $R$  ( $R > 0$ ), formando um conjunto de vizinhos de simetria circular, com ordem de disposição anti-horária (Figura 3.1). Considerando o pixel central  $g_c$  posicionado nas coordenadas  $(0, 0)$ , as coordenadas de  $g_p$  são dadas pela Equação 3.1:

$$(-R \operatorname{seno}\left(\frac{2\pi p}{P}\right), R \operatorname{coseno}\left(\frac{2\pi p}{P}\right)) \quad (3.1)$$



**Figura 3.1 - Ordem de disposição da vizinhança circular**

Os valores do nível de cinza de vizinhos não localizados no centro exato de um pixel são estimados por interpolação bilinear. Dois exemplos da vizinhança circular utilizada no *Sampled LFP* são dados na Figura 3.2.



**Figura 3.2 - Vizinhança circular utilizada no *Sampled LFP***

A função de pertinência do pixel central, para a vizinhança de  $P$  elementos é dada pela média ponderada dos graus de pertinência dos pixels vizinhos, conforme a Equação 3.2:

$$\hat{\mu}_{g(c)} = \frac{\sum_{p=1}^P f_{g(p)}}{P} \quad (3.2)$$

na qual  $f_{g(p)}$  (Equação 3.3) é a função de pertinência dos pixels vizinhos em relação ao pixel central. Tal como no *LFP-s*, uma função sigmoide é utilizada aqui, porém, com a diferença que agora os pixels vizinhos são referenciados pela ordem de seu posicionamento e não mais por coordenadas:

$$f_{g(p)} = \frac{1}{1 + e^{-\frac{[g_c - g(p)]}{\beta}}} \quad (3.3)$$

O *Sampled LFP* garante a característica de invariância à rotação pela não valorização da posição de cada pixel, assim uma matriz de pesos não é utilizada.

Outra diferença para o *LFP-s*, é que o *LFP-s* faz uma varredura de todos os pixels de sua janela  $W \times W$ , incluindo o pixel central, e calcula o grau de pertinência de cada um desses pixels para o pixel central. Já o *Sampled LFP* utiliza somente os graus de pertinência dos pixels da vizinhança para seu cálculo.

No processo de identificação de texturas, cada amostra de imagem de uma textura é analisada, calculando os códigos *Sampled LFP* para toda a imagem. O identificador de cada textura é o histograma de códigos *Sampled LFP*. Porém, como este descriptor gera valores no intervalo  $[0,1]$ , é necessária sua quantização. Aqui, o

*Sampled LFP* se diferencia novamente do *LFP-s* por não converter o código *LFP* em níveis de cinza (256 valores). Ao invés disso, estabelece o número de *bins* como uma variável parametrizável, tal qual o valor de  $\beta$ , utilizado pela função de pertinência de cada pixel da vizinhança (Equação 3.3). É esperado que os graus de pertinência dos pixels (códigos *Sampled LFP*) possam ser agrupados em faixas de similaridade, possibilitando uma melhor descrição da textura, além de melhoria de performance do processo de classificação, visto que o número de *bins* tem impacto direto no esforço computacional exigido pela comparação de histogramas.

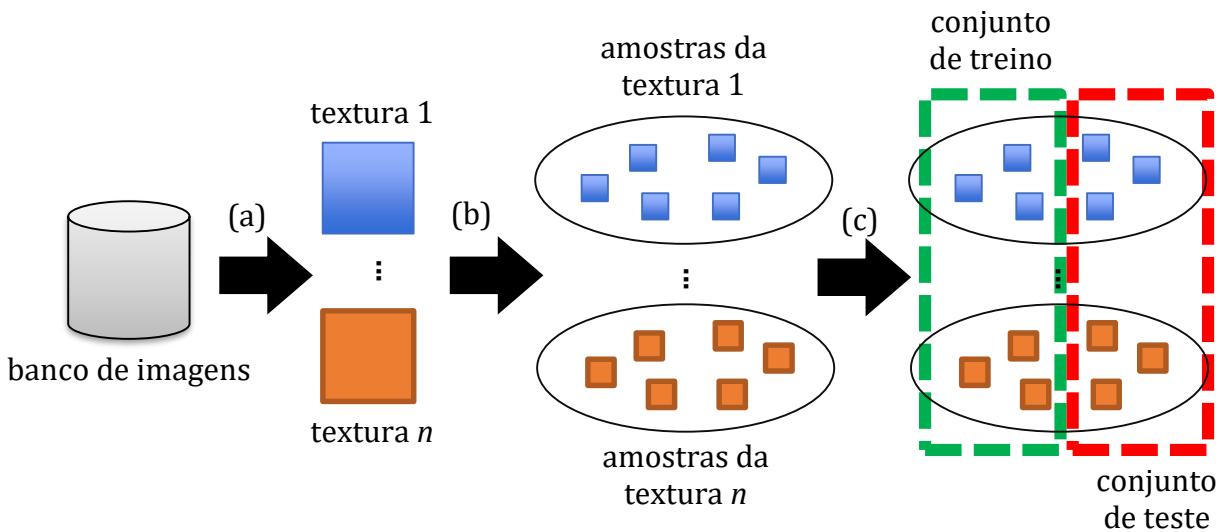
### **3.3. O processo de classificação**

O método para classificação das texturas abordado neste trabalho é o da categorização de texturas com base em sua aparência, a partir de amostras de imagens. Na abordagem adotada, não há restrições impostas sobre o processo de aquisição das imagens, nem sobre as condições de iluminação ou rotação.

Como a cor do material é uma fonte adicional de informação sobre sua textura, não é feito uso de qualquer informação sobre a cor, a fim de tornar o problema mais genérico. Além disso, imagens de textura muitas vezes passam por uma mudança radical em sua aparência quando seus materiais são submetidos a variações da condição de iluminação, sendo que muitas vezes dois materiais diferentes podem apresentar a mesma aparência quando submetidos a condições diferentes de iluminação. Lidar com essas variações com sucesso é uma das tarefas de qualquer algoritmo de classificação de texturas.

Nos vários experimentos executados para a avaliação do descritor *LFP*, todos eles utilizaram o mesmo processo de classificação aqui descrito.

Para todos os experimentos um ou mais banco de imagens foram selecionados. A partir de cada banco, um conjunto de imagens é separado (Figura 3.3, passo *a*). Cada uma dessas imagens representa um tipo de textura e cada uma delas é dividida em  $n$  amostras (Figura 3.3, passo *b*). Parte dessas amostras de cada tipo de textura irá compor o conjunto de imagens de treinamento e as restantes irão compor o conjunto de teste (Figura 3.3, passo *c*).

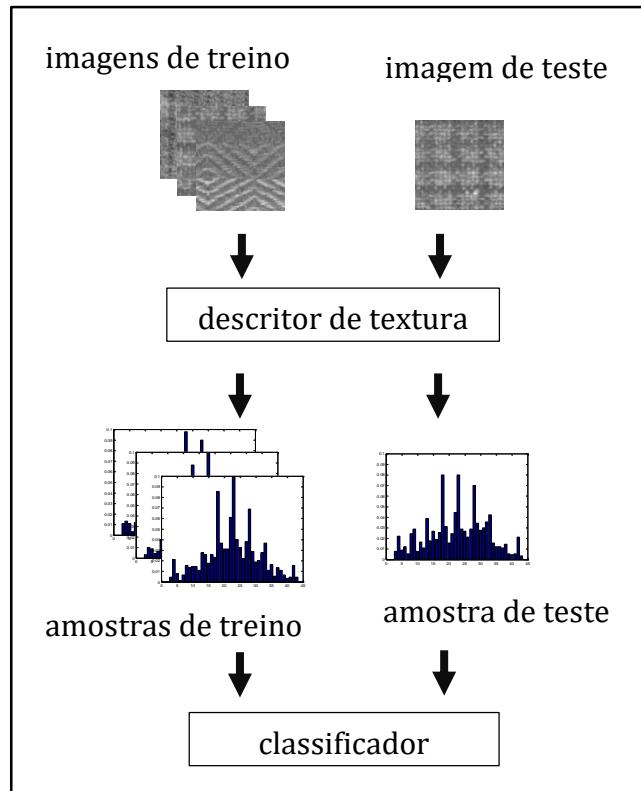


**Figura 3.3 - Separação de imagens para conjuntos de treino e teste**

Antes do processo de classificação ser efetivamente iniciado, os efeitos das propriedades de primeira e segunda ordem da escala de cinza de cada imagem são removidos. Isso foi feito normalizando-se a intensidade de cada amostra, de maneira a ter uma intensidade média de 128 e um desvio médio de 20. Porém, esse preprocessamento não foi aplicado ao experimento executado com o Outex\_TC\_00012 (vide item 3.5.2 – Outex), visto que seu objetivo é o de avaliar a performance dos descritores na classificação de amostras capturadas sob diversas condições de iluminação.

Para cada amostra, é aplicado determinado descritor de texturas. A seguir é construído um histograma registrando a distribuição das probabilidades dos códigos gerados. Esse histograma é considerado o identificador daquela amostra. Se o descritor escolhido for o *LBP*, então a construção do histograma é direta, visto que este operador gera um valor discreto, porém se o descritor escolhido for o *LFP*, o valor de seu código deverá ser quantizado, de modo a permitir a construção do histograma.

Cada histograma que compõe o conjunto de teste é comparado contra todos os histogramas do conjunto de treinamento, utilizando-se métrica de distância para avaliação do melhor ajuste entre histogramas. Uma amostra de teste é considerada corretamente classificada se tiver a mesma classe da amostra de treino para qual se obteve a menor distância, como ilustrado pela Figura 3.4.



**Figura 3.4 - Processo de classificação**

Entre as diversas medidas de distâncias utilizadas pela literatura para determinação de maior semelhança entre histogramas, foi avaliada a métrica chi-quadrado (Equação 3.4), por ser amplamente aplicada em experimentos similares.

Considerando S a amostra atribuída ao modelo M, a distância chi-quadrado é dada por:

$$X(S, M) = \frac{1}{2} \sum_{b=1}^B \frac{(S_b - M_b)^2}{(S_b + M_b)} \quad (3.4)$$

Nestas equações,  $B$  é o número de *bins* dos histogramas e  $S_b$  e  $M_b$  são as probabilidades do *bin*  $b$ , na amostra e no modelo, respectivamente.

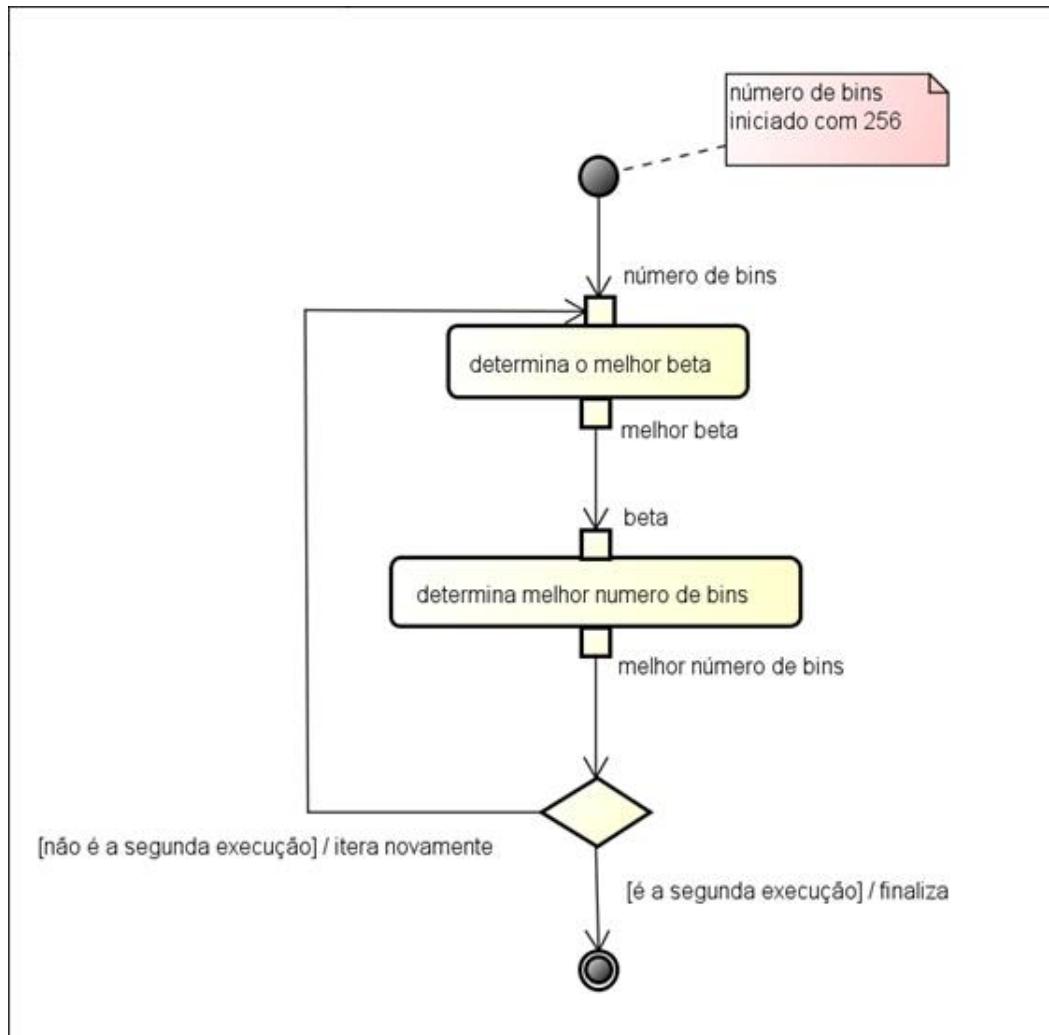
Uma matriz de confusão é utilizada para armazenamento dos resultados e o cálculo da sensibilidade ( $S$ ) de cada descritor é dado pela Equação 3.5, onde  $VP$  é o número de verdadeiros positivos e  $FN$  o número de falsos negativos:

$$S = \frac{VP}{VP + FN} \quad (3.5)$$

### 3.3.1. Parametrização do Sampled LFP

O *Sampled Local Fuzzy Pattern* é um método paramétrico, o que significa que existem alguns parâmetros que devem ser ajustados. A determinação dos melhores valores de  $\beta$ , utilizado na definição do valor da pertinência de cada pixel da vizinhança, é chamada sintonização e é feita por meio de treinamento, onde o processo de classificação é repetido variando-se o parâmetro em questão, até obter a sensibilidade máxima. A determinação do melhor número de bins consiste da repetição do processo de classificação, variando-se o número de intervalos da faixa de valores do LFP. A ideia aqui é a de agrupar valores similares em cada faixa e gerar histogramas com o menor número de bins, sem causar perda de sensibilidade.

Nos experimentos executados, a determinação dos valores de  $\beta$  e do número de *bins* consistiu da classificação apenas para o conjunto de treinamento, utilizando validação cruzada *leave-one-out*, onde cada amostra de treino é comparada contra todas outras, menos a da comparação, e métrica chi-quadrado, conforme indicado pela Figura 3.5. A faixa de possíveis valores para  $\beta$  foi definida entre [0.1, 2.5], com intervalo de 0.050, e a de possíveis valores para o número de *bins* entre [5, 256], com intervalo de 5 *bins*. Essas faixas de valores, bem como os valores dos intervalos foram definidos empiricamente.



**Figura 3.5 - Processo de parametrização do *Sampled LFP***

### 3.4. Avaliação do *LFP* para classificação de texturas

Por ser um descritor de texturas recentemente introduzido na literatura, o *Local Fuzzy Pattern* não foi testado em muitos casos até o momento. Nos trabalhos publicados pelos autores (VIEIRA et al., 2012a; VIEIRA et al., 2012b; VIEIRA et al., 2012c; CHIERICI et al., 2013), a apreciação do método foi feita com base em dois experimentos. No primeiro, o *LFP* foi aplicado para o problema de classificação de texturas, utilizando o banco de imagens de Brodatz (1966) com amostras sem mudanças de orientação. Já o segundo experimento compreendeu dois casos de teste em que imagens de textura do banco Outex (OJALA et al., 2002) foram utilizados para avaliação de sensibilidade do *LFP*-s aplicado ao problema de classificação invariantes à rotação. Para este último experimento, as amostras de texturas do banco Outex

tinham orientações diferentes para as amostras de treino e teste. Apesar do *LFP*-s ter superado o *LBP*, utilizado como referência para comparação, o processo de parametrização do valor de  $\beta$  não utilizou um conjunto distinto das amostras de teste para esta sintonização.

Assim, nesse primeiro experimento, o *LFP* é avaliado objetivando estabelecer se este descritor tem potencial para aplicação em problemas que exigem um classificador invariante à rotação.

Para esta avaliação, uma série de testes foi executada tendo como base o *Outex*, um arcabouço público, disponibilizado pela Universidade de Oulu, da Finlândia, composto por um banco de texturas capturadas em diversas condições de iluminação, mais suítes de testes, criados com o objetivo de facilitar a criação e a comparação de algoritmos destinados à classificação e segmentação de texturas (OJALA et al., 2002). Dois exemplos de texturas, capturadas sob a mesma condição de iluminação, com e sem rotação, são apresentadas pela Figura 3.6.



(a) Canvas014, Inca, 00



(b) Canvas014, Inca, 30



(c) Wood010, Inca, 00



(d) Wood010, Inca, 45

**Figura 3.6 - Amostras de texturas do banco de imagens Outex**

Cada suíte de teste do Outex é especificamente composta por determinadas classes de texturas, sendo que cada classe foi digitalizada sobre determinada condição de iluminação e rotação, gerando um conjunto de amostras em escala de cinza. Cada suíte de teste especifica quais amostras serão utilizadas na fase de treinamento e teste, além de determinar seu número de casos de testes, onde cada um deles é composto por um subconjunto das amostras de treinamento e teste.

Foram selecionados 14 casos de testes na experimentação que estão indicados na Tabela 3.1, na qual *TA* é o tamanho das amostras em pixels, *NC* é o número de classes de texturas, *NAC* é o número de amostras por classe, *NTA* é o número total de amostras, *TIL* é o tipo de iluminação utilizado na captura das texturas, *RO* são os ângulos de rotação das texturas, *NATRE* é o número de amostras para treinamento, *NATES* é o número de amostras para teste e *NT* é o número de testes disponíveis para a suíte de teste indicada.

**Tabela 3.1 - Suítes de teste do Outex selecionadas para experimento**

Suíte de teste	TA	NC	NAC	NTA	TIL	RO	NATRE	NATES	NT
TC_00000	128	24	20	480	Inca	00	240	240	100
TC_00001	64	24	88	2112	Inca	00	1056	1056	100
TC_00002	32	24	368	8832	Inca	00	4416	4416	100
TC_00003	128	24	20	480	Inca	00	240	240	100
TC_00004	64	24	88	2112	Inca	00	1056	1056	100
TC_00005	32	24	368	8832	Inca	00	4416	4416	100
TC_00006	64	24	88	2112	Inca	00	1056	1056	100
TC_00007	32	24	368	8832	Inca	00	4416	4416	100
TC_00008	64	24	88	2112	Inca	00	1056	1056	100
TC_00009	32	24	368	8832	Inca	00	4416	4416	100
TC_00010	128	24	180	4320	Inca	00,05,10, 15,30,45, 60,75,90	2160	2160	1
TC_00011	128	24	40	960	Inca	00	480	480	1
TC_00012	128	24	380	9120	Horizon, Inca, TL84	00,05,10, 15,30,45, 60,75,90	4580	4580	2
TC_00015	128	68	20	1360	Horizon, Inca, TL84	00	680	680	100

A fonte de iluminação utilizada para captura das amostras foi um cabide de luzes SpectraLight III da empresa Macbeth e a câmera fotográfica utilizada foi uma Sony CCD DXC-755P. Os tipos de iluminação utilizados no Outex são definidos como

Inca, um iluminante incandescente, de temperatura de cor 2856K padrão CIE A; Horizon, iluminação do sol a 0º (horizonte), de temperatura 2300K e TL84, uma iluminante fluorescente fria, de temperatura 4000K. As amostras rotacionadas foram obtidas a partir da amostra principal, que foi centralizada em uma prancheta, rotacionada em determinado ângulo e então registrada pela câmera.

As suítes de teste TC\_00013 e TC\_00014 foram deixadas de fora por trabalharem com texturas coloridas, algo não abordado neste trabalho.

Apesar de algumas suítes terem aparentemente a mesma configuração, com a mesma quantidade, iluminação e tamanho das amostras, elas são compostas por diferentes tipos de texturas, o que contribui para a variedade de casos e generalidade do método.

Para cada suíte de teste, o *LFP*-s original foi parametrizado (valores de  $\beta$  e número de *bins*) e sua sensibilidade foi calculada, conforme descritos nas seções sobre os processos de classificação e parametrização. Esses resultados foram então comparados com aqueles obtidos pelo descriptor *LBP* mais indicado para cada suíte de teste (*LBP riu2* para classificação de amostras texturas rotacionadas e *LBP* clássico para amostras sem variação de orientação). Assim, se determinada suíte compreende amostras de texturas em uma única orientação, o descriptor escolhido é o *LBP* clássico. Caso a suíte de teste seja referente a um problema de classificação de amostras rotacionadas, o descriptor invariante à rotação escolhido é o *LBP riu2*. Com isso é determinado se o *LFP* pode ser considerado um descriptor invariante à rotação.

Porém, numa aplicação do mundo real não se sabe de antemão se a textura a ser classificada foi ou não rotacionada, consequentemente não permitindo a escolha do método a ser aplicado. Portanto, é desejado que um mesmo operador seja capaz de trabalhar com um banco de imagens rotacionadas ou não. Dessa forma o *LBP riu2* foi testado como um descriptor “universal”, invariante à rotação, sendo aplicado a todas às suítes de teste do Outex. Seus resultados foram comparados com os do *LFP*-s.

A escolha do *LBP* como o descriptor de referência a ser usado para comparação não é por acaso. Diversos outros trabalhos trataram de sua extensão e aprimoramento, como o *Completed modeling of Local Binary Pattern – CLBP* (GUO; ZHANG; ZHANG, 2010b) e o *Monogenic-LBP* (ZHANG et al.; 2010). Assim, o *LBP* pode ser considerado como o ponto de partida para diversos outros descritores e, dado que o *LFP* propõe uma nova abordagem para classificação de texturas e o

estudo de seus fundamentos e capacidade está em seu estágio inicial, sua comparação com o *LBP* proporciona um cenário justo para sua avaliação, pavimentando o caminho para futuras melhorias e extensões.

### **3.5. Classificação de amostras de texturas rotacionadas e em multiresolução**

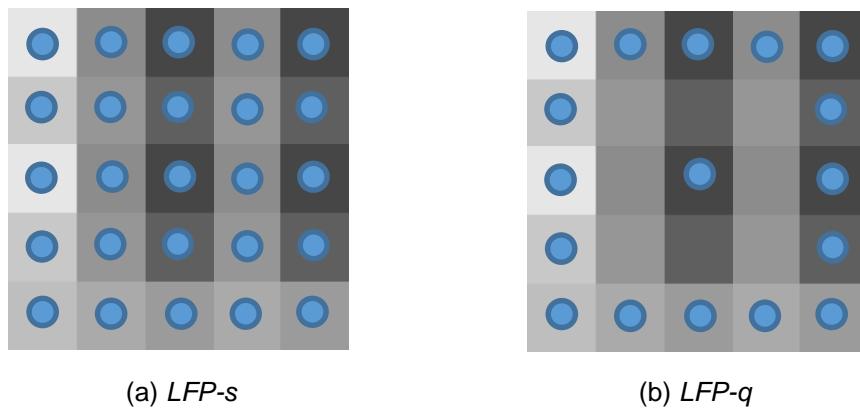
Para a validação do *Local Fuzzy Pattern* como um descritor robusto a variações de rotação, bem como a validação de seu poder de caracterização em multiresolução, é necessária a geração de evidências claras, vindas de experimentos conduzidos com variadas bases de imagens, compostos por amostras de tamanhos variados e por diferentes ângulos de rotação, além de ser também necessário colocá-lo sob comparação diante de um descritor de alta performance também designado para este tipo de trabalho.

Dessa maneira, alguns experimentos foram conduzidos, comparando a performance do *Sampled LFP* ao *LBP riu2*, diante de 3 bases de imagens rotacionadas: Brodatz, Outex e VisTex (VISTEX; 1995).

Para todos os experimentos, foram utilizados os mesmos processos para parametrização e classificação do *LFP* já introduzidos anteriormente, além dos operadores *Sampled LFP* e *LBP riu2* terem suas performances comparadas para as resoluções de 8x1, 16x2 e 24x3 pixels.

Além desses descritores, o *LFP-s* recebeu uma pequena alteração para permitir sua operação em modo de multiresolução e assim, sua comparação com os demais descritores.

Dado que o *LFP-s* trabalha com uma janela de  $W \times W$  pixels, onde todos os pixels são analisados para cálculo do grau de pertinência do pixel central, foi introduzida uma modificação para permitir que este descritor opere em outras resoluções diferentes da janela de 3x3 pixels e assim, sua comparação com os demais descritores. A essa variante do *LFP-s* foi dado o nome de *LFP-q*, de vizinhança quadrada e sua diferença para o descritor original é que somente os pixels de borda da vizinhança são utilizados para o cálculo do grau de pertinência do pixel central. Como exemplo, a Figura 3.7 ilustra os pixels utilizados pelo *LFP-s* e *LFP-q* no cálculo do grau de pertinência do pixel central, para uma janela de 5x5 pixels.



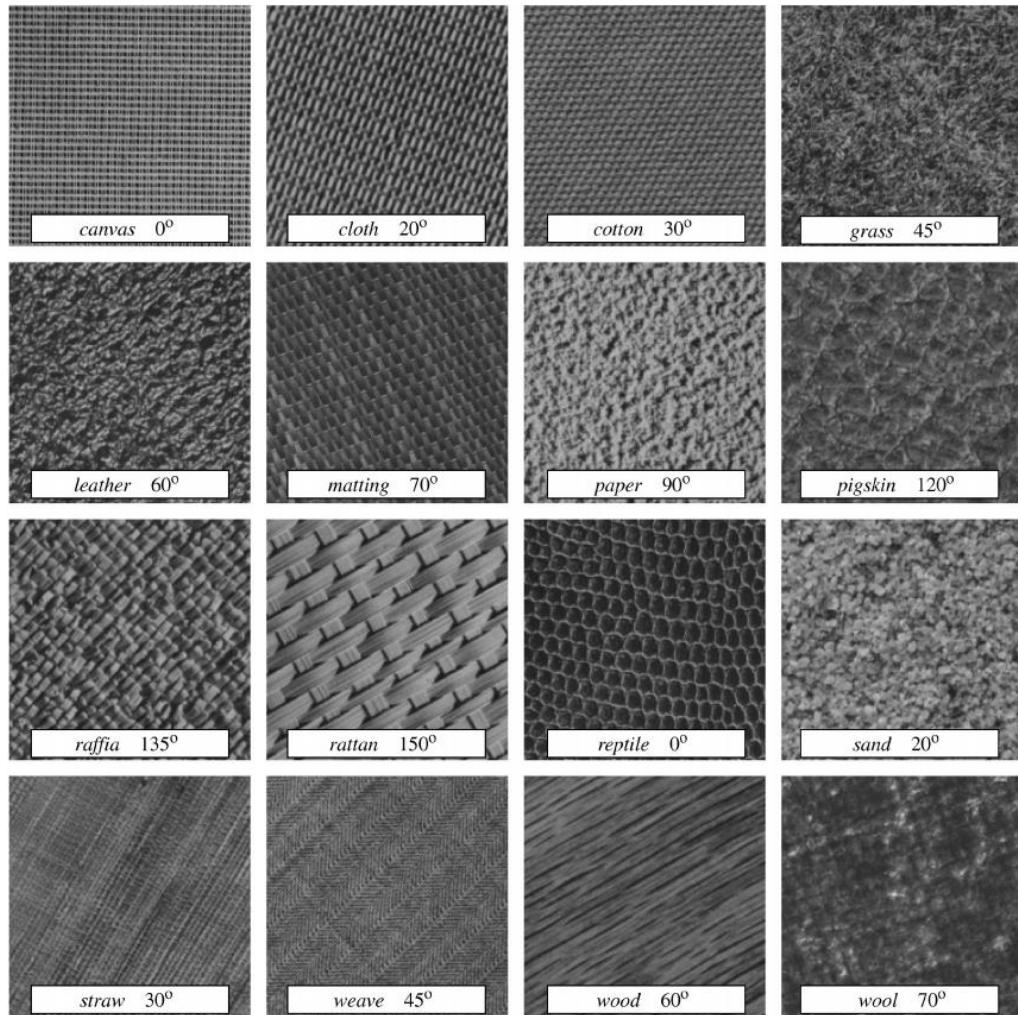
**Figura 3.7 - Pixels utilizados para cálculo do grau de pertinência do pixel central, para uma vizinhança 5x5 pixels: (a)  $LFP-s$ ; (b)  $LFP-q$**

### 3.5.1. Álbum de Brodatz

O álbum de Brodatz é um banco de imagens bem conhecido e utilizado para a avaliação de algoritmos de classificação de texturas. Nesse experimento, 3 configurações diferentes são utilizadas para a avaliação dos descritores quanto sua invariância à rotação e suporte a multiresolução.

Baseados nos experimentos feitos por Porter e Canagarajah (1997), Ojala, Pietikäinen e Mäenpää (2002) replicaram a configuração utilizada pelos primeiros e a disponibilizaram como uma suíte de teste do Outex, a Contrib\_TC\_00000.

Para a construção dessa suíte, foram utilizadas 16 texturas do álbum de Brodatz (Figura 3.8), sendo que para cada classe de textura, originalmente havia oito imagens de 256x256 pixels. A primeira imagem foi usada para treinamento do descritor de texturas e as outras sete foram usadas para execução dos testes de sensibilidade. Cada imagem foi rotacionada em diversos graus, criando novas imagens de 180x180 pixels.



**Figura 3.8 - Amostras das 16 texturas usadas no experimento I do álbum de Brodatz**  
Fonte: Ojala, Pietikäinen e Mäenpää (2002, p. 978)

Nos experimentos com o álbum de Brodatz, os descritores foram treinados com amostras rotacionadas a partir de determinados ângulos e testados com amostras rotacionadas em outros ângulos. Isso foi assim conduzido porque a orientação da textura, tal qual veios de madeira que podem partir da esquerda para a direita em uma amostra, pode ser alterada com a rotação dessa imagem. Um descritor invariante à rotação, que nunca tenha sido treinado para classificar amostras rotacionadas nesse ângulo, ainda assim deverá classificar essa amostra como madeira.

Na Contrib\_TC\_00000 os ângulos 0°, 30°, 45° e 60° foram escolhidos para treinamento e cada uma dessas 4 imagens foi dividida em 121 amostras de 16x16 pixel, totalizando 7744 amostras para treino.

Visto que as amostras de 16x16 pixels poderiam ser muito limitadas para serem consideradas confiáveis e que 7744 amostras diferentes poderiam resultar em um

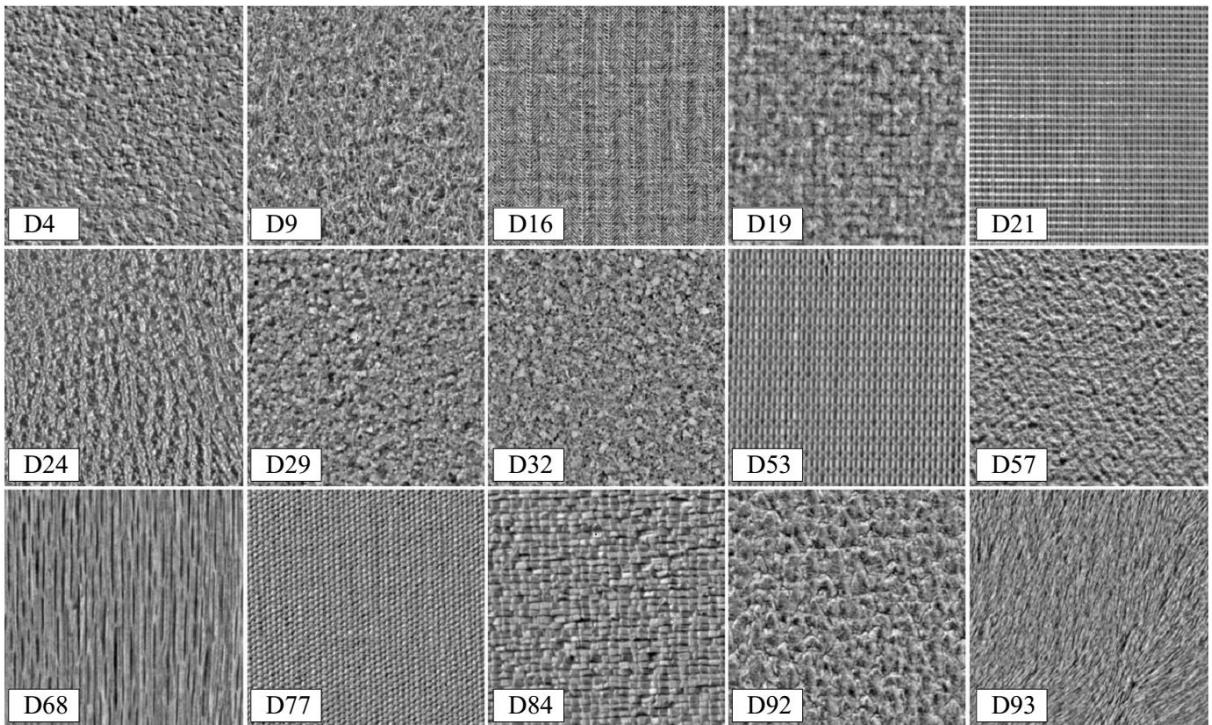
grande esforço computacional para sua comparação, os histogramas de cada operador de textura, para todas as amostras pertencentes a uma classe particular, foram somados, gerando um único histograma representativo da classe (OJALA; PIETIKÄINEN; MÄENPÄÄ, 2002). Já o conjunto de teste foi composto pelos ângulos 20°, 70°, 90°, 120°, 135° e 150°, sendo que havia sete amostras de 180x180 pixels para cada ângulo e classe de textura (42 amostras por classe), totalizando 672 amostras para teste. A Tabela 3.2 apresenta um resumo da configuração desse primeiro experimento:

**Tabela 3.2 - Configuração do experimento I de Brodatz**

<b>Número de classes</b>	16
<b>Treino</b>	Amostras 16 x 16 pixels; 4 ângulos – 00°, 30°, 45° e 60°; 121 amostras por ângulo e classe; (484 amostras por classe); Total de 7744 amostras;
<b>Teste</b>	Amostras 180 x 180 pixels; 6 ângulos – 20°, 70°, 90°, 120°, 135° e 150°; 7 amostras por ângulo e classe; (42 amostras por classe); Total de 672 amostras;

O segundo experimento usando o álbum de Brodatz utiliza a suíte de teste Contrib\_TC\_00002, também analisada por Pietikäinen, Ojala e Xu (2000) e baseada nos experimentos originais de Porter e Canagarajah.

Para este experimento, 15 classes de texturas foram escolhidas (Figura 3.9), cada qual fornecida como uma imagem. Cada uma delas foi rotacionada em 11° ao redor de seu centro por meio de interpolação bi cúbica, de maneira a ter um comportamento uniforme com respeito aos efeitos de interpolação aplicados a qualquer ângulo usado no experimento (PIETIKÄINEN; OJALA; XU, 2000). Imagens nessa angulação foram chamadas de imagens de referência e foram usadas para compor o conjunto de amostras de treino.



**Figura 3.9 - Amostras das 15 texturas usadas no experimento II do álbum de Brodatz**  
Fonte: Ojala e Pietikäinen (1999, p. 5)

As imagens usadas para teste dos classificadores foram geradas rotacionando cada imagem ao redor de seu centro no sentido anti-horário, com os seguintes graus de rotação: 30°, 60°, 90°, 120°, 150° e 200°.

As amostras das texturas foram extraídas a partir de um retângulo imaginário de 256x256 pixels, centrado na imagem de cada textura/ângulo. Cada um dos ângulos de teste foi avaliado separadamente. A Tabela 3.3 apresenta o resumo da configuração do experimento II de Brodatz.

O terceiro e último experimento usando o álbum de Brodatz utiliza a suíte de teste Contrib\_TC\_00003 e tem basicamente a mesma configuração do experimento II, com a diferença do tamanho e do número de amostras. Nessa configuração as amostras possuem 64x64 pixels e os conjuntos de treino e teste são compostos por 240 e 1440 amostras, respectivamente (Tabela 3.4).

**Tabela 3.3 - Configuração do experimento II de Brodatz**

<b>Número de classes</b>	15
<b>Tamanho das amostras</b>	32 x 32 pixels
<b>Treino</b>	Ângulo 11º; 64 amostras por classe; (15 classes * 64 amostras); Total de 960 amostras;
<b>Teste</b>	6 testes, um por ângulo; 6 ângulos: 30º, 60º, 90º, 120º, 150º, 200º; 64 amostras por classe e ângulo; (15 classes * 64 amostras * 6 ângulos); Total de 5760 amostras;

**Tabela 3.4 - Configuração do experimento III de Brodatz**

<b>Número de classes</b>	15
<b>Tamanho das amostras</b>	64 x 64 pixels
<b>Treino</b>	Ângulo 11º; 16 amostras por classe; (15 classes * 16 amostras); Total de 240 amostras;
<b>Teste</b>	6 testes, um por ângulo; (30º, 60º, 90º, 120º, 150º, 200º); 16 amostras por classe e ângulo; (15 classes * 16 amostras * 6 ângulos); Total de 1440 amostras;

É importante ressaltar o interesse original de seus autores na construção dessas suítes de teste, que foi a avaliação de seu descritor invariante à rotação observando sua performance na análise de amostras pequenas (16x16, 32x32 e 64x64 pixels).

### 3.5.2. Outex

O banco de texturas Outex possui apenas 2 suítes de teste concebidas para a avaliação de classificadores invariantes à rotação, a Outex\_TC\_00010 e a Outex\_TC\_00012.

A Outex\_TC\_00010 é composta por 24 classes de texturas, digitalizadas sob iluminação incandescente, chamada "inca" e em nove ângulos diferentes de rotação. O conjunto de treinamento é composto pelas amostras não rotacionadas ( $0^\circ$ ) de todas as 24 classes e o conjunto de teste é composto pelas amostras dos ângulos restantes. Originalmente esta suíte de teste possui amostras de 128x128 pixels, mas, visando analisar a sensibilidade dos descritores diante de amostras menores, também foi experimentado amostras no tamanho de 64x64 pixels, conforme configuração indicada pela Tabela 3.5. Para isso, cada amostra original foi dividida em outras, medindo 64x64 pixels.

**Tabela 3.5 - Configuração do Outex\_TC\_00010**

	<b>Amostras 128 x 128 pixels</b>	<b>Amostras 64 x 64 pixels</b>
<b>Número de classes</b>	24	24
<b>Treino</b>	1 ângulo – $0^\circ$ ; 20 amostras por classe; Total de 480 amostras;	1 ângulo – $0^\circ$ ; 80 amostras por classe; Total de 1.920 amostras;
<b>Teste</b>	8 ângulos; $5^\circ, 10^\circ, 15^\circ, 30^\circ, 45^\circ, 60^\circ, 75^\circ, 90^\circ$ ; 20 amostras por ângulo e classe; (160 amostras por classe); Total de 3.840 amostras;	8 ângulos; $5^\circ, 10^\circ, 15^\circ, 30^\circ, 45^\circ, 60^\circ, 75^\circ, 90^\circ$ ; 80 amostras por ângulo e classe; (640 amostras por classe); Total de 15.360 amostras;

Já a suíte Outex\_TC\_00012 é um pouco mais elaborada que a anterior, sendo que a principal diferença está na inclusão duas novas condições de iluminação: a "TL84" e a "horizon". As mesmas classes, ângulos de rotação, tamanho e número de amostras por classe e angulo são mantidos nesta nova configuração. O conjunto de treinamento é o mesmo utilizado na Outex\_TC\_00010, mas agora existem dois conjuntos de teste, um composto pelas amostras rotacionadas capturadas sob a

iluminação "TL84" e outra com a "horizon". A Tabela 3.6 indica a composição dos testes para a Outex\_TC\_00012.

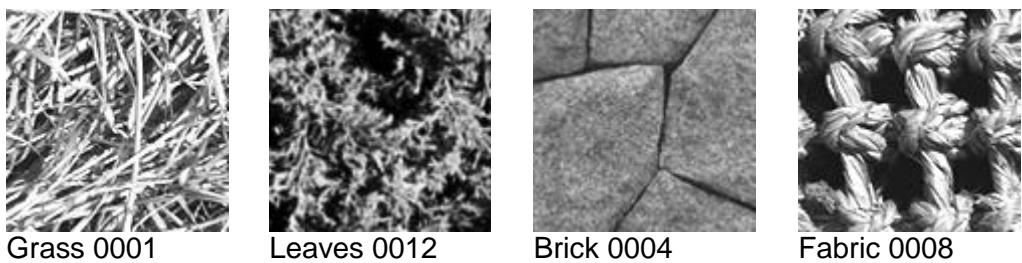
**Tabela 3.6 - Configuração do Outex\_TC\_00012**

	<b>Amostras 128 x 128 pixels</b>	<b>Amostras 64 x 64 pixels</b>
<b>Número de classes</b>	24	24
<b>Treino</b>	Iluminação "inca"; 1 ângulo – 0°; 20 amostras por classe; Total de 480 amostras;	Iluminação "inca"; 1 ângulo – 0°; 80 amostras por classe; Total de 480 amostras;
<b>Teste</b>	Iluminação "tl84" e "horizon"; 8 ângulos: 5°, 10°, 15°, 30°, 45°, 60°, 75°, 90°; 20 amostras por ângulo, classe e iluminação; (160 amostras por classe e iluminação); Total de 3.840 amostras por iluminação;	Iluminação "tl84" e "horizon"; 8 ângulos: 5°, 10°, 15°, 30°, 45°, 60°, 75°, 90°; 80 amostras por ângulo, classe e iluminação; (640 amostras por classe e iluminação); Total de 3.840 amostras por iluminação;

Assim, além de avaliar os descritores em um segundo banco de imagens, será possível verificar seu comportamento diante de diferentes condições de iluminação.

### **3.5.3. VisTex**

Diferentemente de outras coleções de textura, as condições de perspectivas e iluminação das imagens do VisTex não se restringem ao plano frontal e condições de iluminação de estúdio. O objetivo do VisTex foi o de fornecer imagens de textura representativos das condições do mundo real. Em particular, o conjunto foi criado como uma alternativa ao álbum de Brodatz. Foi criado e mantido pelo Media Lab do Massachusetts Institute of Technology até 2002, quando foi descontinuado. Alguns exemplos de texturas do VisTex são exibidos pela Figura 3.10:



**Figura 3.10 – Exemplos de texturas do VisTex**

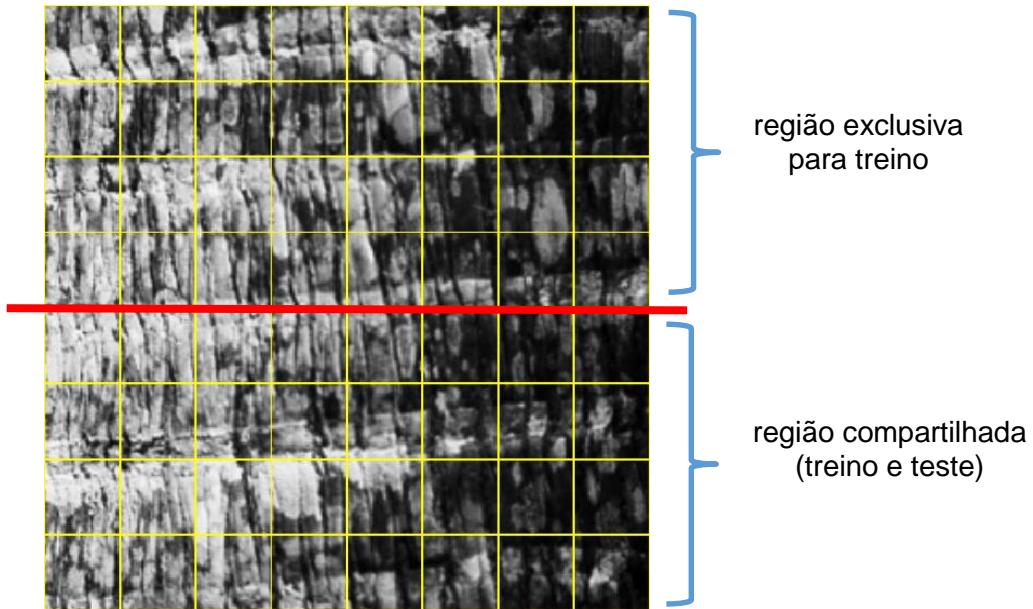
A Universidade de Oulu, disponibiliza no site do Outex a suíte de teste Contrib\_TC\_00006, que utiliza a base de imagens VisTex para o problema de classificação de texturas com amostras sem variação de orientação. Este teste é composto por 54 classes de texturas, sendo que cada uma delas é representada por uma imagem colorida, medindo 512x512 pixels. Cada imagem de textura foi dividida em 16 amostras medindo 128x128 pixels e metade delas foram selecionadas para o treino do descritor, enquanto as restantes serviram para o teste de classificação. Portanto, foram 432 amostras para treino e 432 para teste.

Apesar de disponibilizarem o teste, o grupo de pesquisa de Oulu não a utilizou em nenhuma publicação relacionada ao *LBP*, ainda assim, serviu de base para a configuração do experimento apresentado aqui, visto o interesse na utilização de um terceiro banco de texturas e de um teste, com conformação bem definida quanto à composição e amostragem das texturas.

O objetivo deste experimento é o de criar um novo teste baseado no Contrib\_TC\_00006 e verificar o comportamento do *Sampled LFP* aplicado a um terceiro banco de imagens, além de analisar o comportamento do descritor mediante a variação do número de amostras disponíveis para treinamento.

Neste experimento, as mesmas 54 imagens representativas das classes de texturas são utilizadas, só que agora elas foram convertidas para escala de cinza.

As amostras que poderão compor o conjunto de treinamento foram obtidas a partir da divisão de cada imagem original, não rotacionada, em 8 colunas e 8 linhas, gerando 64 amostras de 64x64 pixels cada. A Figura 3.11 mostra a divisão das amostras para treinamento da textura “Bark 0000”.



**Figura 3.11 – Amostras para treino da textura “Bark 0000”, não rotacionada**

Diferentemente do experimento original, agora as amostras de teste compreenderão amostras rotacionadas (sentido anti-horário) nos seguintes ângulos:  $5^\circ$ ,  $10^\circ$ ,  $15^\circ$ ,  $30^\circ$ ,  $45^\circ$ ,  $60^\circ$ ,  $75^\circ$ ,  $90^\circ$ . Para a geração dessas amostras (teste), somente a metade inferior da imagem original de cada textura foi considerada (Figura 3.11, região compartilhada). Assim, para cada textura, sua imagem original é rotacionada no ângulo desejado e 32 amostras de  $64 \times 64$  pixels são distribuídas, de modo a cobrir a metade selecionada para compor o conjunto de teste. No total, serão 13.824 amostras de teste (54 classes, 8 ângulos e 32 amostras por classe/ângulo). A Figura 3.12 mostra a textura “Bark 0000” rotacionada em  $10^\circ$  e as amostras criadas para esta classe e ângulo de textura.



**Figura 3.12 – Amostras para teste da Textura “Bark 0000”, rotacionada em 10°**

A ideia aqui é repetir o experimento, variando o número de amostras que compõem o conjunto de treino, podendo variar de 1 a 64 amostras por classe de textura. A seleção das amostras que compõem o conjunto de treino segue uma ordem, que pode ser dita **sequencia ordenada**, quando a seleção das amostras na imagem original (Figura 3.11) for feita a partir da esquerda para direita e de cima para baixo; ou **sequencia reversa**, quando seguir a ordem inversa, da direita para esquerda e de baixo para a cima.

Esta escolha visa proporcionar a análise do comportamento do descritor diante de amostras de imagens nunca analisadas – isso quando o conjunto de treino contiver 32 amostras por classe ou menos (região exclusiva para treino na imagem original) e de somente amostras já exploradas, quando o conjunto de treino usar a sequência reversa, com até 32 amostras por classe (região compartilhada na imagem original).

### **3.6. Considerações finais**

Este capítulo apresentou uma nova variante do *LFP-s*, o *Sampled Local Fuzzy Pattern (Sampled LFP)*, como um descritor de texturas baseado em números fuzzy e com suporte à análise de padrões em multiresolução. Foi também apresentado um processo de parametrização que visa a otimização de sua sensibilidade, além da

elaboração de uma metodologia para validação das capacidades do *LFP* para classificação de texturas.

Os experimentos foram implementados na ferramenta Matlab e o código fonte está registrado nos Apêndices A, B, C, D, E e F.

No próximo capítulo os resultados desses ensaios são apresentados e analisados.

## 4. RESULTADOS

Neste capítulo são apresentados os resultados dos experimentos executados a partir de imagens do álbum de Brodatz, Outex e VisTex. Os descritores baseados no *LFP* (*LFP-s*, *LFP-q* e *Sampled LFP*) são comparados com o descritor *LBP*.

### 4.1. Avaliação do *LFP* para classificação de texturas

Para cada suíte de teste do Outex, foi executado o processo de parametrização dos valores de  $\beta$  e do número de *bins*. Como indicado na metodologia, esse processo envolve a sintonização do descritor para determinação dos melhores valores de  $\beta$  e do número de *bins*.

A métrica utilizada para a classificação das amostras foi a distância chi-quadrado. Como exemplo do processo de parametrização de  $\beta$ , as Figuras 4.1 e 4.2 apresentam a relação entre o valor de  $\beta$  e a sensibilidade obtida, calculada com o uso do classificador chi-quadrado, para as suítes de teste TC\_00000 e TC\_00010.

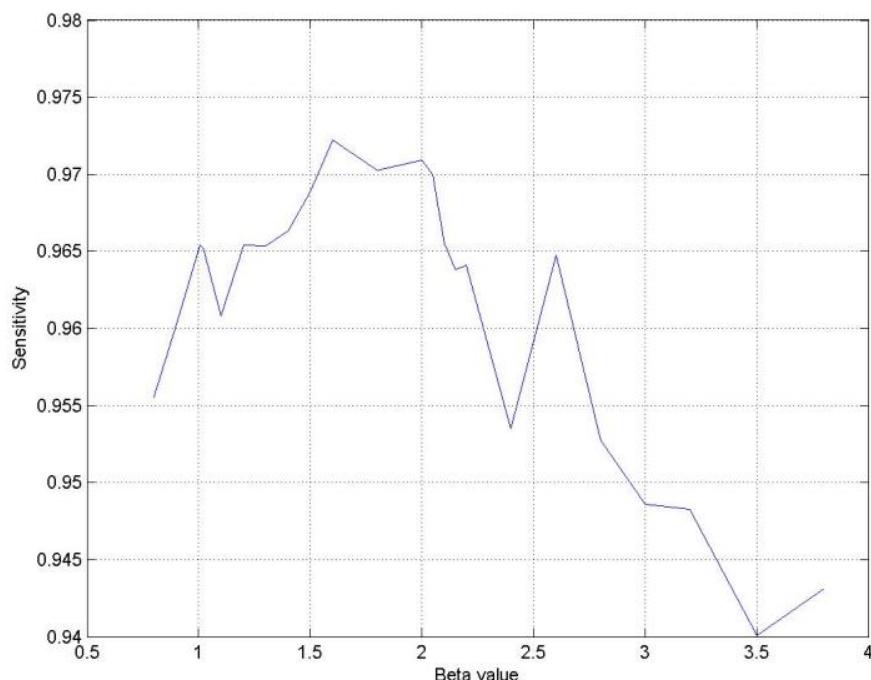
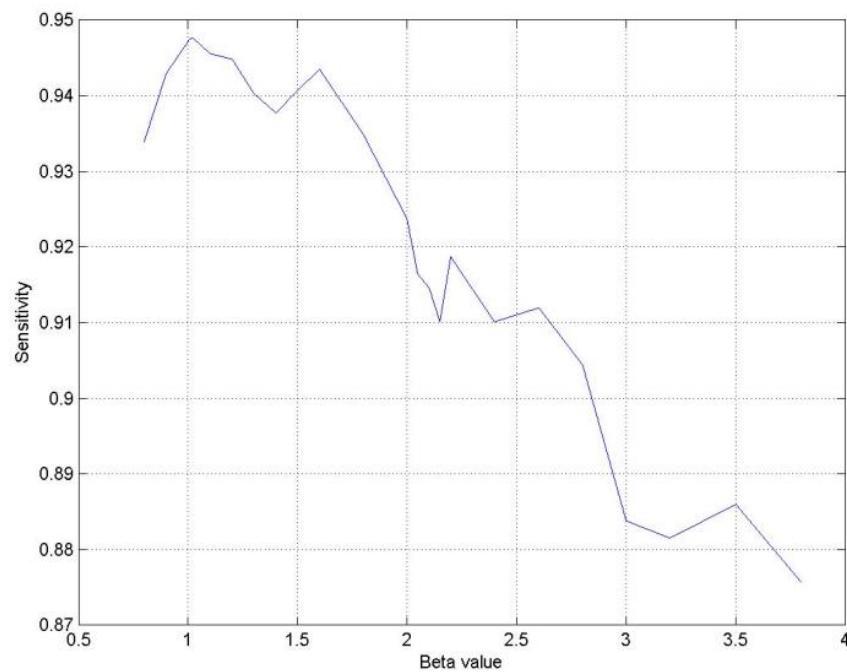
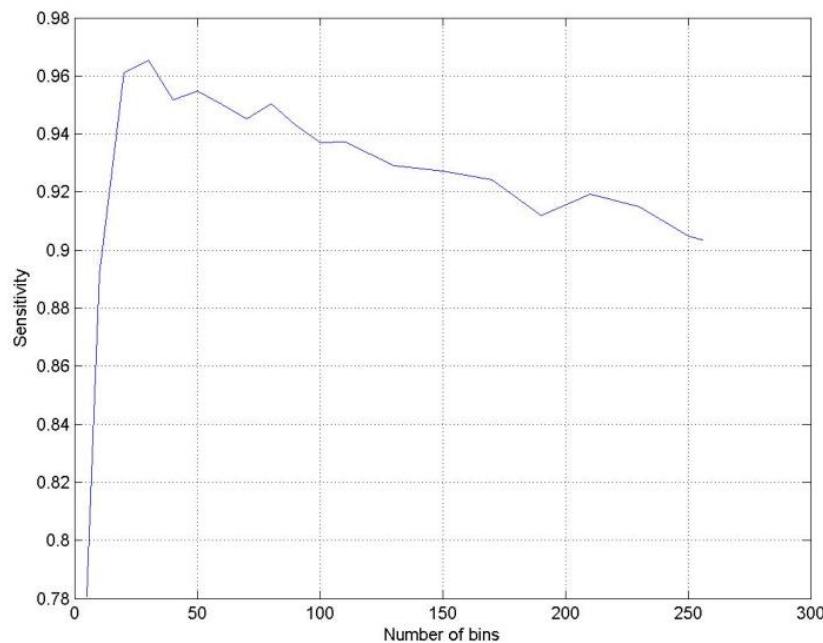


Figura 4.1 - Sensibilidade versus valor de  $\beta$  para a suíte Outex\_TC\_00000

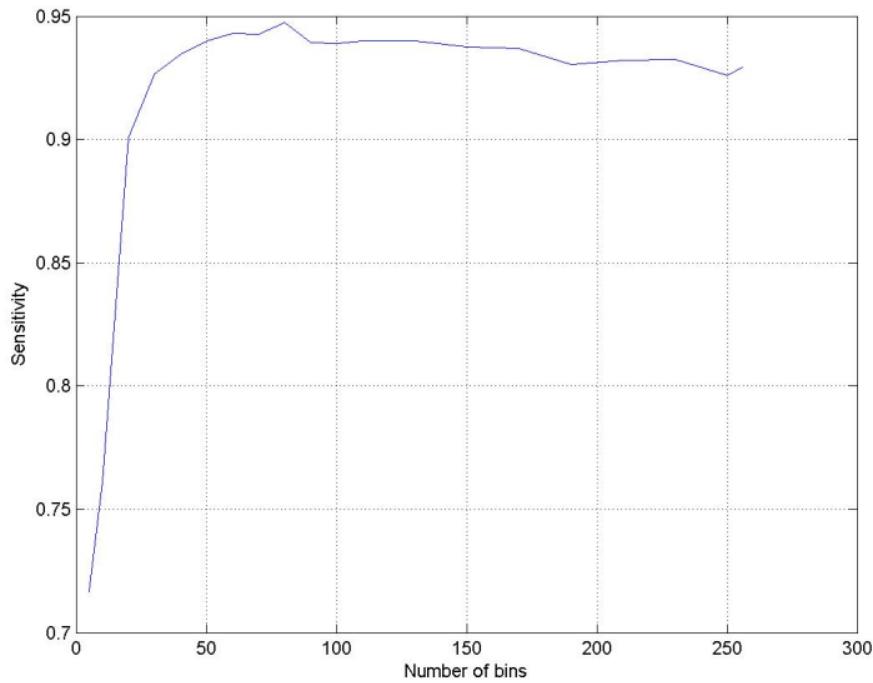


**Figura 4.2 - Sensibilidade versus valor de  $\beta$  para a suíte Outex\_TC\_00010**

Já a parametrização do número de *bins* é exemplificada pelas Figuras 4.3 e 4.4, onde a relação entre o número de *bins* e sensibilidade para as suítes TC00000 e TC00010 são apresentadas.



**Figura 4.3 - Sensibilidade versus valor do número de *bins* para a suíte TC\_00000, com  $\beta = 1,6$**



**Figura 4.4 - Sensibilidade versus valor do número de *bins* para a suíte TC\_00010, com  $\beta = 1,02$**

Após a conclusão do processo de parametrização de  $\beta$  e do número de *bins* para cada suíte de teste, as seguintes sensibilidades foram apuradas (Tabela 4.1):

**Tabela 4.1 - Melhor configuração do *LFP* para cada suíte de teste**

Suíte de teste	Número de <i>bins</i>	Beta	Sensibilidade
TC_00000	20	1,6	97,29
TC_00001	80	1,02	91,77
TC_00002	80	0,9	67,69
TC_00003	30	1,6	97,18
TC_00004	60	1,005	91,42
TC_00005	80	0,9	67,90
TC_00006	60	1,005	90,46
TC_00007	80	0,9	74,75
TC_00008	60	1,005	90,83
TC_00009	60	1,005	72,82
TC_00010	80	1,02	94,77
TC_00011	80	1,5	88,54
TC_00012	20	0,8	68,17
TC_00015	30	1,4	70,99

Como observado, existe um  $\beta$  e número de *bins* ótimos para cada suíte de teste, algo esperado visto que a composição do banco de imagens é alterada para cada suíte de teste.

A Tabela 4.2 apresenta a comparação entre as sensibilidades obtidas pelo *LFP-s* com aquelas obtidas pelo *LBP* melhor indicado para cada tipo de problema (*LBP* clássico para classificação de amostras sem variação de orientação e o *LBP riu2* para amostras rotacionadas).

**Tabela 4.2 - Comparação entre o *LFP-s* e o *LBP* melhor indicado**

Suíte de teste	Sensibilidade		Melhor descriptor	<i>LFP-s - LBP</i>
	<i>LBP</i>	<i>LFP-s</i>		
TC_00000	99,61	97,29	LBP	-2,32
TC_00001	98,44	91,77	LBP	-6,67
TC_00002	85,67	67,69	LBP	-17,98
TC_00003	99,58	97,18	LBP	-2,40
TC_00004	98,48	91,42	LBP	-7,06
TC_00005	85,77	67,9	LBP	-17,87
TC_00006	98,47	90,46	LBP	-8,01
TC_00007	89,17	74,75	LBP	-14,42
TC_00008	98,57	90,83	LBP	-7,74
TC_00009	89,62	72,82	LBP	-16,80
TC_00010	84,82	94,77	<i>LFP-s</i>	9,95
TC_00011	93,33	88,54	LBP	-4,79
TC_00012	64,57	68,17	<i>LFP-s</i>	3,60
TC_00015	78,28	70,99	LBP	-7,29

De acordo com esses resultados, somente em duas suítes de teste o *LFP-s* apresenta um resultado superior ao *LBP*: a TC\_00010 e a TC\_00012, justamente as duas únicas suítes de teste que trabalham com amostras rotacionadas. Em todas as outras 14 suítes de teste, que trabalham com amostras sem rotação, o *LBP* clássico se mostrou como opção mais interessante como descritor de texturas.

Porém, como indicado anteriormente, numa aplicação do mundo real não se sabe se a amostra a ser classificada foi ou não rotacionada. É necessário que o descritor escolhido seja capaz de trabalhar com imagens rotacionadas ou não. Assim, tanto o *LFP-s*, quanto o *LBP riu2* foram posicionados como descritores genéricos, invariantes à rotação e aplicáveis a qualquer tipo de situação, tendo seus resultados comparados conforme indica a Tabela 4.3:

**Tabela 4.3 - Comparação entre o *LFP-s* e o *LBP riu2***

Suíte de teste	Sensibilidade		Melhor descriptor	<i>LFP-s – LBP riu2</i>
	<i>LBP riu2</i>	<i>LFP-s</i>		
TC_00000	95,01	97,29	<i>LFP-s</i>	2,28
TC_00001	83,13	91,77	<i>LFP-s</i>	8,65
TC_00002	55,87	67,69	<i>LFP-s</i>	11,83
TC_00003	94,95	97,18	<i>LFP-s</i>	2,22
TC_00004	82,81	91,42	<i>LFP-s</i>	8,61
TC_00005	55,38	67,90	<i>LFP-s</i>	12,52
TC_00006	83,33	90,45	<i>LFP-s</i>	7,13
TC_00007	63,76	74,75	<i>LFP-s</i>	11,00
TC_00008	82,56	90,83	<i>LFP-s</i>	8,26
TC_00009	64,33	72,82	<i>LFP-s</i>	8,50
TC_00010	84,82	94,77	<i>LFP-s</i>	9,95
TC_00011	77,50	88,54	<i>LFP-s</i>	11,04
TC_00012	64,57	68,17	<i>LFP-s</i>	3,60
TC_00015	71,29	70,99	<i>LBP riu2</i>	-0,31

Vê-se que o *LFP-s* apresenta melhor sensibilidade que o *LBP riu2* em quase todos os casos de teste, somente tendo uma sensibilidade inferior na última suíte de teste, a TC\_00015.

Algumas informações podem ser obtidas a partir dos resultados onde o *LFP-s* não obteve uma margem maior de vantagem sobre o *LBP riu2*. Primeiro, o *LFP-s* é mais sensível ao número de amostras utilizadas para treinamento. Isto explica os resultados em TC\_00000, TC\_00003, já que essas duas suítes são as que possuem o menor número de amostras para treino (240 amostras). Segundo, o *LFP-s* sofre tanto quanto o *LBP riu2* para a classificação de texturas capturas sob diferentes condições de iluminação (TC\_00012 e TC\_00015).

Como crítica aos últimos testes executados, é apontado o uso do *LFP-s* otimizado para cada teste. Dado o critério adotado que motivou a aplicação do *LBP riu2* para todos os testes, o mesmo critério também deveria ser aplicado ao *LFP-s*, ou seja, deveria existir um valor de  $\beta$  e número de *bins* comuns a todos os testes.

Tomando os melhores valores de  $\beta$  e número de *bins*, foram calculadas várias médias para de  $\beta$  e número de *bins*, como demonstrado pela Tabela 4.4.

**Tabela 4.4 - Médias dos melhores parâmetros para o *LFP-s***

<b>Tipo de média</b>	<b>Número de bins</b>	<b>Beta</b>
Mediana	60	1,00500
Aritmética	59	1,11857
Geométrica	53	1,08967
Harmônica	45	1,06422

A seguir, o processo de classificação foi executado para todas as suítes de teste, aplicando os pares de parâmetros obtidos acima. Os resultados dessa classificação são apresentados pela Tabela 4.5.

**Tabela 4.5 - Sensibilidade do *LFP-s* parametrizado com  $\beta$  e número de *bins* médios**

<b>Parâmetros do <i>LFP-s</i></b>	<b>Mediana</b>	<b>Aritmética</b>	<b>Harmônica</b>	<b>Geométrica</b>
Beta	1,005	1,11857	1,08967	1,06422
Número de <i>bins</i>	60	59	53	45
<b>Suíte de teste</b>	<b>Sensibilidades</b>			
TC_00000	95,00	95,18	95,40	95,82
TC_00001	91,59	91,23	89,89	92,43
TC_00002	66,19	65,38	64,17	66,88
TC_00003	94,63	94,65	94,77	95,34
TC_00004	91,42	91,15	89,99	92,43
TC_00005	66,01	65,58	64,59	67,01
TC_00006	90,45	90,08	89,40	91,46
TC_00007	73,03	72,67	71,47	73,58
TC_00008	90,83	90,29	89,61	91,70
TC_00009	72,82	71,81	71,50	73,08
TC_00010	94,30	94,17	93,20	94,48
TC_00011	86,25	86,46	87,29	87,92
TC_00012	53,30	55,89	55,53	56,96
TC_00015	68,28	68,41	68,38	69,59

A diferença entre a sensibilidade obtida pelo *LFP-s*, calculado e classificado com os parâmetros médios, e o *LBP riu2* é dada pela Tabela 4.6:

**Tabela 4.6 - Diferença entre as sensibilidades obtidas pelo *LFP-s* calculado com parâmetros médios e o *LBP riu2***

Parâmetros do <i>LFP-s</i>	Mediana	Aritmética	Harmônica	Geométrica
Beta	1,005	1,11857	1,08967	1,06422
Número de bins	60	59	53	45
Suíte de teste	Sensibilidades			
TC_00000	-0,01	0,17	0,39	0,81
TC_00001	8,47	8,11	6,76	9,31
TC_00002	10,32	9,52	8,31	11,01
TC_00003	-0,32	-0,30	-0,18	0,39
TC_00004	8,61	8,33	7,18	9,62
TC_00005	10,64	10,21	9,21	11,63
TC_00006	7,13	6,75	6,07	8,13
TC_00007	9,28	8,91	7,71	9,83
TC_00008	8,26	7,73	7,05	9,14
TC_00009	8,50	7,48	7,17	8,75
TC_00010	9,48	9,35	8,39	9,66
TC_00011	8,75	8,96	9,79	10,42
TC_00012	-11,27	-8,68	-9,04	-7,62
TC_00015	-3,01	-2,88	-2,91	-1,71

Verifica-se que o *LFP-s* apresenta melhor sensibilidade que o *LBP riu2* em 12 das 14 suítes de teste, quando utilizada a média geométrica. Porém, o *LFP-s* apresenta um desempenho inferior ao *LBP riu2* quando a iluminação das amostras de treino e teste for diferente, como foi o caso para as suítes 12 e 15, as únicas que trabalham com uma base de imagens capturadas sob três condições diferentes de iluminação.

#### 4.1.1. Considerações

Analizando o processo de parametrização, é visto que a variável  $\beta$  é a maior responsável pela sensibilidade obtida pelo descritor. Ainda assim, a otimização do número de *bins* exerce um papel de “sintonia fina” do *LFP-s*, melhorando, ainda que discretamente, a sensibilidade original.

A comparação entre o *LFP-s* e o *LBP* melhor indicado ao caso apontou para o *LFP* como um bom descritor invariante à rotação, mas não para o caso onde as

amostras de treino têm a mesma orientação das amostras de teste (amostras sem variação de orientação).

Apesar de ser desejável que um operador seja invariante à rotação, ainda podem existir aplicações que exijam amostras que não estejam rotacionadas. Portanto, para essas aplicações, é sugerido o uso do *LBP* clássico. Porém, para a condição geral, onde a orientação das amostras pode ser alterada, o *Local Fuzzy Pattern* é o descritor indicado, visto que gera os melhores resultados.

#### **4.2. Classificação de texturas invariante à rotação e multiresolução**

A análise do *Sampled LFP* como um método para classificação de texturas invariante à rotação e com suporte a análise de padrões em multiresolução é feita a partir do exame dos vários resultados apresentados a seguir.

##### **4.2.1. Álbum de Brodatz**

A parametrização dos descritores *Sampled LFP* e *LFP-q* foi feita de forma individual e em suas várias resoluções (Tabela 4.7). A Tabela 4.8 exibe a sensibilidade obtida pelos vários descritores comparados.

**Tabela 4.7 - Valores de  $\beta$  para o experimento I de Brodatz**

Operador	Resolução	Beta	Núm. bins
Sampled LFP	8x1	0,6	150
Sampled LFP	16x2	0,6	160
Sampled LFP	24x3	2,5	190
LFP-q	$W=3$	1.3	256
LFP-q	$W=5$	1.4	256

É observado que os valores de  $\beta$  e número de bins variam conforme a resolução analisada. Isso indica que o padrão muda conforme a resolução aumenta e, portanto, o processo de parametrização deve ser executado especificamente para

cada resolução. Isso gera um custo computacional extra, mas também pode customizar o descritor para determinada tarefa de classificação.

**Tabela 4.8 - Sensibilidade obtida no experimento I de Brodatz**

Operador	Resolução		
	8x1	16x2	24x3
LBP riu2	88,54	96,66	99,26
Sampled LFP	92,71	99,85	98,21
LFP-q	91,52	88,99	-

Analizando a sensibilidade obtida pelos descritores, é notado que o *Sampled LFP* é superior ao *LBP riu2* até a resolução de 16x2 pixels, apresentando degradação na resolução de 24x3 pixels. Esse fato pode ser atribuído ao tamanho das amostras de treino, muito pequenas (16x16 pixels) para um descritor com R = 3.

Comparando-se o *Sampled LFP* e o *LFP-q*, nota-se uma sensibilidade muito parecida quando ambos trabalharam em suas resoluções menores, porém, o *LFP-q* teve sua sensibilidade reduzida ao aumentar sua resolução.

O experimento II de Brodatz permite verificar como se comporta a classificação de amostras rotacionadas em determinados ângulos quando comparadas ao ângulo de treino base ( $11^\circ$ ). A Tabela 4.9 apresenta a sensibilidade obtida na análise de cada ângulo, quando executadas por cada descritor em sua menor resolução disponível.

**Tabela 4.9 - Sensibilidade por ângulo – experimento II de Brodatz**

Operador	Ângulo de treino						Média
	30	60	90	120	150	200	
LBP riu2 P = 8, R = 1	50,10	65,73	70,42	50,52	67,08	59,48	60,56
Sampled LFP P = 8, R = 1	52,08	68,96	81,04	53,65	68,44	62,50	64,44
LFP-q W = 3	55,00	64,90	81,88	56,88	65,63	63,44	64,62

O *Sampled LFP* obteve melhor sensibilidade do que o *LBP riu2* para qualquer ângulo de teste, porém, assim como o *LFP-q* e o próprio *LBP riu2*, sofreu com os “ângulos interpolados”. Apesar do *LFP-q* apresentar uma sensibilidade similar ao *Sampled LFP* na resolução mínima, ao incrementá-la foi observado novamente uma degradação em sua performance, conforme mostra a Tabela 4.10.

**Tabela 4.10 - Sensibilidade por resolução de amostragem – experimento II de Brodatz**

Operador	Sensibilidade média		
	8x1	16x2	24x3
LBP riu2	60,56	70,50	66,46
Sampled LFP	64,44	66,08	57,52
LFP-q	64,62	57,43	-

O experimento II apresenta uma drástica redução no número de amostras utilizadas na fase de treinamento. Enquanto o experimento I utiliza 484 amostras por classe, o experimento II utiliza apenas 64 imagens. Essa redução impacta mais o *Sampled LFP* conforme sua resolução de sua vizinhança de análise aumenta e, somado ao pequeno tamanho das amostras (32x32), debilita ainda mais o poder de caracterização do descritor quando utilizado na resolução de 24x3 pixels. Já o *LFP-q* novamente apresentou sensibilidade pior que os outros descritores para resolução maior que 8x1 pixels (Tabela 4.10).

**Tabela 4.11 - Sensibilidade por resolução de amostragem – experimento III de Brodatz**

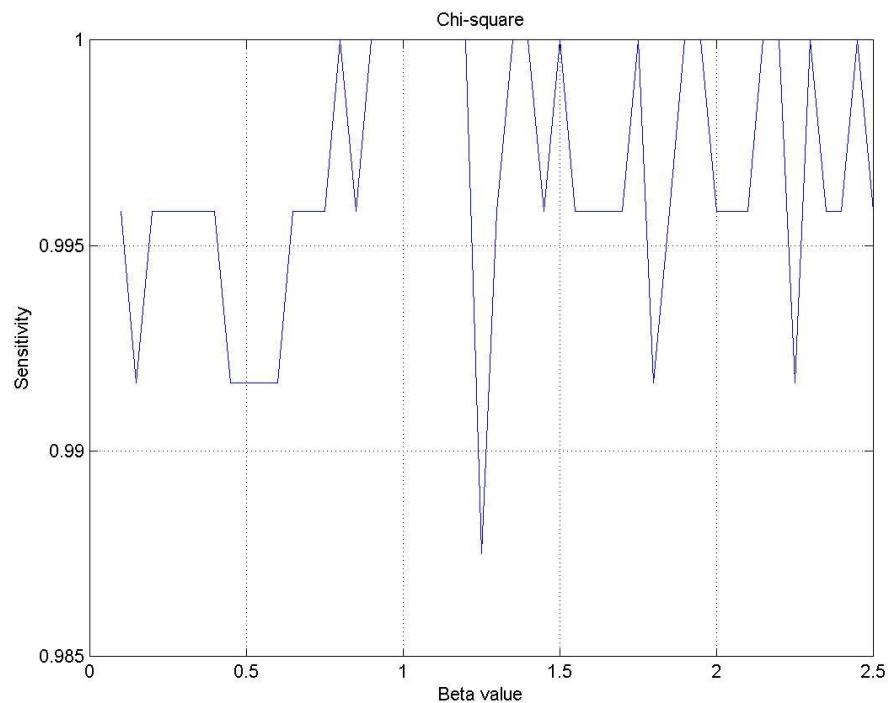
Operador	Sensibilidade média		
	8x1	16x2	24x3
LBP riu2	78,68	90,49	90,63
Sampled LFP	81,18	82,36	78,26
LFP-q	76,94	80,69	-

No experimento III, o tamanho das amostras foi aumentado para 64x64 pixels, porém o número de amostras por classe utilizadas para treinamento foi novamente reduzido, passando agora para apenas 16 amostras.

Quando se compara os resultados do experimento II com aqueles do experimento III (tabelas 4.10 e 4.11), nota-se que amostras de tamanho 32x32 pixels, utilizadas pelo experimento II, não contribuem para uma boa classificação.

Apesar do *Sampled LFP* ter apresentado melhor sensibilidade que os outros descritores para a resolução de 8x1 pixels no experimento III, o incremento de sua performance é marginal quando houve aumento de sua resolução para 16x2 pixels. Conforme visto nos gráficos da parametrização de  $\beta$  para o descritor *Sampled LFP* na resolução 8x1 (Figura 4.5), este descritor sofreu com o pouco número de amostras

utilizadas no treino. Com um número tão baixo de amostras de treino, sua avaliação estatística ficou comprometida.

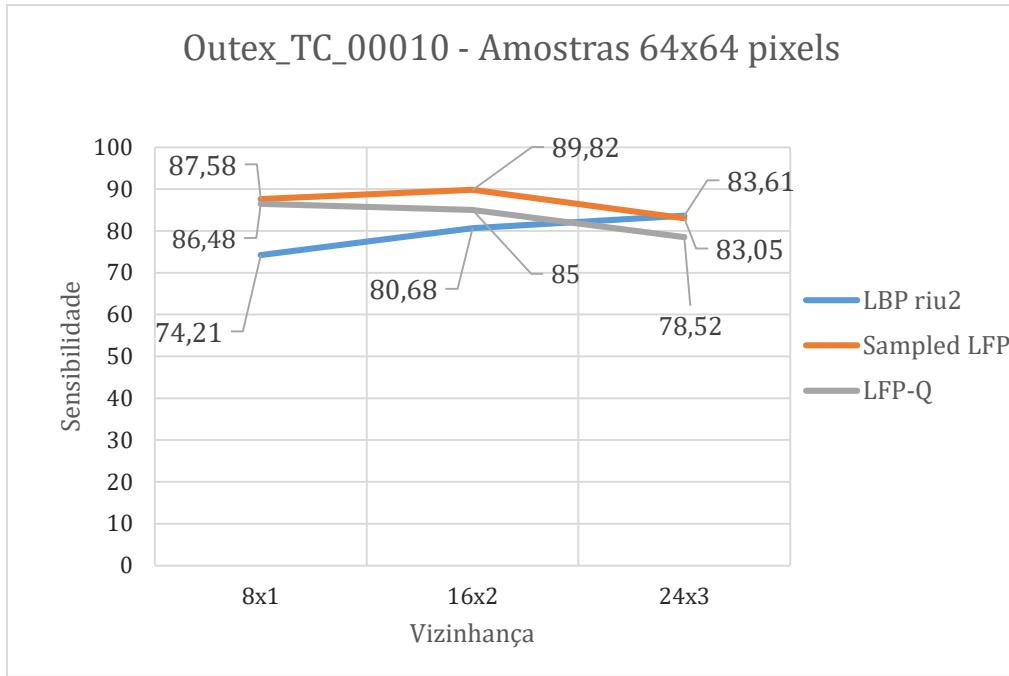


**Figura 4.5 - Melhor  $\beta$  para o experimento III de Brodatz**

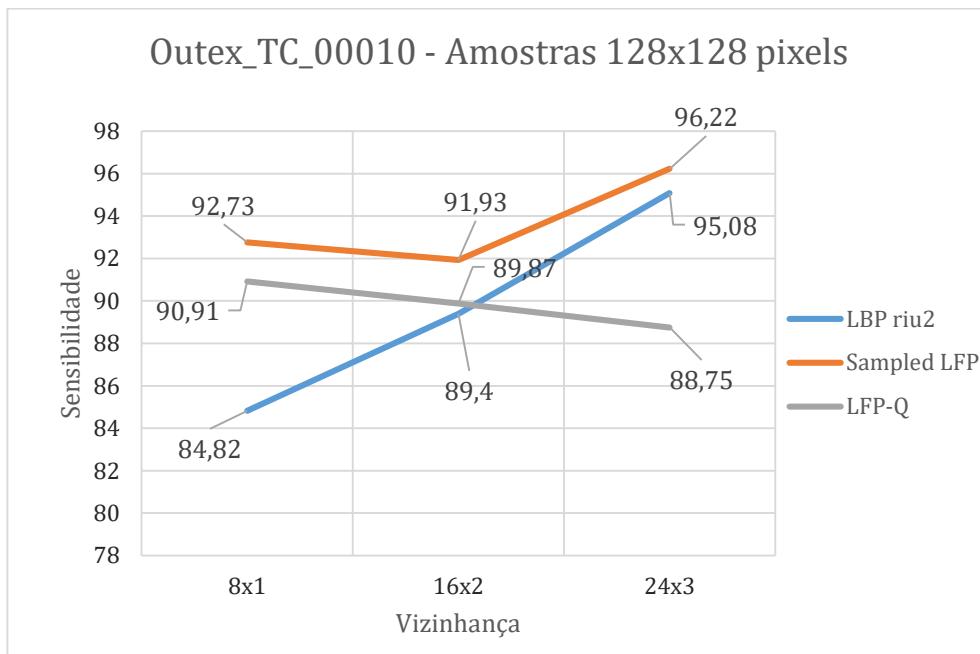
O *LFP-q* novamente obteve a menor sensibilidade em resolução de 16x2 pixels, evidenciando sua inaptidão para processamento multiresolução (janela de análise de tamanho variável).

#### 4.2.2. Outex

Os resultados para o experimento do Outex\_TC\_00010 são apresentados pela Figura 4.6 (amostras 64x64 pixels) e Figura 4.7 (amostras 128x128 pixels) e mostraram que, independentemente do tamanho das amostras, a sensibilidade do *Sampled LFP* foi superior aos outros descritores.



**Figura 4.6 - Sensibilidade no Outex\_TC\_00010, para amostras de 64x64 pixels**



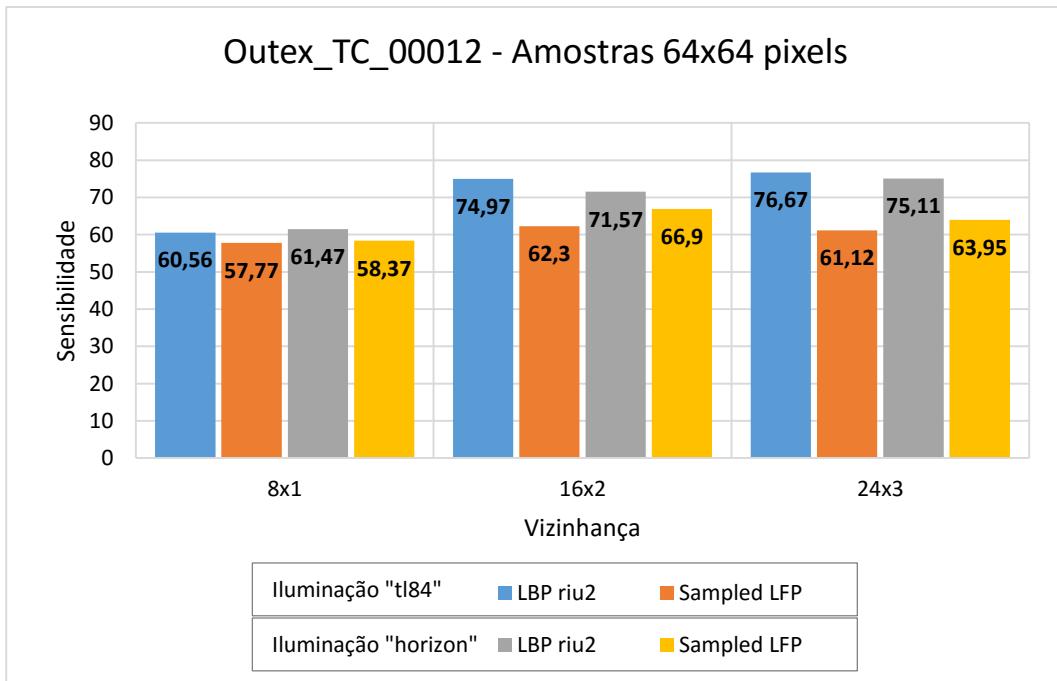
**Figura 4.7 - Sensibilidade no Outex\_TC\_00010, para amostras de 128x128 pixels**

No caso de vizinhança 8x1, a diferença de desempenho entre o *Sampled LFP* e o *LBP* foi mais de 12%, para amostras 64x64 pixels e de cerca de 6% para amostras de tamanho 128x218 pixels.

Porém, apesar do *LBP* ter sido superado nos testes com os dois tamanhos de amostras, ele apresentou uma tendência ascendente para sua sensibilidade,

conforme o tamanho da janela de análise se ampliava, fato já observado nos experimentos com o álbum de Brodatz. Já o *Sampled LFP*, para amostras 64x64 pixel, apresentou um pequeno incremento em sua sensibilidade quando trabalhando com uma janela 16x2 e uma queda de rendimento quando trabalhando com uma janela 24x3, comportamento também observado nos experimentos anteriores. Quando o tamanho de amostras foi aumentado para 128x128 pixels, a curva de tendência do *Sampled LFP* se alterou, fazendo com que o descritor obtivesse seu melhor desempenho quando trabalhando com uma janela 24x3 pixels, fato que corrobora com as observações já feitas sobre os experimentos anteriores, que indica que o *Sampled LFP* é mais sensível ao tamanho da amostra do que o *LBP*.

Como em todos os testes realizados, o *LFP-q* apresentou sensibilidade inferior ao *Sampled LFP* e tendência decrescente conforme o aumento do tamanho da janela de análise, sendo assim considerado como um descritor não apto à tarefa de classificação de texturas em multiresolução.

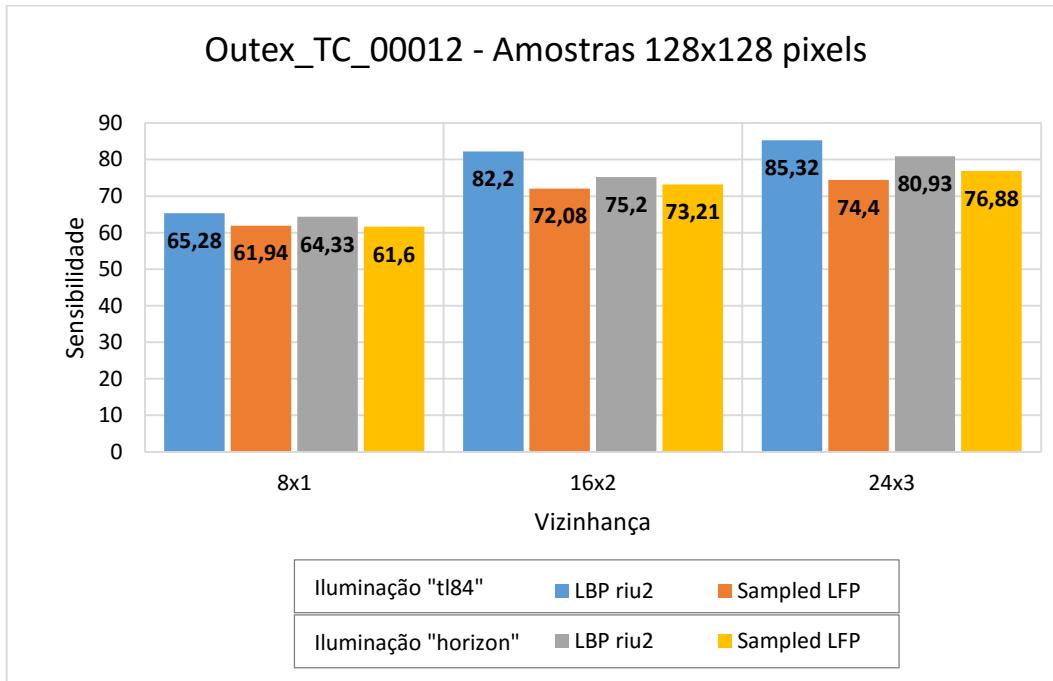


**Figura 4.8 - Sensibilidade no Outex\_TC\_00012, para amostras de 64x64 pixels**

Quando os operadores de textura são submetidos ao experimento Outex\_TC\_00012 (Figura 4.8 e Figura 4.9), com variação nas condições de iluminação, é verificado que o *Sampled LFP* apresenta pior sensibilidade que o *LBP* para todos tamanhos de janela e amostra.

Essa dificuldade na classificação de amostras capturadas em diferentes condições de iluminação é devido à forma que o código do *Sampled LFP* é montado. Ao contrário do *LBP*, onde o valor da diferença de nível de cinza entre um pixel da vizinhança e o pixel central não é importante (somente importa se o nível de cinza do pixel da vizinhança é maior ou igual ao do pixel central), para o *Sampled LFP* essa diferença tem significado, visto que o *Sampled LFP* utiliza a função sigmoide para seu cálculo.

Apesar da deterioração de sua performance, o *Sampled LFP* ainda manteve o mesmo padrão para incremento de sensibilidade, confirmando que o *Sampled LFP* é mais sensível à variação do tamanho das amostras e que amostras maiores possibilitam melhor desempenho.



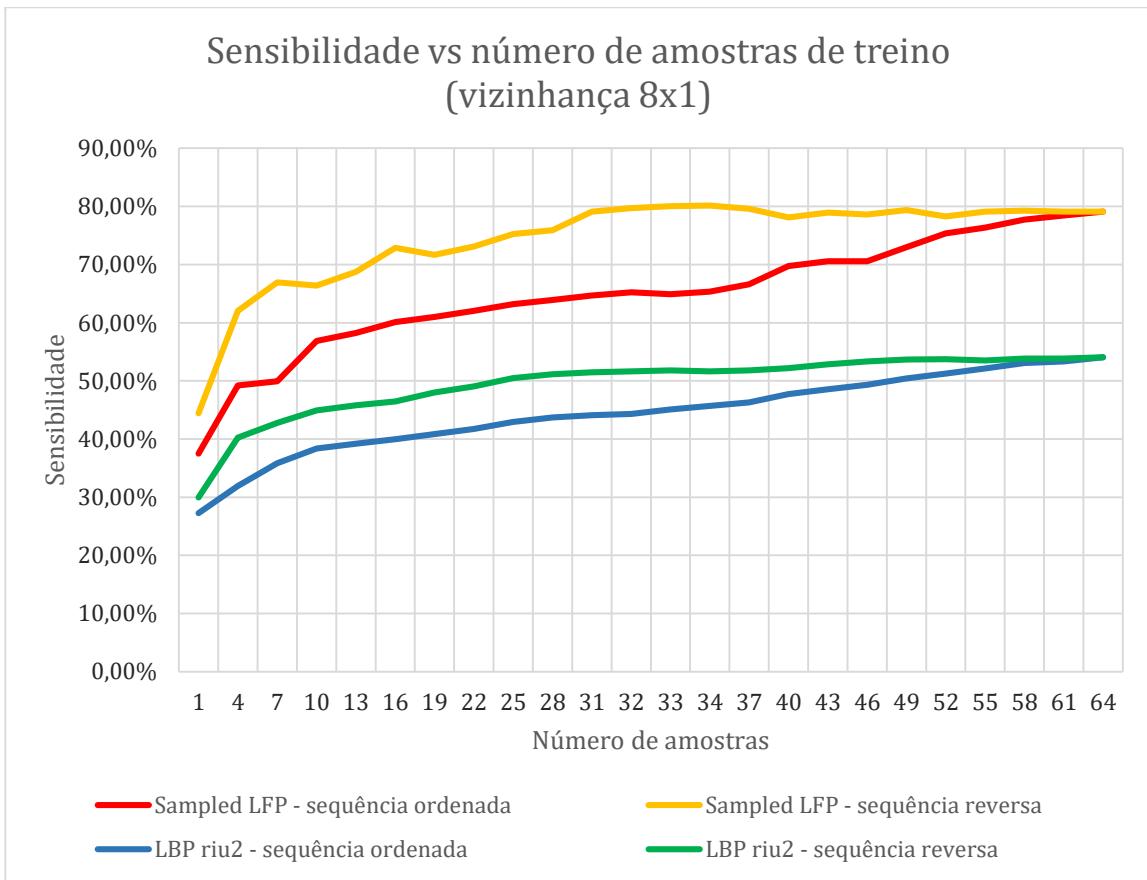
**Figura 4.9 - Sensibilidade no Outex\_TC\_00012, para amostras de 128x128 pixels**

#### 4.2.3. VisTex

De todos os experimentos, o realizado com a base de imagens VisTex é o que trabalha com o maior número de classes (54 classes), tornando o problema de classificação mais difícil. Este ensaio também permite a análise da sensibilidade obtida pelos descritores de textura quando treinados com um conjunto de amostras

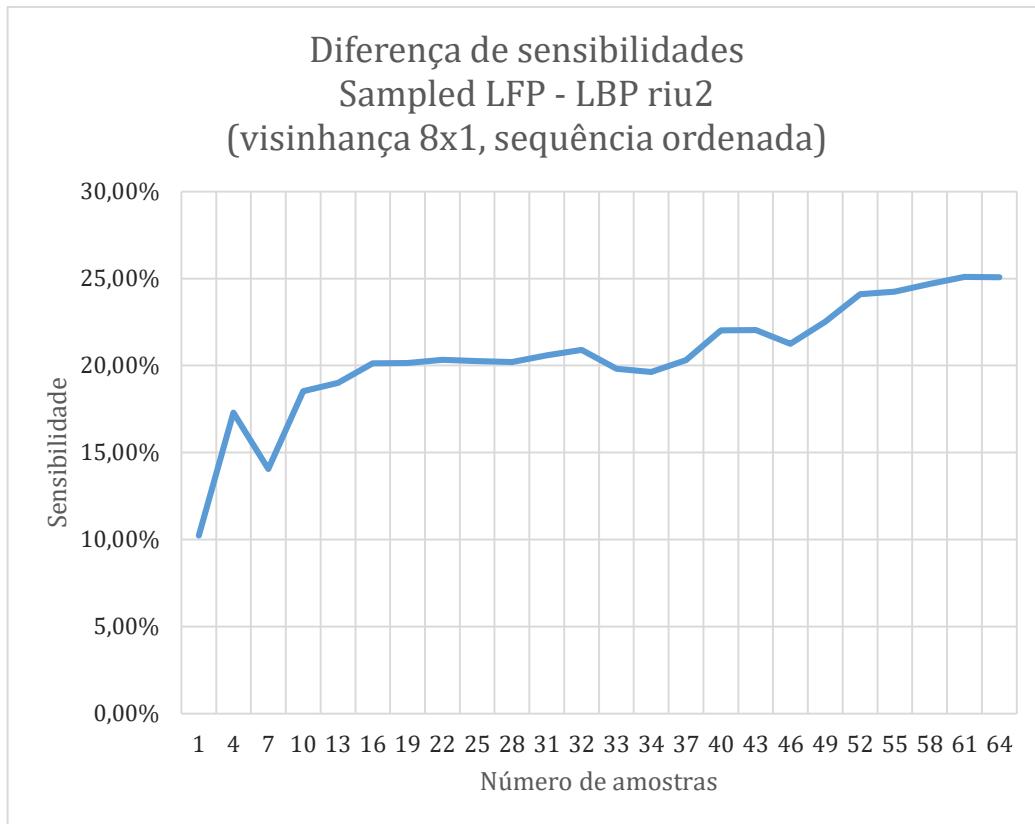
que, de fato, não serão também testadas a posteriori, algo que acontece com os experimentos baseados no álbum de Brodatz e no Outex, quando as amostras de treino e teste compreendem a mesma área de uma imagem.

As sensibilidades obtidas pelos operadores *Sampled LFP* utilizando uma janela de análise 8x1 são apresentadas pela Figura 4.10. Os parâmetros do Sampled LFP obtidos pelo processo de sintonia, mais sua sensibilidade nos testes, é apresentada no Apêndice H, já a sensibilidade obtida pelo LBP é apresentada pelo Apêndice I.



**Figura 4.10 - Sensibilidade conforme número de amostras de treinamento, para imagens do VisTex, usando amostras de 64x64 pixels e vizinhança 8x1**

Os resultados mostram que o *Sampled LFP* obteve um excelente desempenho, apresentando um ganho de sensibilidade médio de 20% sobre o *LBP riu2*. Também é notado que a sensibilidade obtida pelos dois descritores tem relação com o número de amostras que compõem o conjunto de teste – quanto mais amostras para treino, melhor a sensibilidade obtida pelo descritor, sendo que o *Sampled LFP* se beneficia mais da adição de amostras ao conjunto de treino (Figura 4.11).



**Figura 4.11 - Diferença de sensibilidade dos descritores (Sampled LFP - LBP riu2) conforme número de amostras de treinamento para imagens do VisTex, usando amostras 64x64 pixels e vizinhança 8x1**

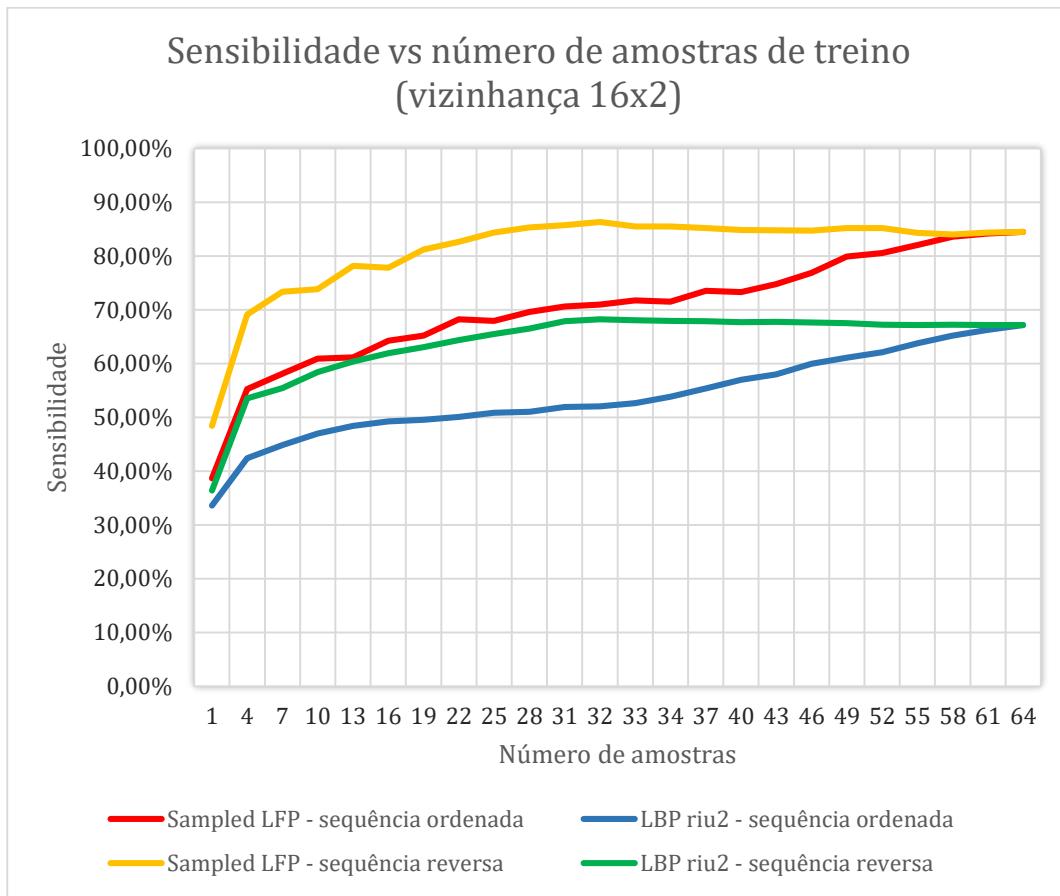
É importante notar as diferenças de sensibilidade quando o treinamento é realizado com amostras compartilhadas ou não pelos processos de treinamento e teste. Descritores treinados com amostras compartilhadas (Figura 4.10, sequencia reversa, até amostra número 32) apresentam uma curva de sensibilidade mais íngreme do que quando treinados com amostras exclusivas para treino (Figura 4.10, sequencia ordenada, até amostra número 32).

Quando uma mesma região de uma imagem é utilizada para a geração de amostras de treino (amostras não rotacionadas) e teste (amostras rotacionadas), é esperado que exista uma composição (níveis de cinza) muito similar entre as amostras de mesma região, mesmo com o processo de interpolação dos níveis de cinza que ocorre na rotação artificial (por computador) da imagem. Assim, experimentos que utilizam as mesmas imagens para obtenção de amostras de treino e teste sempre obterão melhores resultados do que aqueles que utilizam imagens diferentes.

Infelizmente os bancos de imagens disponíveis para pesquisa, como o álbum de Brodatz, Outex e VisTex, não disponibilizam diversas imagens para cada textura, tornando assim os resultados dos experimentos conduzidos com essas imagens

superestimados. Como exemplo, quando o descritor *Sampled LFP* foi treinado com as 32 amostras por classe de textura exclusivas para treino, a sensibilidade resultante foi de 65,21%. Já quando treinado com as 32 amostras obtidas da região da imagem compartilhada pelo treino e teste, obteve 79,72%.

Analizando o comportamento dos descritores quando utilizando uma janela de análise 16x2 (Figura 4.12), é observado que o *Sampled LFP* novamente obteve um excelente desempenho, com uma sensibilidade de 71,01% contra 52,07% do *LBP riu2*, quando treinados com 32 amostras (sequência ordenada).



**Figura 4.12 - Sensibilidade conforme número de amostras de treinamento, para imagens do VisTex, usando amostras de 64x64 pixels e vizinhança 16x2**

Tal qual nos resultados obtidos quando utilizando uma janela de análise 8x1, os descritores se beneficiaram do incremento no número de amostras utilizadas na composição do conjunto de teste. Quando treinados seguindo a sequência ordenada, ambos descritores apresentaram uma taxa de crescimento linear de sua sensibilidade, já quando o treinamento seguia a sequência reversa, apresentavam inicialmente uma curva de sensibilidade mais íngreme, se estabilizando a partir da 32<sup>a</sup>. amostra,

comportamento explicado pelo fato de que as primeiras 32 amostras de treinamento, na sequencia reversa, representarem a mesma região da imagem analisada na etapa de teste.

#### **4.3. Considerações finais**

Nos experimentos conduzidos com o álbum de Brodatz, o tamanho das amostras foi aumentado, indo de 16x16 a 64x64 pixels, mas com o conjunto de treinamento sendo reduzido de 484 para 16 amostras. Para essas configurações, a sensibilidade apresentada pelo descritor *Sampled LFP* mostrou-se superior àquelas obtidas pelos outros descritores, quando definida a resolução da janela de análise em 8x1 pixels. Conforme a resolução da janela de análise aumentou, os descritores baseados em números fuzzy apresentaram uma pequena diminuição de sensibilidade. Esses resultados sugeriram que, para resoluções maiores que 8x1 pixels, o *Sampled LFP* necessita de mais amostras para treinamento e que elas tenham maior área para sua análise (amostras maiores que 32x32 pixels).

A indicação de que o *Sampled LFP* obtém melhores resultados quando treinado com mais amostras e de tamanhos maiores ou iguais a 64x64 pixels é corroborada pelos resultados do experimento com o Outex, que também mostra que o *Sampled LFP* obtém melhor sensibilidade para amostras treinadas e testadas sob a mesma condição de iluminação. Porém, o *Sampled LFP* apresentou piores resultados que o *LBP riu2* quando testado com amostras capturadas sob condições diferentes de iluminação, demonstrando que não é um método invariante à iluminação.

O aumento da resolução da janela de análise também proporcionou uma melhora na sensibilidade, porém este aumento é limitado pelo tamanho das amostras analisadas. Em geral, a combinação de número de amostras utilizadas na fase de treinamento mais tamanho das amostras é chave para proporcionar o melhor ambiente de trabalho ao *Sampled LFP*.

O experimento com a base de imagens VisTex mostrou que o *Sampled LFP* obtém melhor sensibilidade que o *LBP riu2* quando trabalhando com um maior número de classes de texturas e que o aumento no número de amostras proporciona melhora significativa nos resultados obtidos pelos descritores. Este experimento ainda mostrou

que, quando a mesma região de uma imagem é utilizada para geração de amostras de treino e teste, o processo de classificação consegue melhores índices do que quando os descritores são treinados e testados com amostras de treino e teste geradas a partir de imagens diferentes. Isso significa que os resultados obtidos em experimentos que utilizam as bases de imagens do VisTex, Outex ou álbum de Brodatz não devem ser considerados como indicativos de bom funcionamento de determinado descritor de textura quando aplicado em imagens do mundo real, onde as imagens colhidas do ambiente nunca terão sido experimentadas pelos descritores.

Ainda para todos os experimentos executados, a variante multiresolução do *LFP* clássico, o *LFP-q*, apresentou queda acentuada de sensibilidade conforme a resolução da janela de análise era aumentada, obtendo sempre resultados inferiores ao *Sampled LFP*, não sendo portanto indicado para processamento multiresolução.

## 5. CONCLUSÕES

O trabalho desenvolvido apresentou uma metodologia consistente, aplicada na verificação da viabilidade do *Local Fuzzy Pattern* como um descritor utilizado para classificação de texturas. Os resultados dessa investigação apontaram este descritor como não apto à tarefa de classificação de amostras de texturas sem variação de orientação, sendo, para este caso, indicado o uso do operador *Local Binary Pattern* clássico, que não é robusto a variações de rotação. Não obstante, esses mesmos resultados apontaram o *LFP-s* como um método viável, com melhores resultados que o *LBP riu2*, quando aplicado ao problema de classificação de amostras rotacionadas.

Uma nova variante do *LFP-s* foi proposta, o *Sampled Local Fuzzy Pattern*, adaptado à tarefa de classificação de texturas que tenham sofrido variações de rotação e com suporte à multiresolução. Ainda que a extração de características de texturas em diferentes ângulos de rotação ainda sejam um desafio (RAJU; DURAI, 2013), o *Sampled LFP* mostrou ganhos significativos, quando comparado ao *LBP riu2*.

Porém, os experimentos indicaram que o *Local Fuzzy Pattern* não é um descritor invariante às condições de iluminação, assim, para problemas envolvendo amostras de texturas rotacionadas e capturadas sob diferentes condições de iluminação, é mais indicado o descritor *LBP riu2*.

Os resultados dos vários experimentos também permitiram a constatação de que amostras maiores possibilitam a extração de mais características, melhorando a sensibilidade obtida pelos descritores de textura. A aplicação de janelas de análise maiores também permitiu melhora na sensibilidade, porém, a escolha do tamanho dessa janela depende do tamanho das amostras. Associar uma vizinhança circular de  $P=24$ ,  $R=3$  com amostras pequenas, como 64x64 pixels ou menores, acarreta numa sensibilidade pior do que se fosse escolhida uma vizinhança circular de  $P=16$ ,  $R=2$  para o mesmo tamanho de amostras. Em geral, o *Sampled LFP* obtém máximo desempenho quando trabalhando com uma vizinhança de  $P=16$ ,  $R=2$  em amostras 64x64 pixels.

Apesar do *Sampled LFP* ter se mostrado-superior ao *LBP* como um descritor mais robusto com relação à variação de rotação, mesmo em variadas resoluções, seu processo de parametrização dos valores de  $\beta$  e número de bins se coloca como um limitador para a aplicação deste descritor de texturas, visto que essa fase pode ser

demorada e consumir muitos recursos computacionais, como memória e capacidade de processamento.

Aplicações que manipulem rapidamente várias amostras, como processamento de vídeo, bem como aquelas onde pode haver inclusão de outras classes de textura durante a execução da aplicação ou ainda aplicações que necessitam de trabalhar com amostras capturadas sob diferentes condições de iluminação, não são indicadas para este descritor. O *Sampled LFP* é mais indicado para aquelas onde as condições de iluminação sejam controladas e onde não haja exigência de alteração constante das classes manipuladas.

A execução dos experimentos, utilizando três bancos de imagens bem conhecidos, testes públicos e amplamente utilizados pela comunidade científica e a composição independente para os grupos de treino e teste dão mais força a essas conclusões.

### **5.1. Sugestões para trabalhos futuros**

Algumas propostas para continuidade das pesquisas sobre o *LFP* e o *Sampled LFP*:

- Neste trabalho, a fase de sintonização dos descritores *LFP* foi feita variando-se os valores de  $\beta$  e do número de *bins*. Porém, tal processo poderia ser substituído por outro baseado em inteligência artificial, com a possibilidade de emprego de redes neurais utilizando os histogramas de cada amostra do conjunto alvo, além de outras informações;
- Também neste trabalho, a classificação de texturas com o *Sampled LFP* sempre ocorreu empregando uma única vizinhança circular para treinamento e classificação. Assim, seria interessante pesquisar:
  - Seria possível aumentar a sensibilidade do *Sampled LFP* por meio da combinação dos resultados dos treinamentos obtidos com vizinhanças diferentes?
  - Seria possível estabelecer uma nova forma de identificação de uma textura, baseada na comparação dos descritores aplicados em vizinhanças diversas?

- Verificar o comportamento do *Sampled LFP*, trabalhando com várias resoluções, para o problema de segmentação de imagens.

## REFERÊNCIAS BIBLIOGRÁFICAS

AHMED, F.; HOSSAIN, E.; BARI, A.S.M.H.; SHIHAVUDDIN, A. Compound local binary pattern (clbp) for robust facial expression recognition, **12th IEEE International Symposium on Computational Intelligence and Informatics**, p. 391–395, 2011.

BARCELÓ, A.; MONTSENY, E.; SOBREVILLA, P. On Fuzzy Texture Spectrum for Natural Microtextures Characterization, 4th Conference for Fuzzy Logic and Technology and 11 Rencontres Pharancophones sur la Logique Floue et ses Applications, p. 685-690. 2005.

BOAVENTURA, I. A. G.; GONZAGA, A. Border detection in digital images: An approach by fuzzy numbers. In: **Seventh International Conference on Intelligent Systems Design and Applications**, ISDA, p. 341–346, 2007.

BRODATZ, P. Textures: A photographic album for artists and designers. New York: Dover Publications, 1966.

CHAUDHURI, B. B.; SARKAR, N. Texture segmentation using fractal dimension. In: **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 17, n. 1, p. 72-77, 1995.

CHEN, D.; CAO, X.; WEN, F.; SUN, J. Blessing of Dimensionality: High-dimensional Feature and Its Efficient Compression for Face Verification, Computer Vision and Pattern Recognition (CVPR), 2013.

CHIERICI, C. E. O.; VIEIRA, R.T.; FERRAZ, C. T.; TRAVAINI, J. N.; GONZAGA, A. A new approach for analyzing rotated textures. In: Anais do IX Workshop de Visão Computacional, Rio de Janeiro, 2013.

COGGINS, J. M. A Framework for Texture Analysis Based on Spatial Filtering. 1982 PhD. Thesis - Computer Science Department, Michigan State University, East Lansing, Michigan, 1982.

CROSS, G. R.; JAIN, A. K. Markov Random Field Texture Models. In: **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. PAMI-5, n. 1, p. 25-39, 1983. Disponível em: <http://jivp.eurasipjournals.com/content/2013/1/31>. Acesso em: 02 de setembro de 2013.

DO, M. N.; VETTERLI, M. Wavelet-based texture retrieval using generalized Gaussian density and Kullback-Leibler distance, **IEEE Transactions on Image Processing**, v. 11, n. 2, p. 146-158, 2002.

FAUGERAS, O. D.; PRATT, W. K. Decorrelation methods of texture feature extraction. In: **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 2, p. 323–332, 1980.

GABOR, D. Theory of Communication. **Journal of the Institute Of Electrical Engineers**, v. 93, p. 429-457, 1946.

GARCÍA-OLALLA, O.; ALEGRE, E.; FERNÁNDEZ-ROBLES, L.; GARCÍA-ORDÁS, M. T.; GARCÍA-ORDÁS, D. Adaptive local binary pattern with oriented standard deviation (ALBPS) for texture classification, EURASIP Journal on Image and Video Processing 2013, p. 1-11, Springer.

GONZALEZ, R. C.; WOODS, R. E. Digital Image Processing. Prentice Hall, 2008.

GUO, Z.; ZHANG, L.; ZHANG, D. A completed modeling of local binary pattern operator for texture classification, **IEEE Transactions on Image Processing**, v. 19, n. 6, p. 1657–1663, 2010a.

GUO, Z.; ZHANG, L.; ZHANG, D. Rotation invariant texture classification using LBP variance (LBPV) with global matching, **Pattern Recognition**, v. 43, n. 3, 706-719, 2010b.

HANMANDLU, M.; MADASU, V.; VASIKALRA, S. A fuzzy approach to texture segmentation, **International Conference on Information Technology: Coding and Computing**. Proceedings. ITCC 2004. , v. 1, n. 5-7, p. 636-642, 2004.

HARALICK, R. M. Statistical and Structural Approaches to Texture. In: **Proceedings of the IEEE**, v. 67, n. 5, p. 786-804, 1979.

HARALICK, R. M.; SHANMUGAM, K.; DINSTEIN, I. Textural features for image classification. **IEEE Transactions on System, Man, and Cybernetics**, v. SMC-3, n. 6, p. 610-621, 1973.

HAWKINS, J. K. Textural Properties for Pattern Recognition. In: Lipkin, B.; Rosenfeld, A. (Ed.). **Picture Processing and Psychopictorics**, Academic Press, New York, 1969.

HE, D. C.; WANG, L. Texture Unit, Texture Spectrum, and Texture Analysis. **IEEE Transactions on Geoscience and Remote Sensing**, v. 28, p. 509-512, 1990.

HUANG, Y.; CHAN, K. L.; ZHANG, Z. Texture classification by multi-model feature integration using Bayesian networks, **Pattern Recognition Letters**, v. 24, n. 1-3, p. 393-401, 2003.

IEEE Standard Glossary of Image Processing and Pattern Recognition Terminology, **IEEE Standard**. 610.4-1990, 1990.

JI, H.; YANG, X.; LING, H.; XU, Y. Wavelet Domain Multifractal Analysis for Static and Dynamic Texture Classification, **IEEE Transactions on Image Processing**, v. 22, n. 1, p. 286-299, 2013.

JIA, X.; YANG, X.; CAO, K.; ZANG, Y.; ZHANG, N.; DAI, R.; ZHU, X.; TIAN, J. Multi-scale local binary pattern with filters for spoof fingerprint detection, **Information Sciences**, 2013, <http://dx.doi.org/10.1016/j.ins.2013.06.041>, disponível em: <http://www.sciencedirect.com/science/article/pii/S0020025513004787>. Acesso em: 10 de julho de 2013.

JU, Y.; CHANG, K. H.; HUNG, C.-C. Efficient Edge Detection and Object Segmentation Using Gabor Filters. In: **Proceedings of the 42nd annual ACM Southeast regional conference**, ACM, n. 42, p. 454-459, 2004.

JULESZ, B. Experiments in the visual perception of texture, In: **Scientific American**, v. 232 n. 4, p. 34–43, 1975.

JULESZ, B. Textons, the elements of texture perception, and their interactions. **Nature**. v. 209, p. 91-97, 1981.

LAWS, K.I. Textured image segmentation, PhD Thesis, USCIPI Rep. 940, Image Processing Institute, University of Southern California, Los Angeles, 1980.

LEE, B. A New Method for Classification of Structural Textures. In: **International Journal of Control, Automation, and Systems**, v. 2, n. 1, 2004.

LI, W.; PRASAD, S.; FOWLER, J. E. Hyperspectral Image Classification Using Gaussian Mixture Models and Markov Random Fields. **IEEE Geoscience and Remote Sensing Letters**, v.11, n.1, p.153-157, 2014.

LIAO, S.; ZHU, X.; LEI, Z.; ZHANG, L.; LI, S.Z. Learning Multi-scale Block Local Binary Patterns for Face Recognition. **Lecture Notes in Computer Science**, Springer Berlin Heidelberg, p. 828-837, 2007.

LIU, Y.; ZHANG D.; LU, G.; MA, W. A survey of content based image retrieval with high-level semantics. **Pattern Recognition**, v. 40, n. 1, p. 262–282, 2007.

LOBAY, A.; FORSYTH, D. A. Shape from texture without boundaries. **International Journal of Computer Vision**, Springer Science, Netherlands, v. 67. n. 1, p. 71–91, 2006.

LORIS, N.; ALESSANDRA, L.; SHERYL, B. Survey on LBP based texture descriptors for image classification, **Expert Systems with Applications**, v. 39, n. 3, p. 3634-3641, 2012.

MAANI, R.; KALRA, S.; YANG, Y.H. Rotation Invariant Local Frequency Descriptors for Texture Classification, **IEEE Transactions on image Processing**, v. 22, n. 6, 2013.

MÄENPÄÄ, T.; The Local Binary Pattern Approach to Texture Analisys - Extensions and Applications. Dissertação - Faculty of Technology, Department of Electrical and Information Engineering, University of Oulu, Finlandia, 2003.

MALLAT, S. G. Multifrequency channel decompositions of images and wavelet models, **IEEE Transactions on Acoustics, Speech and Signal Processing**, v.37, n. 12, p. 2091-2110, 1989.

MANDELBROT, B. B. The Fractal Geometry of Nature, Freeman, San Francisco, 1983.

MATERKA, A.; STRZELECKI, M. Texture Analysis Methods – A Review. Technical University of Lodz, Institute of Electronics, COST B11 Report, Brussels, 1998.

MIRMEHDI, M.; XIE, X.; SURI, J. A Galaxy of Texture Features. Chapter. In: **Handbook of Texture Analysis**, p. 375-406, Imperial College Press, 2008.

NIXON, M.; AGUADO, A. Feature Extraction and Image Processing. 2nd ed. Elsevier, 2008.

OJALA, T. and PIETIKÄINEN, M. Texture Classification. Machine Vision and Media Processing Unit, University of Oulu, Finland. Disponível em: [http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL\\_COPIES/OJALA1/texclas.htm](http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OJALA1/texclas.htm), acessado em: 03 de agosto, 2013.

OJALA, T.; MÄENPÄÄ, T.; PIETIKÄINEN, M.; VIERTOLA, J.; KYLLÖNEN, J., HUOVINEN, S. Outex - new framework for empirical evaluation of texture analysis algorithms. In: **Proceedings of the 16th International Conference on Pattern Recognition** (ICPR'02), v. 1, p. 701-706, 2002.

OJALA, T.; PIETIKÄINEN, M. Unsupervised texture segmentation using feature distributions. **Pattern Recognition**, v. 32, p. 477-486, 1999.

OJALA, T.; PIETIKÄINEN, M.; HARWOOD, D. A comparative study of texture measures with classification based on feature distributions. **Pattern Recognition**, v. 29, n. 1, p. 51–59, 1996.

OJALA, T.; PIETIKÄINEN, M.; MÄENPÄÄ, T. Gray Scale and Rotation Invariant Texture Classification with Local Binary Patterns, Proceedings of the 6th European Conference on Computer Vision-Part I, ECCV 2000, p. 404-420, 2000.

OJALA, T.; PIETIKÄINEN, M.; MÄENPÄÄ, T. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. In: **IEEE Transactions on Pattern Analysis and Machine Intelligence**, vol.24, no. 7, p. 971-987, 2002.

PIETIKÄINEN, M.; OJALA, T.; XU, Z. Rotation-invariant texture classification using feature distributions, **Pattern Recognition**, v. 33, p. 43-52, 2000.

PIETIKÄINEN, M.; ZHAO, G.; HADID, A.; AHONEN, T. Computer Vision Using Local Binary Pattern. London: Springer-Verlag, 2011.

POPESCU, D.; DOBRESCU, R.; ANGELESCU, N. STATISTICAL TEXTURE ANALYSIS OF ROAD FOR MOVING OBJECTIVES, U.P.B. Scientific Bulletin, series C, v. 70, n. 4, 2008.

PORTER, R.; CANAGARAJAH, N., Robust rotation-invariant texture classification: wavelet, Gabor filter and GMRF based schemes,Vision, **IEE Proceedings on Image and Signal Processing**, v. 144, n.3, p. 180-188, 1997.

RAJU, J.; DURAI, C. A. D. A survey on texture classification techniques. In: **International Conference on Information Communication and Embedded Systems (ICICES) 2013**, p. 180-184, 2013.

RICHARDS, W.; POLIT, A. Texture matching. **Kybernetik**, Springer-Verlag, v. 16, n. 3, p. 155-162, 1974.

SCHAEFER, G.; DOSHI, N. P. Multi-dimensional local binary pattern descriptors for improved texture analysis, **21st International Conference on Pattern Recognition (ICPR)**, p. 2500-2503, 2012.

SHADKAM, N.; HELFROUSH, M.S.; KAZEMI, K. Local binary patterns partitioning for rotation invariant texture classification. **16th CSI International Symposium on Artificial Intelligence and Signal Processing** (AISP 2012), p. 386-391, 2012.

SHINIVASAN, G. N.; SHOBHA, G. Statistical Texture Analisys. In: **Proceedings of World Academy Of Science, Engeneering and Technology**, v. 36, p. 1264-1269, 2008.

SIVAKUMAR, K.; GOUTSIAS, J. Morphologically constrained GRFs: applications to texture synthesis and analysis, **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 21, n. 2, p. 99-113, 1999.

SKLANSKY, J. Image Segmentation and Feature Extration. **IEEE Transactions on System, Man and Cybernectics**, v. 13, n. 5, p. 907-916, 1978.

SMEUDERS, A. W. M.; WORRING, M., SANTINI, S.; GUPTA, A.; JAIN, R. Content-based image retrieval at the end of the early years. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 22, n. 12, p. 1349-1380, 2000.

SZELISK, R.; Computer Vision: Algorithms and Applications. **Texts in Computer Science**, Springer, 2011.

TAMURA, H.; MORI, S.; YAMAWAKI, T. Textural Features Corresponding to Visual Perception. **IEEE Transactions on System, Man and Cybernetics**, v. 8, p. 460-473, 1978.

TOU, J. Y.; TAY, Y. H.; LAU, P. Y. Recent Trends in Texture Classification: A Review, **Symposium on Progress in Information & Communication Technology**, 2009.

TUCERYAN, M.; JAIN, A. K. Texture Analysis. In: Chen, C.; Pau, L.; Wang, P. (Ed.). **The Handbook of Pattern Recognition and Computer Vision**. World Scientific Publishing Co., cap. 3.1, p. 235-276, 1993.

VIEIRA, R. T. Análise de micropadrões em imagens digitais baseada em números fuzzy. Dissertação de Mestrado, Departamento de Engenharia Elétrica e de Computação, Universidade de São Paulo, São Carlos, 2012.

VIEIRA, R. T.; FERRAZ, C. T.; LANGONI, V. M.; GONZAGA, A. Uma nova proposta para análise de textura local por conjuntos fuzzy. In: **Anais do VIII Workshop de Visão Computacional**, Goiânia, 2012a.

VIEIRA, R. T.; CHIERICI, C. E. O.; FERRAZ, C. T.; GONZAGA, A. Local Fuzzy Pattern: A New way for Micro-pattern Analysis. In: **13th International Conference on Intelligent Data Engineering and Automated Learning**, Natal, 2012b.

VIEIRA, R. T.; CHIERICI, C. E. O.; FERRAZ, C. T.; GONZAGA, A. Image micro-pattern analysis using Fuzzy Numbers. In: **Proceedings of SIBGRAPI 2012 - Conference on Graphics, Patterns and Images**, Ouro Preto, 2012c.

VisTex - Vision Texture Database, Vision and Modeling Group, MIT Media Laboratory, 1995. Disponível em: <<http://vismod.media.mit.edu/vismod/imagery/VisionTexture/>>. Acessado em: 01 de junho de 2013.

WESZKA, J. S.; DYER, C. R.; ROSENFELD, A. A Comparative Study of Texture Measures for Terrain Classification. In: **IEEE Transactions on Systems, Man and Cybernetics**, v. SMC-6, n. 4, p. 269-285, 1976.

XIE, X. A Review of Recent Advances in Surface Defect Detection using Texture analysis Techniques, **Electronic Letters on Computer Vision and Image Analysis**, v. 7, n. 3, p. 1-22, 2008.

ZHANG, J.; TAN, T. Brief review of invariant texture analysis methods. In: **Pattern Recognition**, v. 35, p. 735-747, 2002.

ZHANG, L.; ZHANG, D.; GUO, Z.; Zhang, D. Monogenic-LBP: A new approach for rotation invariant texture classification, **17th IEEE International Conference on Image Processing (ICIP)**, p. 2677-2680, v. 26-29, 2010.

ZUCKER, S. W.; KANT, K. Multiple-level Representations for Texture Discrimination. In: **Proceedings of the IEEE Conference on Pattern Recognition and Image Processing**, p. 609-614, Texas, 1981.



## APÊNDICE A – CÓDIGO FONTE COMUM PARA TESTE E TREINO COM O ÁLBUM DE BRODATZ, OUTEX E VISTEX

```
%%%%%%%%
% BuildLFPMatrix - Gera uma matriz de codigos LFP para um conjunto de
% imagens.
%%%%%%
function lfpMatrix = BuildLFPMatrix(imageSet, janela, numNeighbors, radius, beta,
bellWidth, lfpSamplingFunction)

numImages = size(imageSet, 1);
lfpMatrix = cell(numImages);

for imgIndex = 1 : numImages
    switch lfpSamplingFunction
        case 1 % Sigmoidal function
            lfpMatrix(imgIndex) = { LFP(imageSet{imgIndex}, janela, beta) } ;
        case 2 % Gaussian function
            lfpMatrix(imgIndex) = { LFP_G(imageSet{imgIndex}, janela, bellWidth) } ;
        case 3 % Triangular function
            lfpMatrix(imgIndex) = { LFP_T(imageSet{imgIndex}, janela, beta) } ;
        case 4 % LFP Sampling - sigmoidal
            lfpMatrix(imgIndex) = { LFPS(imageSet{imgIndex}, radius, numNeighbors,
beta) } ;
        case 5 % LFP_Q - sigmoidal
            lfpMatrix(imgIndex) = { LFP_Q(imageSet{imgIndex}, radius, beta) } ;
    end;
end;

%%%%%
% BuildLBPMatrix - Gera uma matriz de codigos LBP para um conjunto de
% imagens.
%%%%%
function lbpMatrix = BuildLBPMatrix(imageSet, numSamples, radius, lbpMode)

numImages = size(imageSet, 1);
lbpMatrix = cell(numImages);

if (lbpMode == 0) % LBP not interpolated
    SP=[-1 -1; -1 0; -1 1; 0 -1; -0 1; 1 -1; 1 0; 1 1];

    for imgIndex = 1 : numImages
        lbpMatrix(imgIndex) = { LBP(imageSet{imgIndex}, SP, 0, 'h') } ; % LBP 8x1
    sem interpolacao
    end;
elseif (lbpMode == 1) % LBP ri
    lbpMapping = getmapping(numSamples, 'ri');

    for imgIndex = 1 : numImages
        lbpMatrix(imgIndex) = { LBP(imageSet{imgIndex}, radius, numSamples,
lbpMapping, 'h') } ;
    end;
elseif (lbpMode == 2) % LBP riu2
    lbpMapping = getmapping(numSamples, 'riu2');

    for imgIndex = 1 : numImages
        lbpMatrix(imgIndex) = { LBP(imageSet{imgIndex}, radius, numSamples,
lbpMapping, 'h') } ;
    end;
end;
```

```

end;

%%%%%%%%%%%%%%%
% ClassifyOnNN computes the classification accuracy
% CP=ClassifyOnNN(DM,trainClassIDs,testClassIDs) returns the classification
accuracy
% The input "DM" is a m*n distance matrix, m is the number of test samples, n is
the number of training samples
% 'trainClassIDs' and 'testClassIDs' stores the class ID of training and
% test samples
%
% Version 1.0
% Authors: Zhenhua Guo, Lei Zhang and David Zhang
% Copyright @ Biometrics Research Centre, the Hong Kong Polytechnic University
%
% Examples
% -----
% I1=imread('rice1.png');
% I2=imread('rice2.png');
% I3=imread('rice3.png');
% I4=imread('rice4.png');
% mapping=getmapping(8,'u2');
% M(1,:)=LBPV(I1,1,8,mapping); % LBPV histogram in (8,1) neighborhood using
uniform patterns
% M(2,:)=LBPV(I2,1,8,mapping);
% S(1,:)=LBPV(I3,1,8,mapping);
% S(2,:)=LBPV(I4,1,8,mapping);
% M = ConvertU2LBP(M,8); % convert u2 LBP or LBpv to meet the requirement of
global matching scheme
% S = ConvertU2LBP(S,8);
% DM = distGMPDRN(M,S,8,2,3);
% CP=ClassifyOnNN(DM,[1,1],[1,1]);
%%%%%%%%%%%%%%%
function result = ClassifyOnNN(DM, trainClassIDs, testClassIDs, threshold)

if nargin<3
    disp('Not enough input parameters.')
    return
end

useThreshold = false;

numClassIDs = length(unique(trainClassIDs));
resultMatrix = zeros(length(testClassIDs), 3 + numClassIDs); %
trainsClasses(output, 0/1-Yes/No), testClass(input), minDistance, maxDistance

if (threshold > 0)
    useThreshold = true;
end;

for i = 1 : length(testClassIDs);

    if (useThreshold)
        indexes = find(DM(i,:) < threshold);
        values = DM(i, indexes);
    else
        [values, indexes]= min(DM(i,:));    % find Nearest Neighborhood
    end;

    for j = 1 : length(indexes)
        trainClassId = trainClassIDs(indexes(j));

        if (resultMatrix(i, trainClassId+1) == 0)
            resultMatrix(i, trainClassId+1) = 1;
    end;
end;

```

```

    end;
end;

resultMatrix(i, numClassIDs + 1) = testClassIDs(i);

if (~isempty(values))
    resultMatrix(i, numClassIDs + 2) = min(values);
    resultMatrix(i, numClassIDs + 3) = max(values);
end;
end;

confusaoResult = Confusao(resultMatrix, trainClassIDs);
result = confusaoResult;

%%%%%%%%%%%%%%%
% Gera a matriz de confusao
% resultMatrix - no seguinte formato:
%
% resultMatrix(:, 1-n) = resultClass (output) (valores: 0/1 == Neg/Pos)
% resultMatrix(:, n+1) = testClass (input)
% resultMatrix(:, n+2) = minDistance
% resultMatrix(:, n+3) = maxDistance
%%%%%%%%%%%%%%%
function result = Confusao( resultMatrix, trainClassIDs )

trainClassIDs = unique(trainClassIDs);
numClassIDs = length(unique(trainClassIDs));

testClassIndex = numClassIDs + 1;
minDistanceIndex = numClassIDs + 2;
maxDistanceIndex = numClassIDs + 3;

vp = 0; % verdadeiros positivos
fp = 0; % falsos positivos
vn = 0; % verdadeiros negativos
fn = 0; % falsos negativos

[numLinhas numColunas] = size(resultMatrix);
searchResult = zeros(numLinhas, 3);

for i = 1 : numLinhas
    testClassId = resultMatrix(i, testClassIndex);
    searchResult(i, 1) = i;
    searchResult(i, 2) = testClassId;

    for j = 1 : numClassIDs
        result = resultMatrix(i, j);

        if (result == 1) % classe encontrada

            if searchResult(i, 3) == 0
                searchResult(i, 3) = trainClassIDs(j);
            else
                searchResult(i, 3) = trainClassIDs(j);
            end

            if (testClassId == trainClassIDs(j))
                % verdadeiro positivo
                vp = vp + 1;
            else
                % falso positivo
                fp = fp + 1;
            end;
        end;
    end;
end;

```

```

    else % classe nao encontrada
        if (testClassId == trainClassIDs(j))
            % falso negativo
            fn = fn + 1;
        else
            % verdadeiro negativo
            vn = vn + 1;
        end;
    end;

end;

filename = 'Matches.xls';
sheet = 'result';
rangeSpecifier = 'A1:C%d';

data = searchResult;
range = sprintf(rangeSpecifier, numLinhas);

saveConfusionMatrix = false;

if saveConfusionMatrix
    disp('Matriz de confusao salva (Confusao.m).');
    xlswrite(filename, data, sheet, range);
else
    disp('Matriz de confusao NÃO salva (Confusao.m).');
end;

minDist = min(resultMatrix(:, minDistanceIndex));
maxDist = max(resultMatrix(:, maxDistanceIndex));

totalAcertos = vp + fp;
acertosPositivos = vp;
acertosNegativos = vn;
totalDados = numLinhas;
totalPositivos = vp + fn;
totalNegativos = vn + fp;
totalPredicoesPos = vp + fp;
totalPredicoesNeg = vn + fn;

acuracia = totalAcertos / totalDados;
sensibilidade = acertosPositivos / totalPositivos;
especificidade = acertosNegativos / totalNegativos;
eficiencia = (sensibilidade + especificidade)/2;
preditividadePos = acertosPositivos / totalPredicoesPos;
preditividadeNeg = acertosNegativos / totalPredicoesNeg;
% coeficiente de correlacao de Matthews. Medida de qualidade das classificacoes
binarias. Retorna um valor entre -1 e +1. Predicao perfeita = +1. Predicao inversa
= -1.
coefMatthews = ((vp*vn)-(fp*fn))/sqrt( (vp+fp)*(vp+fn)*(vn+fp)*(vn+fn) );

result = [minDist, maxDist, sensibilidade, especificidade];

%%%%%%%%%%%%%
% distMATCHisSquare computes the dissimilarity between training samples and a test
sample
% DV = distMATCHisSquare(train, test) returns the distance vector between training
samples and a test sample.
% The input "train" is a n*d matrix, and each row of it represent one
% training sample. The "test" is a 1*d vector.
%
% Version 1.0
% Authors: Zhenhua Guo, Lei Zhang and David Zhang

```

```
% Copyright @ Biometrics Research Centre, the Hong Kong Polytechnic University
%
% Examples
% -----
%
% I1=imread('rice1.png');
% I2=imread('rice2.png');
% I3=imread('rice3.png');
% mapping=getmapping(8,'u2');
% M(1,:)=LBPV(I1,1,8,mapping); % LBPV histogram in (8,1) neighborhood using
uniform patterns
% M(2,:)=LBPV(I2,1,8,mapping);
% S=LBPV(I3,1,8,mapping);
% DV = distMATCHChiSquare(M,S);
%%%%%%%%%%%%%%%
function DV = distMATCHChiSquare(trains, test)

[train_row, train_col] = size(trains);
[test_row, test_col] = size(test);

testExtend = repmat(test, train_row, 1);
subMatrix = trains-testExtend;
subMatrix2 = subMatrix.^2;
addMatrix = trains+testExtend;

idxZero = find(addMatrix==0);
addMatrix(idxZero)=1;
DistMat = subMatrix2./addMatrix;
DV = sum(DistMat,2);

%%%%%%%%%%%%%%%
% GETMAPPING returns a structure containing a mapping table for LBP codes.
% MAPPING = GETMAPPING(SAMPLES,MAPPINGTYPE) returns a
% structure containing a mapping table for
% LBP codes in a neighbourhood of SAMPLES sampling
% points. Possible values for MAPPINGTYPE are
% 'u2' for uniform LBP
% 'ri' for rotation-invariant LBP
% 'riu2' for uniform rotation-invariant LBP.
%
% Version 0.1.1
% Authors: Marko Heikkilä and Timo Ahonen
% Changelog
% 0.1.1 Changed output to be a structure
% Fixed a bug causing out of memory errors when generating rotation
% invariant mappings with high number of sampling points.
% Lauge Sorensen is acknowledged for spotting this problem.
%
% Example:
% I=imread('rice.tif');
% MAPPING=getmapping(16,'riu2');
% LBPHIST=lbp(I,2,16,MAPPING,'hist');
% Now LBPHIST contains a rotation-invariant uniform LBP
% histogram in a (16,2) neighbourhood.
%%%%%%%%%%%%%%%
function mapping = getmapping(samples,mappingtype)

table = 0:2^samples-1;
newMax = 0; %number of patterns in the resulting LBP code
index = 0;

if strcmp(mappingtype,'u2') %Uniform 2
newMax = samples*(samples-1) + 3;
for i = 0:2^samples-1
j = bitset(bitshift(i,1,samples),1,bitget(i,samples)); %rotate left

```

```

numt = sum(bitget(bitxor(i,j),1:samples)); %number of 1->0 and
%0->1 transitions
%in binary string
%x is equal to the
%number of 1-bits in
%XOR(x, Rotate left(x))

if numt <= 2
    table(i+1) = index;
    index = index + 1;
else
    table(i+1) = newMax - 1;
end
end
end

if strcmp(mappingtype,'ri') %Rotation invariant
tmpMap = zeros(2^samples,1) - 1;
for i = 0:2^samples-1
    rm = i;
    r = i;

    for j = 1:samples-1
        r = bitset(bitshift(r,1,samples),1,bitget(r,samples)); %rotate
        %left
        if r < rm
            rm = r;
        end
    end
    if tmpMap(rm+1) < 0
        tmpMap(rm+1) = newMax;
        newMax = newMax + 1;
    end
    table(i+1) = tmpMap(rm+1);
end
end

if strcmp(mappingtype,'riu2') %Uniform & Rotation invariant
newMax = samples + 2;
for i = 0:2^samples - 1
    j = bitset(bitshift(i,1,samples),1,bitget(i,samples)); %rotate left
    numt = sum(bitget(bitxor(i,j),1:samples));
    if numt <= 2
        table(i+1) = sum(bitget(i,1:samples));
    else
        table(i+1) = samples+1;
    end
end
end

mapping.table=table;
mapping.samples=samples;
mapping.num=newMax;

%%%%%%%%%%%%%%%
% LBP returns the local binary pattern image or LBP histogram of an image.
% J = LBP(I,R,N,MAPPING,MODE) returns either a local binary pattern
% coded image or the local binary pattern histogram of an intensity
% image I. The LBP codes are computed using N sampling points on a
% circle of radius R and using mapping table defined by MAPPING.
% See the getmapping function for different mappings and use 0 for
% no mapping. Possible values for MODE are
% 'h' or 'hist' to get a histogram of LBP codes
% 'nh'          to get a normalized histogram
% Otherwise an LBP code image is returned.
%
```

```
% J = LBP(I) returns the original (basic) LBP histogram of image I
%
% J = LBP(I,SP,MAPPING,MODE) computes the LBP codes using n sampling
% points defined in (n * 2) matrix SP. The sampling points should be
% defined around the origin (coordinates (0,0)).
%
% Version 0.3.2
% Authors: Marko Heikkilä and Timo Ahonen
%
% Changelog
% Version 0.3.2: A bug fix to enable using mappings together with a
% predefined spoints array
% Version 0.3.1: Changed MAPPING input to be a struct containing the mapping
% table and the number of bins to make the function run faster with high number
% of sampling points. Lauge Sorensen is acknowledged for spotting this problem.
%
% Examples
-----
%
% I=imread('rice.png');
% mapping=getmapping(8,'u2');
% H1=LBP(I,1,8,mapping,'h'); %LBP histogram in (8,1) neighborhood
% %using uniform patterns
% subplot(2,1,1),stem(H1);
%
%
% H2=LBP(I);
% subplot(2,1,2),stem(H2);
%
%
% SP=[-1 -1; -1 0; -1 1; 0 -1; -0 1; 1 -1; 1 0; 1 1];
% I2=LBP(I,SP,0,'i'); %LBP code image using sampling points in SP
% %and no mapping. Now H2 is equal to histogram
% %of I2.
%%%%%%%%%%%%%%%
function result = LBP(varargin) % image,radius,neighbors,mapping,mode)

% Check number of input arguments.
error(nargchk(1,5,nargin));

image=varargin{1};
d_image=double(image);

if nargin==1
    spoints=[-1 -1; -1 0; -1 1; 0 -1; -0 1; 1 -1; 1 0; 1 1];
    neighbors=8;
    mapping=0;
    mode='h';
end

if (nargin == 2) && (length(varargin{2}) == 1)
    error('Input arguments');
end

if (nargin > 2) && (length(varargin{2}) == 1)
    radius=varargin{2};
    neighbors=varargin{3};
    spoints=zeros(neighbors,2);
    % Angle step.
    a = 2*pi/neighbors;

    for i = 1:neighbors
        spoints(i,1) = -radius*sin((i-1)*a);
        spoints(i,2) = radius*cos((i-1)*a);
    end

    if(nargin >= 4)
        mapping=varargin{4};
        if(isstruct(mapping) && mapping.samples ~= neighbors)
            error('Incompatible mapping');
    end
end
```

```

        end
    else
        mapping=0;
    end

    if(nargin >= 5)
        mode=varargin{5};
    else
        mode='h';
    end
end

if (nargin > 1) && (length(varargin{2}) > 1)
spoints=varargin{2};
neighbors=size(spoints,1);

if(nargin >= 3)
    mapping=varargin{3};
    if(isstruct(mapping) && mapping.samples ~= neighbors)
        error('Incompatible mapping');
    end
else
    mapping=0;
end

if(nargin >= 4)
    mode=varargin{4};
else
    mode='h';
end
end

% Determine the dimensions of the input image.
[ysize xsize] = size(image);
miny=min(spoints(:,1));
maxy=max(spoints(:,1));
minx=min(spoints(:,2));
maxx=max(spoints(:,2));

% Block size, each LBP code is computed within a block of size bsizex*bsizey
bsizex=ceil(max(maxy,0))-floor(min(miny,0))+1;
bsizey=ceil(max(maxx,0))-floor(min(minx,0))+1;

% Coordinates of origin (0,0) in the block
origy=1-floor(min(miny,0));
origx=1-floor(min(minx,0));

% Minimum allowed size for the input image depends
% on the radius of the used LBP operator.
if(xsize < bsizex || ysize < bsizey)
    error('Too small input image. Should be at least (2*radius+1) x (2*radius+1)');
end

% Calculate dx and dy;
dx = xsize - bsizex;
dy = ysize - bsizey;

% Fill the center pixel matrix C.
C = image(origy:origy+dy,origx:origx+dx);
d_C = double(C);
bins = 2^neighbors;

% Initialize the result matrix with zeros.
result=zeros(dy+1,dx+1);

%Compute the LBP code image

```

```

for i = 1:neighbors
    y = spoints(i,1)+origy;
    x = spoints(i,2)+origx;
    % Calculate floors, ceils and rounds for the x and y.
    fy = floor(y); cy = ceil(y); ry = round(y);
    fx = floor(x); cx = ceil(x); rx = round(x);
    % Check if interpolation is needed.
    if (abs(x - rx) < 1e-6) && (abs(y - ry) < 1e-6)
        % Interpolation is not needed, use original datatypes
        N = image(ry:ry+dy,rx:rx+dx);
        D = N >= C;
    else
        % Interpolation needed, use double type images
        ty = y - fy;
        tx = x - fx;

        % Calculate the interpolation weights.
        w1 = (1 - tx) * (1 - ty);
        w2 = tx * (1 - ty);
        w3 = (1 - tx) * ty ;
        w4 = tx * ty ;
        % Compute interpolated pixel values
        N = w1*d_image(fy:fy+dy,fx:fx+dx) + w2*d_image(fy:fy+dy,cx:cx+dx) + ...
            w3*d_image(cy:cy+dy,fx:fx+dx) + w4*d_image(cy:cy+dy,cx:cx+dx);
        D = N >= d_C;
    end
    % Update the result matrix.
    v = 2^(i-1);
    result = result + v*D;
end

%Apply mapping if it is defined
if isstruct(mapping)
    bins = mapping.num;
    for i = 1:size(result,1)
        for j = 1:size(result,2)
            result(i,j) = mapping.table(result(i,j)+1);
        end
    end
end

%%%%%%%%%%%%%%%
% LBP_Hist - Gera um histograma para uma matriz de codigos LBP.
%%%%%%%%%%%%%%%
function result = LBP_Hist(lbp_matrix, numBins)

result=hist(lbp_matrix(:,0:(numBins-1));
result=result/sum(result);

%%%%%%%%%%%%%%%
% LFP - Local Fuzzy Pattern - classic form (LFP-s).
%
% Input arguments:
% img - Gray scale image.
% neighborSize - number of sample point in an square window.
% beta - pertinence function argument.
%%%%%%%%%%%%%%%
function result = LFP(img, neighborSize, beta)

if neighborSize < 3 || floor(neighborSize/2) == 0
    error('A vizinhança deve ser um número ímpar maior ou igual a 3!');

```

```

end;

img = double(img);

% Tamanho da imagem original
[ysize xsize] = size(img);

if(xsize < neighborSize || ysize < neighborSize)
    error('Imagen muito pequena. De ter pelo menos o tamanho da janela.');
end

weightMatrix = [1 1 1; 1 1 1; 1 1 1]; %matriz de pesos 1
border = fix(neighborSize/2);
dataMatrix = img(2*border : ysize - border, 2*border : xsize - border);

[matrixSizeY matrixSizeX] = size(dataMatrix);
pertinenceSum = zeros(matrixSizeY, matrixSizeX);
weightSum = 0;

for i = 1 : neighborSize
    for j = 1 : neighborSize
        weight = weightMatrix(i, j);

        windowData = img(i : (i+matrixSizeY) - 1, j : (j+matrixSizeX) - 1);
        pert = 1./(1 + exp(-(windowData - dataMatrix)/beta));

        pertinenceSum = pertinenceSum + (pert * weight);
        weightSum = weightSum + weight;
    end;
end;

result = pertinenceSum / weightSum;

%%%%%%%%%%%%%%%
% LFP-G - Local Fuzzy Pattern - Gaussian function.
%
% Input arguments:
% img - Gray scale image.
% neighborSize - number of sample point in an square window.
% bellWidth - controls the width of the gaussian "bell" curve.
%%%%%%%%%%%%%%%
function result = LFP_G(img, neighborSize, bellWidth)

if neighborSize < 3 || floor(neighborSize/2) == 0
    error('A vizinhança deve ser um número ímpar maior ou igual a 3!');
end;

img = double(img);

% Tamanho da imagem original
[ysize xsize] = size(img);

if(xsize < neighborSize || ysize < neighborSize)
    error('Imagen muito pequena. De ter pelo menos o tamanho da janela.');
end

weightMatrix = [1 1 1; 1 1 1; 1 1 1]; %matriz de pesos 1
border = fix(neighborSize/2);
dataMatrix = img(2*border : ysize - border, 2*border : xsize - border);

[matrixSizeY matrixSizeX] = size(dataMatrix);
pertinenceSum = zeros(matrixSizeY, matrixSizeX);

```

```

weightSum = 0;

for i = 1 : neighborSize
    for j = 1 : neighborSize
        weight = weightMatrix(i, j);
        windowData = img(i : (i+matrixSizeY) - 1, j : (j+matrixSizeX) - 1);

        % Gaussian function arguments:
        %
        % peakHeight - height of the curve's peak.
        % centrePos - position of the centre of the peak.
        % bellWidth - controls the width of the "bell".
        %
        % Gaussian function:
        % result = peakHeight*(exp( -( (x-centrePos).^2)/(2*(bellWidth^2)) ) );
        %
        pert = exp( -( (windowData-dataMatrix).^2)/(2*(bellWidth^2)) ) ;

        pertinenceSum = pertinenceSum + (pert * weight);
        weightSum = weightSum + weight;
    end;
end;

result = pertinenceSum / weightSum;

%%%%%%%%%%%%%%%
% LBP_Hist - Gera um histograma para uma matriz de codigos LFP.
%%%%%%%%%%%%%%%
function result = LFP_Hist(lfp_matrix, numBins)

result = imhist(lfp_matrix, numBins);
result = reshape(result, 1, size(result, 1) * size(result, 2));
result = result/sum(result);

%%%%%%%%%%%%%%%
% LFP-Q - Local Fuzzy Pattern Q
%
% Input arguments:
% img - Gray scale image.
% neighborSize - number of sample point in an square window.
% beta - pertinence function argument.
%%%%%%%%%%%%%%%
function result = LFP_Q(img, ratio, beta)

if ratio < 1
    error('O valor do raio deve ser maior ou igual a 1.');
end

img = double(img);
[ysize xsize] = size(img);
neighborSize = (ratio * 2) + 1;

if(xsize < neighborSize || ysize < neighborSize)
    error('Imagen muito pequena. De ter pelo menos o tamanho da janela.');
end

dataMatrix = img(ratio+1 : ysize - ratio, ratio+1 : xsize - ratio);

[matrixSizeY matrixSizeX] = size(dataMatrix);
pertinenceSum = zeros(matrixSizeY, matrixSizeX);
weight = 1;

```

```

weightSum = 0;

for x = 1 : neighborSize
    if x == 1 || x == neighborSize
        step = 1;
    else
        step = neighborSize-1;
    end

    for y = 1 : step : neighborSize
        windowData = img(y : (y+matrixSizeY)-1, x : (x+matrixSizeX)-1);
        pert = 1./(1 + exp(-(windowData - dataMatrix)/beta));
        pertinenceSum = pertinenceSum + (pert * weight);
        weightSum = weightSum + weight;
    end
end

result = pertinenceSum / weightSum;

%%%%%%%%%%%%%%%
% LFP triangular - Local Fuzzy Pattern
%
% Input arguments:
% img - Gray scale image.
% neighborSize - number of sample point in an square window.
% beta - pertinence function argument.
%%%%%%%%%%%%%%%
function result = LFP_T(img, neighborSize, beta)

if neighborSize < 3 || floor(neighborSize/2) == 0
    error('A vizinhança deve ser um número ímpar maior ou igual a 3!');
end;

img = double(img);

% Tamanho da imagem original
[ysize xsize] = size(img);

if(xsize < neighborSize || ysize < neighborSize)
    error('Imagen muito pequena. De ter pelo menos o tamanho da janela.');
end

weightMatrix = [1 1 1; 1 1 1; 1 1 1]; %matriz de pesos 1
border = fix(neighborSize/2);
dataMatrix = img(2*border : ysize - border, 2*border : xsize - border);

[matrixSizeY matrixSizeX] = size(dataMatrix);
pertinenceSum = zeros(matrixSizeY, matrixSizeX);
weightSum = 0;

for i = 1 : neighborSize
    for j = 1 : neighborSize
        weight = weightMatrix(i, j);

        windowData = img(i : (i+matrixSizeY) - 1, j : (j+matrixSizeX) - 1);

        val1 = (windowData - (dataMatrix-beta))/beta;
        val2 = ((dataMatrix+beta) - windowData)/beta;
        pert = max(min(val1, val2), 0);

        pertinenceSum = pertinenceSum + (pert * weight);
        weightSum = weightSum + weight;
    end;
end;

```

```

end;

result = pertinenceSum / weightSum;

%%%%%%%%%%%%%%%
% LFPS - Sampled Local Fuzzy Pattern
%
% Input arguments:
% image - Gray scale image.
% radius - distance from neighborhood to the center pixel.
% numNeighbors - number of sample points in the neighborhood circle.
% beta - pertinence function argument.
%%%%%%%%%%%%%%%
function result = LFPS(image, radius, numNeighbors, beta)

d_image = double(image);
spoints = zeros(numNeighbors,2);

% Angle step.
a = 2*pi/numNeighbors;

for i = 1:numNeighbors
    spoints(i,1) = -radius*sin((i-1)*a);
    spoints(i,2) = radius*cos((i-1)*a);
end

% Determine the dimensions of the input image.
[ysize xsize] = size(image);
numNeighbors=size(spoints,1);

miny=min(spoints(:,1));
maxy=max(spoints(:,1));
minx=min(spoints(:,2));
maxx=max(spoints(:,2));

% Block size, each LFP code is computed within a block of size bsizex*bsizey
bsizey=ceil(max(maxy,0))-floor(min(miny,0))+1;
bsizex=ceil(max(maxx,0))-floor(min(minx,0))+1;

% Coordinates of origin (0,0) in the block
origy=1-floor(min(miny,0));
origx=1-floor(min(minx,0));

% Minimum allowed size for the input image depends
% on the radius of the used LFP operator.
if(xsize < bsizex || ysize < bsizey)
    error('Imagen muito pequena. De ter pelo menos (2*radius+1) x (2*radius+1).');
end

% Calculate dx and dy;
dx = xsize - bsizex;
dy = ysize - bsizey;

% Fill the center pixel matrix C.
C = image(origy:origy+dy,origx:origx+dx);
d_C = double(C);

% Initialize the result matrix with zeros.
pertinenceSum = zeros(dy+1, dx+1);

%Compute the LBP code image
for i = 1 : numNeighbors
    y = spoints(i,1)+origy;

```

```

x = spoints(i,2)+origx;

% Calculate floors, ceils and rounds for the x and y.
fy = floor(y);
cy = ceil(y);
ry = round(y);
fx = floor(x);
cx = ceil(x);
rx = round(x);

% Check if interpolation is needed.
if (abs(x - rx) < 1e-6) && (abs(y - ry) < 1e-6)
    % Interpolation is not needed, use original datatypes
    N = image(ry:ry+dy,rx:rx+dx);
else
    % Interpolation needed, use double type images
    ty = y - fy;
    tx = x - fx;

    % Calculate the interpolation weights.
    w1 = (1 - tx) * (1 - ty);
    w2 =      tx * (1 - ty);
    w3 = (1 - tx) *      ty ;
    w4 =      tx *      ty ;

    % Compute interpolated pixel values
    N = w1*d_image(fy:fy+dy,fx:fx+dx) + w2*d_image(fy:fy+dy,cx:cx+dx) + ...
        w3*d_image(cy:cy+dy,fx:fx+dx) + w4*d_image(cy:cy+dy,cx:cx+dx);
end

pert = 1./(1 + exp(-(N - d_C) / beta)); % sigmoid function
pertinenceSum = pertinenceSum + max(0, pert);
end

result = pertinenceSum/numNeighbors;

%%%%%%%%%%%%%%%
% LogLikelihood - Faz a comparacao entre dois conjuntos, utilizando a
% metrica estatistica G.
%%%%%%%%%%%%%%%
function result = LogLikelihood(trains, test)

[train_row, train_col] = size(trains);
[test_row, test_col] = size(test);

testExtend = repmat(test, train_row, 1);
trains = trains +1;
testExtend = testExtend +1;

result = 2*sum(abs(trains.* (log(trains./testExtend))), 2);

%%%%%%%%%%%%%%%
% ReadOutexTxt gets picture IDs and class IDs from txt for Outex Database
% [filenames, classIDs] = ReadOutexTxt(txtfile) gets picture IDs and class
% IDs from TXT file for Outex Database
%
% Version 1.0
% Authors: Zhenhua Guo, Lei Zhang and David Zhang
% Copyright @ Biometrics Research Centre, the Hong Kong Polytechnic University
%%%%%%%%%%%%%%%
function [filenames, classIDs] = ReadOutexTxt(testCaseIndex, testInUse, txtfile)

```

```

testPath = testInUse{1};

contribTest(1) = {'Contrib_TC_00000'};
contribTest(2) = {'Contrib_TC_00001'};
contribTest(3) = {'Contrib_TC_00002'};
contribTest(4) = {'Contrib_TC_00003'};

useAlternativeFormat = false;

for i = 1 : length(contribTest)
    contribTestPath = contribTest{i};

    idx = strfind(testPath, contribTestPath);

    if length(idx) > 0
        useAlternativeFormat = true;
    end
end;

fid = fopen(txtfile,'r');
firstLine = true;
i = 0;

while 1
    tline = fgetl(fid);

    if ~ischar(tline)
        break;
    end

    if firstLine == true
        if length(strfind(tline, ' ')) == 0 % header - number of imgs
            continue;
        end

        firstLine = false;
    end

    index = findstr(tline, '.');

    if isempty(index)
        index = findstr(tline, ' ');
    end

    i = i+1;
    filenames(i) = str2num(tline(1:index-1))+1; % the picture ID starts from 0, but
the index of Matlab array starts from 1

    index = findstr(tline, ' ');

    if (useAlternativeFormat == true)
        classIDs(i) = str2num(tline(index:end-3)); % Contrib_TC_00000 - CONTRIB
ONLY!
    else
        classIDs(i) = str2num(tline(index:end));
    end;
end

fclose(fid);

```



## APÊNDICE B – CÓDIGO FONTE EXCLUSIVO PARA TREINO COM O ÁLBUM DE BRODATZ E OUTEX

```
%%%%%%%%%%%%%%%%
% LFPBestBeta - Determina o melhor beta para o LFP e determinado caso de
% teste.
%%%%%%%%%%%%%%%
function result = LFPBestBeta(imgSet, testCaseIndex, testInUse, janela,
numNeighbors, radius, numBins, fileVersion, lfpSamplingFunction)

testPath = testInUse{1};
numImages = testInUse{3};
classifier = testInUse{6};

fprintf('\n');
disp('-----');
message = sprintf('LFP Best Beta - %s\nParams:\nfunction: %d, winSize: %d,
numNeighbors: %d, radius: %d, numBins: %d, iteration: %d\n', testPath,
lfpSamplingFunction, janela, numNeighbors, radius, numBins, fileVersion);
disp(message);
disp('-----');

% Beta list
if lfpSamplingFunction == 3 % triangular function
    betaList = [1:1:256]';
else
    betaList = [0.100:0.050:2.500]';
end

threshold = 0;
lfp_hist = zeros(numImages, numBins);
sensitivityTable = zeros(length(betaList), 3);

for betaIndex = 1 : size(betaList);
    beta = betaList(betaIndex);

    % Generates LFP matrix
    lfp_matrix = BuildLFPMatrix(imgSet, janela, numNeighbors, radius, beta, 0,
lfpSamplingFunction);

    % Generates LFP histogram
    disp('Generating LFP histogram...');
    tic
    for imgIndex = 1 : numImages
        lfp_hist(imgIndex, :) = LFP_Hist(lfp_matrix{imgIndex}, numBins);
    end;
    toc

    % Classification process
    resultClassifier = ClassifyData(testCaseIndex, testInUse, lfp_hist, threshold);
    sensitivityTable(betaIndex, 1) = beta;
    sensitivityTable(betaIndex, 2) = resultClassifier(:, 4);
end;

% Display the results
disp('-----');
disp('LFP Best Beta');
message = sprintf('Test case: %s', testPath);
disp(message);
message = sprintf('Function: %d', lfpSamplingFunction);
```

```

disp(message);
message = sprintf('Num bins: %d', numBins);
disp(message);
fprintf('\n');

[sensitivity row] = max(sensitivityTable(:,2));
bestBeta = sensitivityTable(row, 1);

classifierName = 'Chi-square';
if (classifier == 1)
    classifierName = 'Log-likelihood';
end;

message = sprintf('%s - Best beta: %f, sensitivity: %f', classifierName, bestBeta,
sensitivity);
disp(message);

fprintf('\n');
disp('Sensitivity table:');
fprintf('\tBeta\tSensitivity\n');
disp(sensitivityTable);
disp('-----');

% Results
slashOccurr = strfind(testPath, '\\');
numOccurr = size(slashOccurr, 2);
testName = testPath(slashOccurr(numOccurr-1)+2 : slashOccurr(numOccurr)-1);

% Plot and Saves Sensitivity Graph
h = figure;
plot(sensitivityTable(:,1), sensitivityTable(:,2));
title(classifierName);
ylabel('Sensitivity');
xlabel('Beta value');
grid;
fileName = sprintf('%s_Beta_classifier%d_func%d_ver%d', testName, classifier,
lfpSamplingFunction, fileVersion);
saveas(h, fileName, 'jpg');
close(h);

% Saves the results in an excel file
filename = sprintf('Outex_BestBeta_classifier%d_func%d_ver%d.xls', classifier,
lfpSamplingFunction, fileVersion);
mainSheet = 'Best_Beta';
rangeSpecifier = 'A%d:F%d';

% Header
data = [{'Test Name'}, {'Bins'}, {'Best Beta'}, {'Sensitivity'}];
range = sprintf(rangeSpecifier, 1, 1);
xlswrite(filename, data, mainSheet, range);

% Summary
data = [{testName}, {numBins}, {bestBeta}, {sensitivity}];
range = sprintf(rangeSpecifier, testCaseIndex+1, testCaseIndex+1);
xlswrite(filename, data, mainSheet, range);

% Detailed Data
message = sprintf('Saving results in %s', filename);
disp(message);

detailSheet = testName;
xlswrite(filename, sensitivityTable, detailSheet);
result = bestBeta;

```

```

%%%%%%%%%%%%%%%
% LFPBestBins - Determina o melhor numero de bins para o LFP e determinado
% caso de teste.
%%%%%%%%%%%%%%%
function result = LFPBestBins(imgSet, testCaseIndex, testInUse, janela,
numNeighbors, radius, beta, bellWidth, fileVersion, lfpSamplingFunction)

testPath = testInUse{1};
numImages = testInUse{3};
classifier = testInUse{6};

fprintf('\n');
disp('=====');
message = sprintf('LFP Best Bins - %s\nParams:\nfunction: %d, winSize: %d,
numNeighbors: %d, radius: %d, beta: %f, bell width: %f, iteration: %d',
testPath,
lfpSamplingFunction, janela, numNeighbors, radius, beta, bellWidth, fileVersion);
disp(message);
disp('-----');

% number of bins
numBinsList = [256, 250:-10:110, 100:-5:10, 5];
% Generates LFP matrix
lfp_matrix = BuildLFPMatrix(imgSet, janela, numNeighbors, radius, beta, bellWidth,
lfpSamplingFunction);

% calculates sensitivity
threshold = 0;

sensitivityTable = zeros(length(numBinsList), 2);

for binsIndex = 1 : length(numBinsList);
    numBins = numBinsList(binsIndex);
    lfp_hist = zeros(numImages, numBins);

    % Generates LFP histogram
    disp('Generating LFP histogram...');
    tic
    for imgIndex = 1 : numImages
        lfp_hist(imgIndex, :) = LFP_Hist(lfp_matrix{imgIndex}, numBins);
    end;
    toc

    % Classification process
    resultsClassifier = ClassifyData(testCaseIndex, testInUse, lfp_hist,
threshold);
    sensitivityTable(binsIndex, 1) = numBins;
    sensitivityTable(binsIndex, 2) = resultsClassifier(:, 4);
end;

% Display the results
disp('-----');
disp('LFP Best Bins');
message = sprintf('Test case: %s', testPath);
disp(message);
message = sprintf('Function: %d', lfpSamplingFunction);
disp(message);
message = sprintf('Beta: %f, Bell width: %f', beta, bellWidth);
disp(message);
fprintf('\n');

[sensitivity row] = max(sensitivityTable(:,2));
bestBins = sensitivityTable(row, 1);

classifierName = 'Chi-square';
if (classifier == 1)

```

```

    classifierName = 'Log-likelihood';
end;

message = sprintf('%s - Best bins: %d, sensitivity: %f', classifierName, bestBins,
sensitivity);
disp(message);

fprintf('\n');
disp('Sensitivity table:');
fprintf('\tBins\tSensitivity\n');
disp(sensitivityTable);
disp('-----');

% Results
slashOccurr = strfind(testPath, '\\');
numOccurr = size(slashOccurr, 2);
testName = testPath(slashOccurr(numOccurr-1)+2 : slashOccurr(numOccurr)-1);

% Plot and Saves Sensitivity Graph
h = figure;
plot(sensitivityTable(:,1), sensitivityTable(:,2));
title(classifierName);
ylabel('Sensitivity');
xlabel('Number of bins');
grid;
fileName = sprintf('%s_Bins_classifier%d_func%d_ver%d', testName, classifier,
lfpSamplingFunction, fileVersion);
saveas(h, fileName, 'jpg');
close(h);

% Saves the results in an excel file
filename = sprintf('Outex_BestBins_classifier%d_func%d_ver%d.xls', classifier,
lfpSamplingFunction, fileVersion);
mainSheet = 'Best_Bins';
rangeSpecifier = 'A%d:E%d';

% Header
data = {[{'Test Name'}, {'Beta'}, {'BellWidth'}, {'Num Bins'}, {'Sensitivity'}]};
range = sprintf(rangeSpecifier, 1, 1);
xlswrite(filename, data, mainSheet, range);

% Summary
data = {[{testName}, {beta}, {bellWidth}, {bestBins}, {sensitivity}]];
range = sprintf(rangeSpecifier, testCaseIndex+1, testCaseIndex+1);
xlswrite(filename, data, mainSheet, range);

% Detailed Data
message = sprintf('Saving results in %s', filename);
disp(message);

detailSheet = testName;
xlswrite(filename, sensitivityTable, detailSheet);
result = bestBins;

%%%%%%%%%%%%%
% LFPBestBell - Determina o melhor valor de sigma para a gaussiana empregada
% pelo LFP em determinado caso de teste.
%%%%%%%%%%%%%
function result = LFPBestBell(imgSet, testCaseIndex, testInUse, janela,
numNeighbors, radius, numBins, fileVersion, lfpSamplingFunction)

testPath = testInUse{1};
numImages = testInUse{3};

```

```

classifier = testInUse{6};

fprintf('\n');
disp('=====');
message = sprintf('LFP Best Bell Width - %s, winSize: %d, numBins: %d, iteration: %d\n', testPath, janela, numBins, fileVersion);
disp(message);

bellWidthList = [0.5:0.1:1, 2:0.5:15]';
threshold = 0;

lfp_hist = zeros(numImages, numBins);
sensitivityTable = zeros(length(bellWidthList), 3);

for bellWidthIndex = 1 : size(bellWidthList);
    bellWidth = bellWidthList(bellWidthIndex);

    % Generates LFP matrix
    lfp_matrix = BuildLFPMatrix(imgSet, janela, numNeighbors, radius, 0, bellWidth, lfpSamplingFunction);

    % Generates LFP histogram
    disp('Generating LFP histogram...');

    tic
    for imgIndex = 1 : numImages
        lfp_hist(imgIndex, :) = LFP_Hist(lfp_matrix{imgIndex}, numBins);
    end;
    toc

    % Classification process
    resultClassifier = ClassifyData(testCaseIndex, testInUse, lfp_hist, threshold);
    sensitivityTable(bellWidthIndex, 1) = bellWidth;
    sensitivityTable(bellWidthIndex, 2) = resultClassifier(:, 4);
end;

% Display the results
disp('-----');
disp('LFP Best Bell Width');
message = sprintf('Test case: %s', testPath);
disp(message);
message = sprintf('Num bins: %d', numBins);
disp(message);
fprintf('\n');

[sensitivity row] = max(sensitivityTable(:,2));
bestBell = sensitivityTable(row, 1);

classifierName = 'Chi-square';
if (classifier == 1)
    classifierName = 'Log-likelihood';
end;

message = sprintf('%s - Best bell width: %f, sensitivity: %f', classifierName, bestBell, sensitivity);
disp(message);

fprintf('\n');
disp('Sensitivity table:');
printf('\tBell width\tSensitivity\n');
disp(sensitivityTable);
disp('-----');

% Results
slashOccurr = strfind(testPath, '\\');
numOccurr = size(slashOccurr, 2);

```

```

testName = testPath(slashOccurr(numOccurr-1)+2 : slashOccurr(numOccurr)-1);

% Plot and Saves Sensitivity Graph
h = figure;
plot(sensitivityTable(:,1), sensitivityTable(:,2));
title(classifierName);
ylabel('Sensitivity');
xlabel('Bell width');
grid;
fileName = sprintf('%s_Bell_classifier%d_func%d_ver%d', testName, classifier,
lfpSamplingFunction, fileVersion);
saveas(h, fileName, 'jpg');
close(h);

% Saves the results in an excel file
filename = sprintf('Outex_BestBell_classifier%d_func%d_ver%d.xls', classifier,
lfpSamplingFunction, fileVersion);
mainSheet = 'Best Bell';
rangeSpecifier = 'A%d:D%d';

% Header
data = {[{'Test Name'}, {'Bins'}, {'Best Bell'}, {'Sensitivity'}]};
range = sprintf(rangeSpecifier, 1, 1);
xlswrite(filename, data, mainSheet, range);

% Summary
data = {[{testName}, {numBins}, {bestBell}, {sensitivity}]];
range = sprintf(rangeSpecifier, testCaseIndex+1, testCaseIndex+1);
xlswrite(filename, data, mainSheet, range);

% Detailed Data
message = sprintf('Saving results in %s', filename);
disp(message);

detailSheet = testName;
xlswrite(filename, sensitivityTable, detailSheet);
result = bestBell;

%%%%%%%%%%%%%
% ClassifierChiSquare - Faz a classificacao de um conjunto de teste
% utilizando a metrica chi-quadrado.
%%%%%%%%%%%%%
function result = ClassifierChiSquare(trains, tests, trainClassIDs, testClassIDs,
thresholds)

trainNum = size(trains, 1);
testNum = size(tests, 1);
DM = zeros(testNum, trainNum);

for i = 1 : testNum;
    test = tests(i,:);
    DM(i,:) = distMATCHiSquare(trains, test)';
end;

if (isempty(thresholds))
    thresholds = 0;
end;

rocData = zeros(length(thresholds), 3);

for i = 1 : length(thresholds)
    classificationResult = ClassifyOnNN(DM, trainClassIDs, testClassIDs,
thresholds(i));

```

```

rocData(i, 1) = thresholds(i);
rocData(i, 2) = classificationResult(1); % min distance
rocData(i, 3) = classificationResult(2); % max distance
rocData(i, 4) = classificationResult(3); % sensitivity
rocData(i, 5) = classificationResult(4); % specificity
end;

result = rocData;

%%%%%%%%%%%%%
% ClassifierLogLikelihood - Faz a classificacao de um conjunto de teste
% utilizando a metrica estatistica G.
%%%%%%%%%%%%%
function result = ClassifierLogLikelihood(trains, tests, trainClassIDs,
testClassIDs, thresholds)

trainNum = size(trains, 1);
testNum = size(tests, 1);
DM = zeros(testNum, trainNum);

for i = 1 : testNum;
    test = tests(i,:);
    DM(i,:) = LogLikelihood(trains, test)';
end;

if isempty(thresholds)
    thresholds = 0;
end;

rocData = zeros(length(thresholds), 3);

for i = 1 : length(thresholds)
    classificationResult = ClassifyOnNN(DM, trainClassIDs, testClassIDs,
thresholds(i));

    rocData(i, 1) = thresholds(i);
    rocData(i, 2) = classificationResult(1); % min distance
    rocData(i, 3) = classificationResult(2); % max distance
    rocData(i, 4) = classificationResult(3); % sensitivity
    rocData(i, 5) = classificationResult(4); % specificity
end;

result = rocData;

%%%%%%%%%%%%%
% ClassifyData - Executa o processo de classificacao de um conjunto de
% teste.
% classifier:
% 0 - ChiSquare, 1 - Log likelihood
%%%%%%%%%%%%%
function resultMatrix = ClassifyData(testCaseIndex, testInUse, histogram,
thresholdList)

testPath = testInUse{1};
testList = testInUse{2}{1};
summarizeHist = testInUse{5};
classifier = testInUse{6};
numTests = size(testList, 2);

```

```
% Classification process
minDist = intmax;
maxDist = 0;

classifierName = 'Chi-square';
if (classifier == 1)
    classifierName = 'Log likelihood';
end

message = sprintf('Classifying data - Classifier: %s', classifierName);
disp(message);
tic
resultMatrix = zeros(length(thresholdList), 5);
disp(' ');

for testIndex = 1 : numTests
    numTest = testList(testIndex);

    % read picture ID of training and test samples, and read class ID
    % of training and test samples
    trainData = sprintf('%s%02d\\train.txt', testPath, numTest-1);
    testData = sprintf('%s%02d\\test.txt', testPath, numTest-1);
    [trainIDs, trainClassIDs] = ReadOutexTxt(testCaseIndex, testInUse, trainData);
    [testIDs, testClassIDs] = ReadOutexTxt(testCaseIndex, testInUse, testData);

    % classification
    trains = histogram(trainIDs,:);
    tests = histogram(testIDs,:);

    % If summarizeHist == true, all histograms belonging to a class will
    % summarized into one same histogram.
    if summarizeHist == true
        newTrainClassIDs = unique(trainClassIDs);
        newTrains = zeros(size(newTrainClassIDs, 2), size(trains, 2)); %
numClasses, numBins
        for i = 1 : length(trainClassIDs)
            trainClassId = trainClassIDs(i);

            newTrainClassIDsIndex = find(newTrainClassIDs == trainClassId);
            newTrains(newTrainClassIDsIndex, :) = newTrains(newTrainClassIDsIndex,
:) + trains(i, :);
        end;

        for i = 1 : size(newTrains, 1)
            newTrains(i, :) = newTrains(i, :)/sum(newTrains(i, :));
        end;
    else
        newTrainClassIDs = trainClassIDs;
        newTrains = trains;
    end;

    if (classifier == 0)
        classificationResult = ClassifierChiSquare(newTrains, tests,
newTrainClassIDs, testClassIDs, thresholdList);
    else
        classificationResult = ClassifierLogLikelihood(newTrains, tests,
newTrainClassIDs, testClassIDs, thresholdList);
    end;

    tempMinDist = classificationResult(:, 2);
    tempMaxDist = classificationResult(:, 3);

    sensitivity = classificationResult(:, 4);
    specificity = classificationResult(:, 5);

```

```

message = sprintf('Test: %d', numTest);
disp(message);
message = sprintf('sensitivity: %f', sensitivity);
disp(message);

minDist = min(minDist, tempMinDist);
maxDist = max(maxDist, tempMaxDist);

resultMatrix = resultMatrix + classificationResult;
end;
endTime = toc;

resultMatrix = resultMatrix / numTests;
resultMatrix(:, 2) = minDist;
resultMatrix(:, 3) = maxDist;

%%%%%%%%%%%%%%%
% LoadOutexImages - carrega os arquivos de imagens.
%%%%%%%%%%%%%%%
function imageSet = LoadOutexImages(testCaseIndex, testData)

testPath = testData{1};
numImages = testData{3};
imageSet = cell(numImages, 1);
imagePath = '%s\\images\\%06d.ras';
convertToGray = false;

bmpTest(1) = {'Contrib_TC_00006'};
bmpTest(2) = {'Contrib_TC_00013'};
bmpTest(3) = {'Contrib_TC_00014'};

for i = 1 : length(bmpTest)
    bmpTestPath = bmpTest{i};

    idx = strfind(testPath, bmpTestPath);

    if length(idx) > 0
        imagePath = '%s\\images\\%06d.bmp';
        convertToGray = true;
    end
end;

disp('Loading images...');

normalize = (strfind(testPath, 'Outex_TC_00012') == 0);

for imgIndex = 1 : numImages
    filename = sprintf(imagePath, testPath, imgIndex-1);

    img = imread(filename);

    if (convertToGray == true)
        img = rgb2gray(img);
    end;

    img = im2double(img);

    if normalize
        img = (img-mean(img(:)))/std(img(:))*20+128; % image normalization, to
remove global intensity
    end;

```

```

imageSet{imgIndex} = img;
end;

%%%%%%%%%%%%%%%
% Script principal para execucao dos testes.
%%%%%%%%%%%%%%%
clc; close all; fclose all;
disp('Running OutexTest...');

% Outex data
databasePath = 'C:\\\\Users\\\\Carlos\\\\Documents\\\\Mestrado\\\\Banco de imagens\\\\';
%
% testCase:
% test path, number of tests, number of pictures in the database, operator,
% summarize histogram
%
testCase = {};
%
% testCase(:, :) = {
% test path: string
% {suit test list}: an array with the list of test cases
% number of images: integer
% LBP mode: 0 - classic, 1 - ri, 2 - riu2
% sum histograms: boolean indicate to sum all histograms into a big one
% classifier: 0 - chi-square, 1 - log likelihood
% }
%
testCase(1, :) = {strcat(databasePath, 'Contrib_TC_00000\\\\'), {1}, 8416, 2,
true, 0}; % experimento 1 de Brodatz
testCase(2, :) = {strcat(databasePath, 'Contrib_TC_00001\\\\'), {1:10}, 20480, 2,
true, 0};
testCase(3, :) = {strcat(databasePath, 'Contrib_TC_00002\\\\'), {1:7}, 6720, 2,
true, 0}; % experimento 2 de Brodatz
testCase(4, :) = {strcat(databasePath, 'Contrib_TC_00003\\\\'), {1:7}, 1680, 2,
true, 0}; % experimento 3 de Brodatz
testCase(5, :) = {strcat(databasePath, 'Contrib_TC_00004\\\\'), {1:10}, 2048, 1,
true, 0};
testCase(6, :) = {strcat(databasePath, 'Contrib_TC_00005\\\\'), {1:100}, 704, 1,
true, 0};
testCase(7, :) = {strcat(databasePath, 'Contrib_TC_00006\\\\'), {1}, 864, 1,
true, 0};
testCase(8, :) = {strcat(databasePath, 'Outex_TC_00000\\\\'), {1:50.0}, 480, 1,
false, 0};
testCase(9, :) = {strcat(databasePath, 'Outex_TC_00001\\\\'), {1:10.0}, 2112, 1,
false, 0};
testCase(10, :) = {strcat(databasePath, 'Outex_TC_00002\\\\'), {1:10.0}, 8832, 1,
false, 0};
testCase(11, :) = {strcat(databasePath, 'Outex_TC_00003\\\\'), {1:50.0}, 480, 1,
false, 0};
testCase(12, :) = {strcat(databasePath, 'Outex_TC_00004\\\\'), {1:10.0}, 2112, 1,
false, 0};
testCase(13, :) = {strcat(databasePath, 'Outex_TC_00005\\\\'), {1:10.0}, 8832, 1,
false, 0};
testCase(14, :) = {strcat(databasePath, 'Outex_TC_00006\\\\'), {1:10.0}, 2112, 1,
false, 0};
testCase(15, :) = {strcat(databasePath, 'Outex_TC_00007\\\\'), {1:10.0}, 8832, 1,
false, 0};
testCase(16, :) = {strcat(databasePath, 'Outex_TC_00008\\\\'), {1:10.0}, 2112, 1,
false, 0};
testCase(17, :) = {strcat(databasePath, 'Outex_TC_00009\\\\'), {1:10.0}, 8832, 1,
false, 0};
testCase(18, :) = {strcat(databasePath, 'Outex_TC_00010\\\\'), {1}, 4320, 2,
false, 0}; % 128x128 pixels

```

```

testCase(19, :) = {strcat(databasePath, 'Outex_TC_00011\\'), {1}, 960, 1,
false, 0};
testCase(20, :) = {strcat(databasePath, 'Outex_TC_00012\\'), {1:2}, 9120, 2,
false, 0};
testCase(21, :) = {strcat(databasePath, 'Outex_TC_00013\\'), {1}, 1360, 1,
false, 0}; % BMP images
testCase(22, :) = {strcat(databasePath, 'Outex_TC_00014\\'), {1}, 4080, 1,
false, 0}; % BMP images
testCase(23, :) = {strcat(databasePath, 'Outex_TC_00015\\'), {1:10.0}, 1360, 1,
false, 0};
testCase(24, :) = {strcat(databasePath, 'Outex_TC_00016\\'), {1:10.0}, 6380, 1,
false, 0};
testCase(25, :) = {strcat(databasePath, 'Vistex\\')}, {1:8}, 17280, 2,
false, 0};
testCase(26, :) = {strcat(databasePath, 'Outex_TC_00010\\TC10-SplitedImagesTest-
32\\'), {1}, 69120, 2, false, 0}; % 32x32 pixels
testCase(27, :) = {strcat(databasePath, 'Outex_TC_00010\\TC10-SplitedImagesTest-
64\\'), {1}, 17280, 2, false, 0}; % 64x64 pixels
testCase(28, :) = {strcat(databasePath, 'Outex_TC_00012\\TC12-SplitedImagesTest-
32\\'), {1:2}, 145920, 2, false, 0}; % 32x32 pixels
testCase(29, :) = {strcat(databasePath, 'Outex_TC_00012\\TC12-SplitedImagesTest-
64\\'), {1:2}, 36480, 2, false, 0}; % 64x64 pixels
testCase(30, :) = {strcat(databasePath,
'Contrib_TC_00002\\Contrib_TC_02_ByAngle\\'), {1:6}, 6720, 2, true, 0}; % usado na
dissertacao
testCase(31, :) = {strcat(databasePath,
'Contrib_TC_00003\\Contrib_TC_03_ByAngle\\'), {1:6}, 1680, 2, true, 0}; % usado na
dissertacao

% Selected test cases to run
testCaseList = [1, 30, 31, 18, 20, 27, 29];

% LFP
lfpWindowSize = 0; % for use with the LFP sigmoidal (classic form)
lfpNumBins = 256;
lfpBeta = 1.005;
lfpBellWidth = 1.005;

% Parametros utilizados em cada caso de teste (beta8x1; numBins8x1;
% beta16x2; numBins16x2; beta24x3; numBins24x3)
lfpTestParams = {};
lfpTestParams(1, :) = {0.6, 150, 0.6, 160, 2.5, 190};
lfpTestParams(2, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(3, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(4, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(5, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(6, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(7, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(8, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(9, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(10, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(11, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(12, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(13, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(14, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(15, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(16, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(17, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(18, :) = {1.55, 130, 0.85, 30, 0.7, 256};
lfpTestParams(19, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(20, :) = {1.55, 130, 0.85, 30, 0.7, 256};
lfpTestParams(21, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(22, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(23, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(24, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(25, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(26, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(27, :) = {0.75, 90, 0.35, 70, 0.35, 256};

```

```

lfpTestParams(28, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(29, :) = {0.75, 90, 0.35, 70, 0.35, 256};
lfpTestParams(30, :) = {1.25, 256, 0.75, 256, 0.95, 256};
lfpTestParams(31, :) = {0.8, 256, 0.2, 256, 0.1, 256};
lfpTestParams(32, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(33, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(34, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(35, :) = {0, 0, 0, 0, 0, 0};
lfpTestParams(36, :) = {0, 0, 0, 0, 0, 0};

% Sampling function:
% 1 - Sigmoidal
% 2 - Gaussian
% 3 - Triangular
% 4 - LFP Sampling - Sigmoidal
% 5 - LFP_Q - Sigmoidal, com vizinhanca quadrangular a partir de um raio
%
lfpSamplingFunction = 4;

% Neighborhood settings
numNeighbors = 8;
neighborRadius = 1;

% Test parameters
findBestValues = false;
finingCycles = 2;

% Test algorithm
%
if lfpSamplingFunction == 2 % Gaussian
    lfpBeta = 0;
else
    lfpBellWidth = 0;
end;

for i = 1 : length(testCaseList)
    testCaseIndex = testCaseList(i);
    selectedTestCase = testCase(testCaseIndex, :);

    testPath = selectedTestCase{1};
    slashOccurr = strfind(testPath, '\\');
    numOccurr = size(slashOccurr, 2);
    testName = testPath(slashOccurr(numOccurr-2)+2 : slashOccurr(numOccurr)-1);
    message = sprintf('Test name: %s', testName);
    disp(' ');
    disp(' ');
    disp('******');
    disp('******');
    disp(message);

    %%%%%%
    % Load Images
    %
    imageSet = LoadOuttexImages(testCaseIndex, selectedTestCase);

    %%%%%%
    % Execute tests
    %
    if findBestValues == false
        if numNeighbors == 8
            lfpBeta = lfpTestParams{testCaseIndex, 1};
            lfpNumBins = lfpTestParams{testCaseIndex, 2};
        elseif numNeighbors == 16
            lfpBeta = lfpTestParams{testCaseIndex, 3};
            lfpNumBins = lfpTestParams{testCaseIndex, 4};
        elseif numNeighbors == 24
    end;

```

```

lfpBeta = lfpTestParams{testCaseIndex, 5};
lfpNumBins = lfpTestParams{testCaseIndex, 6};
else
    disp('Resolution not supported!');
    exit(0);
end;

LBPRference(imageSet, testCaseIndex, selectedTestCase, numNeighbors,
neighborRadius);
LFPReference(imageSet, testCaseIndex, selectedTestCase, lfpWindowSize,
numNeighbors, neighborRadius, lfpBeta, lfpBellWidth, lfpNumBins,
lfpSamplingFunction);
end;

if findBestValues == true
    % set parameters to default
    beta = lfpBeta;
    bell = lfpBellWidth;
    bins = lfpNumBins;

    for cycleNumber = 1 : finingCycles
        %
        switch lfpSamplingFunction
            case 2 % Gaussian function - best bell width
                bestBellWidths = LFPBestBell(imageSet, testCaseIndex,
selectedTestCase, lfpWindowSize, numNeighbors, neighborRadius, bins, cycleNumber,
lfpSamplingFunction);
                bell = bestBellWidths(1);
            otherwise % best beta
                bestBetas = LFPBestBeta(imageSet, testCaseIndex,
selectedTestCase, lfpWindowSize, numNeighbors, neighborRadius, bins, cycleNumber,
lfpSamplingFunction);
                beta = bestBetas(1);
            end;
        %

        bestBins = LFPBestBins(imageSet, testCaseIndex, selectedTestCase,
lfpWindowSize, numNeighbors, neighborRadius, beta, bell, cycleNumber,
lfpSamplingFunction);
        bins = bestBins;
    end;
end;

%ROCCurve(imageSet, testCaseIndex, selectedTestCase, lbpNumSamples, lbpRadius,
lfpWindowSize, lfpNumNeighbors, neighborRadius, lfpBeta, lfpBellWidth, lfpNumBins,
lfpSamplingFunction);
end;

% Finish the test
disp('*'*');
fprintf('\n');
disp('OutexTest finished.');
beep;
%%%%%%%%%%%%%

```



## APÊNDICE C – CÓDIGO FONTE EXCLUSIVO PARA TESTE COM O ÁLBUM DE BRODATZ E OUTEX

```
%%%%%%%%%%%%%%%%
% LFPReference - Executa um teste de referencia do LFP em um caso de teste.
%%%%%%%%%%%%%%%
function LFPReference(imageSet, testCaseIndex, testInUse, windowSize, numNeighbors,
radius, beta, bellWidth, numBins, lfpSamplingFunction)

testPath = testInUse{1};
numImages = testInUse{3};
classifier = testInUse{6};

% Generates LFP code matrix
disp('Generating LFP matrix...');
tic;
lfp_matrix = BuildLFPMatrix(imageSet, windowSize, numNeighbors, radius, beta,
bellWidth, lfpSamplingFunction);
lfpCodeMatrixTime = toc;

message = sprintf('LFP code matrix built in: %f seconds.', lfpCodeMatrixTime);
disp(message);

% calculates sensitivity
lfp_hist = zeros(numImages, numBins);

disp('Generating LFP histogram...');
tic
for imgIndex = 1 : numImages
    lfp_hist(imgIndex, :) = LFP_Hist(lfp_matrix{imgIndex}, numBins);
end;
lfpHistTime = toc;

message = sprintf('LFP histograms built in: %f seconds.', lfpHistTime);
disp(message);

% Classification process
threshold = 0;
resultsClassifier = ClassifyData(testCaseIndex, testInUse, lfp_hist, threshold);

% Display the results
disp('-----');
slashOccurr = strfind(testPath, '\\');
numOccurr = size(slashOccurr, 2);
testName = testPath(slashOccurr(numOccurr-2) + 2 : slashOccurr(numOccurr)-1);

message = sprintf('LFP Reference Results - %s\n', testName);
disp(message);
message = sprintf('Neighbor size: %d', windowSize);
disp(message);
message = sprintf('Num Neighbors: %d', numNeighbors);
disp(message);
message = sprintf('Radius: %d', radius);
disp(message);
message = sprintf('Sampling function: %d', lfpSamplingFunction);
disp(message);
message = sprintf('Bell width: %f', bellWidth);
disp(message);
message = sprintf('Beta: %f', beta);
disp(message);
message = sprintf('Number of bins: %d', numBins);
disp(message);
```

```

message = sprintf('LFP code matrix built in: %f seconds.', lfpCodeMatrixTime);
disp(message);
message = sprintf('LFP histograms built in: %f seconds.', lfpHistTime);
disp(message);

classifierName = 'Chi-Square';
if (classifier == 1)
    classifierName = 'Log likelihood';
end

fprintf('\n');
disp(classifierName);
message = sprintf('Min distance: %f', resultsClassifier(1, 2));
disp(message);
message = sprintf('Max distance: %f', resultsClassifier(1, 3));
disp(message);
message = sprintf('Sensitivity: %f', resultsClassifier(1, 4));
disp(message);
message = sprintf('Specificity: %f', resultsClassifier(1, 5));
disp(message);
disp('-----');

% Save the results
filename = 'OutexReference.xls';
sheet = sprintf('LFP_Reference_f%d', lfpSamplingFunction);
rangeSpecifier = 'A%d:N%d';

% Header
data = {[{'Test Name'}, {'Neighbor size (classic LFP)'}, {'Num Neighbors'},
{'Radius'}, {'Beta'}, {'Number of bins'}, {'LFP code build time'}, {'Histogram
build time'}, {'Min Dist (Chi-Sqr)'}, {'Max Dist (Chi-Sqr)'}, {'Sensitivity (Chi-
Sqr)'}, {'Specificity (Chi-Sqr)'}, {'Min Dist (LogLik)'}, {'Max Dist (LogLik)'},
{'Sensitivity (LogLik)'}, {'Specificity (LogLik)'}];
range = sprintf(rangeSpecifier, 1, 1);
xlswrite(filename, data, sheet, range);

% Data
[rows, cols] = size(resultsClassifier);
zeroResults = zeros(rows, cols);

if (classifier == 0)
    resultsChiSquare = resultsClassifier;
    resultsLogLikelihood = zeroResults;
else
    resultsChiSquare = zeroResults;
    resultsLogLikelihood = resultsClassifier;
end

message = sprintf('Saving results in %s', filename);
disp(message);

data = [{testPath}, {windowSize}, {numNeighbors}, {radius}, {beta}, {numBins},
{lfpCodeMatrixTime}, {lfpHistTime}, {resultsChiSquare(1, 2)}, {resultsChiSquare(1,
3)}, {resultsChiSquare(1, 4)}, {resultsChiSquare(1, 5)}, {resultsLogLikelihood(1,
2)}, {resultsLogLikelihood(1, 3)}, {resultsLogLikelihood(1, 4)},
{resultsLogLikelihood(1, 5)}];
range = sprintf(rangeSpecifier, testCaseIndex+1, testCaseIndex+1);
xlswrite(filename, data, sheet, range);

%%%%%%%%%%%%%%%
% LBPReference - Executa um teste de refencia do LBP em um caso de teste.
%%%%%%%%%%%%%%%
function LBPReference(imageSet, testCaseIndex, testInUse, numSamples, radius)

```

```

testPath = testInUse{1};
numImages = testInUse{3};
classifier = testInUse{6};

numBins = 256;
lbpMode = testInUse{4};

switch numSamples
    case 8
        if (lbpMode == 1) % ri
            numBins = 36;
        else if (lbpMode == 2) % riu2
            numBins = 10;
        else
            error('Invalid LBP mode (selected 8 samples)');
        end
    end

    case 16
        if (lbpMode == 1) % ri
            numBins = 4116;
        else if (lbpMode == 2) % riu2
            numBins = 18;
        else
            error('Invalid LBP mode (selected 8 samples)');
        end
    end

    case 24
        if (lbpMode == 1) % ri
            numBins = 699252;
        else if (lbpMode == 2) % riu2
            numBins = 26;
        else
            error('Invalid LBP mode (selected 16 samples)');
        end
    end
end

% Generates LBP code matrix
disp('Generating LBP matrix...');

tic;
lbp_matrix = BuildLBPMatrix(imageSet, numSamples, radius, lbpMode);
lbpCodeMatrixTime = toc;

message = sprintf('LBP code matrix built in: %f seconds.', lbpCodeMatrixTime);
disp(message);

% calculates sensitivity
lbp_hist = zeros(numImages, numBins);

disp('Generating LBP histogram...');

tic;
for imgIndex = 1 : numImages
    lbp_hist(imgIndex, :) = LBP_Hist(lbp_matrix{imgIndex}, numBins);
end;
lbpHistTime = toc;

message = sprintf('LBP histograms built in: %f seconds.', lbpHistTime);
disp(message);

% Classification process
threshold = 0;
resultsClassifier = ClassifyData(testCaseIndex, testInUse, lbp_hist, threshold);

% Display the results

```

```

disp('-----');
slashOccurr = strfind(testPath, '\\');
numOccurr = size(slashOccurr, 2);
testName = testPath(slashOccurr(numOccurr-2)+2 : slashOccurr(numOccurr)-1);

message = sprintf('LBP Reference Results - %s\n', testName);
disp(message);
message = sprintf('Neighbor size: %d', numSamples);
disp(message);
message = sprintf('Radius: %f', radius);
disp(message);

switch lbpMode
    case 0
        disp('LBP mode: classic');
    case 1
        disp('LBP mode: ri');
    case 2
        disp('LBP mode: riu2');
end;

message = sprintf('LBP code matrix built in: %f seconds.', lbpCodeMatrixTime);
disp(message);
message = sprintf('LBP histograms built in: %f seconds.', lbpHistTime);
disp(message);

classifierName = 'Chi-Square';
if (classifier == 1)
    classifierName = 'Log likelihood';
end

fprintf('\n');
disp(classifierName);
%message = sprintf('Min distance: %f', resultsClassifier(1, 2));
%disp(message);
%message = sprintf('Max distance: %f', resultsClassifier(1, 3));
%disp(message);
message = sprintf('Sensitivity: %f', resultsClassifier(1, 4));
disp(message);
%message = sprintf('Specificity: %f', resultsClassifier(1, 5));
%disp(message);
disp('-----');

% Save the results
filename = 'OutexReference.xls';
sheet = sprintf('LBP_Reference_f%d', lbpMode);
rangeSpecifier = 'A%d:N%d';

% Header
data = [{"Test Name"}, {"Num Samples"}, {"Radius"}, {"LBP Mode"}, {"LBP Code build time"}, {"Histogram build time"}, {"Min Dist (Chi-Sqr)"}, {"Max Dist (Chi-Sqr)"}, {"Sensitivity (Chi-Sqr)"}, {"Specificity (Chi-Sqr)"}, {"Min Dist (LogLik)"}, {"Max Dist (LogLik)"}, {"Sensitivity (LogLik)"}, {"Specificity (LogLik)"}];
range = sprintf(rangeSpecifier, 1, 1);
xlswrite(filename, data, sheet, range);

% Data
[rows, cols] = size(resultsClassifier);
zeroResults = zeros(rows, cols);

if (classifier == 0)
    resultsChiSquare = resultsClassifier;
    resultsLogLikelihood = zeroResults;
else
    resultsChiSquare = zeroResults;
    resultsLogLikelihood = resultsClassifier;
end

```

```

end

message = sprintf('Saving results in %s', filename);
disp(message);

data = [{testPath}, {numSamples}, {radius}, {lbpMode}, {lbpCodeMatrixTime},
{lbpHistTime}, {resultsChiSquare(1, 2)}, {resultsChiSquare(1, 3)},
{resultsChiSquare(1, 4)}, {resultsChiSquare(1, 5)}, {resultsLogLikelihood(1, 2)},
{resultsLogLikelihood(1, 3)}, {resultsLogLikelihood(1, 4)},
{resultsLogLikelihood(1, 5)}];
range = sprintf(rangeSpecifier, testCaseIndex+1, testCaseIndex+1);
xlswrite(filename, data, sheet, range);

%%%%%%%%%%%%%
% ClassifierChiSquare - Faz a classificacao de um conjunto de teste
% utilizando a metrica chi-quadrado.
%%%%%%%%%%%%%
function result = ClassifierChiSquare(testInUse, trains, tests, trainClassIDs,
testClassIDs, thresholds)

summarizeHist = testInUse{5};

trainNum = size(trains, 1);
testNum = size(tests, 1);
DM = zeros(testNum, trainNum);

for i = 1 : testNum;
    test = tests(i,:);
    DM(i,:) = distMATChiSquare(trains, test)';
end;

% Remove as comparacoes diretas entre a mesma amostra
if summarizeHist == false
    for i = 1 : size(DM, 1)
        for j = 1 : size(DM, 1)
            if i == j
                DM(i, j) = 999;
            end
        end;
    end
end

if (isempty(thresholds))
    thresholds = 0;
end;

rocData = zeros(length(thresholds), 3);

for i = 1 : length(thresholds)
    classificationResult = ClassifyOnNN(DM, trainClassIDs, testClassIDs,
thresholds(i));

    rocData(i, 1) = thresholds(i);
    rocData(i, 2) = classificationResult(1); % min distance
    rocData(i, 3) = classificationResult(2); % max distance
    rocData(i, 4) = classificationResult(3); % sensitivity
    rocData(i, 5) = classificationResult(4); % specificity
end;

result = rocData;

```

```

%%%%%%%%%%%%%%%
% ClassifierChiSquare - Faz a classificacao de um conjunto de teste
% utilizando a metrica chi-quadrado.
%%%%%%%%%%%%%%%
function result = ClassifierLogLikelihood(testInUse, trains, tests, trainClassIDs,
testClassIDs, thresholds)

summarizeHist = testInUse{5};

trainNum = size(trains, 1);
testNum = size(tests, 1);
DM = zeros(testNum, trainNum);

for i = 1 : testNum;
    test = tests(i,:);
    DM(i,:) = LogLikelihood(trains, test)';
end;

% Remove as comparacoes diretas entre a mesma amostra
if summarizeHist == false
    for i = 1 : size(DM, 1)
        for j = 1 : size(DM, 1)
            if i == j
                DM(i, j) = 999;
            end
        end;
    end
end

if (isempty(thresholds))
    thresholds = 0;
end;

rocData = zeros(length(thresholds), 3);

for i = 1 : length(thresholds)
    classificationResult = ClassifyOnNN(DM, trainClassIDs, testClassIDs,
thresholds(i));

    rocData(i, 1) = thresholds(i);
    rocData(i, 2) = classificationResult(1); % min distance
    rocData(i, 3) = classificationResult(2); % max distance
    rocData(i, 4) = classificationResult(3); % sensitivity
    rocData(i, 5) = classificationResult(4); % specificity
end;

result = rocData;

%%%%%%%%%%%%%%%
% ClassifyData - Executa o processo de classificacao de um conjunto de
% teste.
% classifier:
% 0 - ChiSquare, 1 - Log likelihood
%%%%%%%%%%%%%%%
function resultMatrix = ClassifyData(testCaseIndex, testInUse, histogram,
thresholdList)

testPath = testInUse{1};
testList = testInUse{2}{1};
summarizeHist = testInUse{5};
classifier = testInUse{6};
numTests = 1;

```

```
% Classification process
minDist = intmax;
maxDist = 0;

classifierName = 'Chi-square';
if (classifier == 1)
    classifierName = 'Log likelihood';
end

message = sprintf('Classifying data - Classifier: %s', classifierName);
disp(message);
tic
resultMatrix = zeros(length(thresholdList), 5);
disp(' ');

for testIndex = 1 : numTests
    numTest = testList(testIndex);

    % read picture ID of training and test samples, and read class ID
    % of training and test samples
    trainData = sprintf('%s%03d\\train.txt', testPath, numTest-1);
    testData = trainData;

    [trainIDs, trainClassIDs] = ReadOutexTxt(testCaseIndex, testInUse, trainData);
    [testIDs, testClassIDs] = ReadOutexTxt(testCaseIndex, testInUse, testData);

    % classification
    trains = histogram(:, :, :);
    tests = histogram(:, :, :);

    % If sumerizeHist == true, all histograms belonging to a class will
    % summarized into one same histogram.
    if summarizeHist == true
        newTrainClassIDs = unique(trainClassIDs);
        newTrains = zeros(size(newTrainClassIDs, 2), size(trains, 2)); %
numClasses, numBins

        for i = 1 : length(trainClassIDs)
            trainClassId = trainClassIDs(i);

            newTrainClassIDsIndex = find(newTrainClassIDs == trainClassId);
            newTrains(newTrainClassIDsIndex, :) = newTrains(newTrainClassIDsIndex,
:) + trains(i, :);
        end;

        for i = 1 : size(newTrains, 1)
            newTrains(i, :) = newTrains(i, :) / sum(newTrains(i, :));
        end;
    else
        newTrainClassIDs = trainClassIDs;
        newTrains = trains;
    end;

    if (classifier == 0)
        classificationResult = ClassifierChiSquare(testInUse, newTrains, tests,
newTrainClassIDs, testClassIDs, thresholdList);
    else
        classificationResult = ClassifierLogLikelihood(testInUse, newTrains, tests,
newTrainClassIDs, testClassIDs, thresholdList);
    end;

    tempMinDist = classificationResult(:, 2);
    tempMaxDist = classificationResult(:, 3);

```

```

sensitivity = classificationResult(:, 4);
specificity = classificationResult(:, 5);

message = sprintf('Test: %d', numTest);
disp(message);
message = sprintf('sensitivity: %f', sensitivity);
disp(message);

minDist = min(minDist, tempMinDist);
maxDist = max(maxDist, tempMaxDist);

resultMatrix = resultMatrix + classificationResult;
end;
endTime = toc;

resultMatrix = resultMatrix / numTests;
resultMatrix(:, 2) = minDist;
resultMatrix(:, 3) = maxDist;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LoadOutexImages - carrega os arquivos de imagens.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function imageSet = LoadOutexImages(testCaseIndex, testData)

testPath = testData{1};
numImages = testData{3};
imagePath = '%s\images\%06d.ras';
convertToGray = false;

bmpTest(1) = {'Contrib_TC_00006'};
bmpTest(2) = {'Contrib_TC_00013'};
bmpTest(3) = {'Contrib_TC_00014'};

for i = 1 : length(bmpTest)
    bmpTestPath = bmpTest{i};

    idx = strfind(testPath, bmpTestPath);

    if length(idx) > 0
        imagePath = '%s\images\%06d.bmp';
        convertToGray = true;
    end
end;

% Monta lista de imagens a ser lida.
imgFileNameList = [];
numImgsFromFile = 0;

testFileName = sprintf('%s\000\train.txt', testPath);

if ~exist(testFileName, 'file')
    error('File not found: %s', testFileName);
end;

testFileDialog = fopen(testFileName, 'r');

if testFileDialog == -1
    error('Erro na abertura do arquivo de teste: %s', testFileName);
end;

firstLine = true;
while true

```

```

tline = fgetl(testFileDialog);

if ~ischar(tline)
    break;
end

if firstLine == true
    if length(strfind(tline, ' ')) == 0 % header - number of imgs
        continue;
    end
    firstLine = false;
end

separatorIndex = strfind(tline,' ');
sampleName = tline(1:separatorIndex-1);

if ~ismember(sampleName, imgFileNameList)
    numImgsFromFile = numImgsFromFile + 1;
    imgFileNameList{numImgsFromFile} = sampleName;
end;
end;

imageSet = cell(numImgsFromFile, 1);
disp('Loading images...');

normalize = (strfind(testPath, 'Outex_TC_00012') == 0);

for imgIndex = 1 : size(imgFileNameList, 2)
    filename = sprintf('%s\\images\\%s', testPath, imgFileNameList{imgIndex});

    img = imread(filename);

    if (convertToGray == true)
        img = rgb2gray(img);
    end;

    img = im2double(img);

    if normalize
        img = (img-mean(img(:)))/std(img(:))*20+128; % image normalization, to
remove global intensity
    end;

    imageSet{imgIndex} = img;
end;

%%%%%%%%%%%%%%%
% Script principal para execucao dos testes.
%%%%%%%%%%%%%%%
clc; close all; fclose all;
disp('Running OutexTest...');

% Outex data
databasePath = 'C:\\Users\\Carlos\\Documents\\Mestrado\\Banco de imagens\\';
%
% testCase:
% test path, number of tests, number of pictures in the database, operator,
% summarize histogram
%
testCase = {};
%
% testCase(:, :) = {
% test path: string

```

```
% {suit test list}: an array with the list of test cases
% number of images: integer
% LBP mode: 0 - classic, 1 - ri, 2 - riu2
% sum histograms: boolean indicate to sum all histograms into a big one
% classifier: 0 - chi-square, 1 - log likelihood
%
testCase(1, :) = {strcat(databasePath, 'Contrib_TC_00000\\'), {1}, 8416, 2,
true, 0}; % experimento 1 de Brodatz
testCase(2, :) = {strcat(databasePath, 'Contrib_TC_00001\\'), {1:10}, 20480, 2,
true, 0};
testCase(3, :) = {strcat(databasePath, 'Contrib_TC_00002\\'), {1:7}, 6720, 2,
true, 0}; % experimento 2 de Brodatz
testCase(4, :) = {strcat(databasePath, 'Contrib_TC_00003\\'), {1:7}, 1680, 2,
true, 0}; % experimento 3 de Brodatz
testCase(5, :) = {strcat(databasePath, 'Contrib_TC_00004\\'), {1:10}, 2048, 1,
true, 0};
testCase(6, :) = {strcat(databasePath, 'Contrib_TC_00005\\'), {1:100}, 704, 1,
true, 0};
testCase(7, :) = {strcat(databasePath, 'Contrib_TC_00006\\'), {1}, 864, 1,
true, 0};
testCase(8, :) = {strcat(databasePath, 'Outex_TC_00000\\'), {1:50.0}, 480, 1,
false, 0};
testCase(9, :) = {strcat(databasePath, 'Outex_TC_00001\\'), {1:10.0}, 2112, 1,
false, 0};
testCase(10, :) = {strcat(databasePath, 'Outex_TC_00002\\'), {1:10.0}, 8832, 1,
false, 0};
testCase(11, :) = {strcat(databasePath, 'Outex_TC_00003\\'), {1:50.0}, 480, 1,
false, 0};
testCase(12, :) = {strcat(databasePath, 'Outex_TC_00004\\'), {1:10.0}, 2112, 1,
false, 0};
testCase(13, :) = {strcat(databasePath, 'Outex_TC_00005\\'), {1:10.0}, 8832, 1,
false, 0};
testCase(14, :) = {strcat(databasePath, 'Outex_TC_00006\\'), {1:10.0}, 2112, 1,
false, 0};
testCase(15, :) = {strcat(databasePath, 'Outex_TC_00007\\'), {1:10.0}, 8832, 1,
false, 0};
testCase(16, :) = {strcat(databasePath, 'Outex_TC_00008\\'), {1:10.0}, 2112, 1,
false, 0};
testCase(17, :) = {strcat(databasePath, 'Outex_TC_00009\\'), {1:10.0}, 8832, 1,
false, 0};
testCase(18, :) = {strcat(databasePath, 'Outex_TC_00010\\'), {1}, 4320, 2,
false, 0}; % 128x128 pixels
testCase(19, :) = {strcat(databasePath, 'Outex_TC_00011\\'), {1}, 960, 1,
false, 0};
testCase(20, :) = {strcat(databasePath, 'Outex_TC_00012\\'), {1:2}, 9120, 2,
false, 0};
testCase(21, :) = {strcat(databasePath, 'Outex_TC_00013\\'), {1}, 1360, 1,
false, 0}; % BMP images
testCase(22, :) = {strcat(databasePath, 'Outex_TC_00014\\'), {1}, 4080, 1,
false, 0}; % BMP images
testCase(23, :) = {strcat(databasePath, 'Outex_TC_00015\\'), {1:10.0}, 1360, 1,
false, 0};
testCase(24, :) = {strcat(databasePath, 'Outex_TC_00016\\'), {1:10.0}, 6380, 1,
false, 0};
testCase(25, :) = {strcat(databasePath, 'Vistex\\'), {1:8}, 17280, 2,
false, 0};
testCase(26, :) = {strcat(databasePath, 'Outex_TC_00010\\TC10-SplitedImagesTest-
32\\'), {1}, 69120, 2, false, 0}; % 32x32 pixels
testCase(27, :) = {strcat(databasePath, 'Outex_TC_00010\\TC10-SplitedImagesTest-
64\\'), {1}, 17280, 2, false, 0}; % 64x64 pixels
testCase(28, :) = {strcat(databasePath, 'Outex_TC_00012\\TC12-SplitedImagesTest-
32\\'), {1:2}, 145920, 2, false, 0}; % 32x32 pixels
testCase(29, :) = {strcat(databasePath, 'Outex_TC_00012\\TC12-SplitedImagesTest-
64\\'), {1:2}, 36480, 2, false, 0}; % 64x64 pixels
testCase(30, :) = {strcat(databasePath,
'Contrib_TC_00002\\Contrib_TC_02_ByAngle\\'), {1:6}, 6720, 2, true, 0}; % usado na
dissertacao
```

```

testCase(31, :) = {strcat(databasePath,
'Contrib_TC_00003\\Contrib_TC_03_ByAngle\\'), {1:6}, 1680, 2, true, 0}; % usado na
dissertacao

% Selected test cases to run
testCaseList = [1, 30, 31, 18, 20, 27, 29];

% LFP
lfpWindowSize = 0; % for use with the LFP sigmoidal (classic form)
lfpNumNeighbors = 8;
lfpRadius = 1;
lfpNumBins = 256;
lfpBeta = 1.005;
lfpBellWidth = 1.005;

% Sampling function:
% 1 - Sigmoidal
% 2 - Gaussian
% 3 - Triangular
% 4 - LFP Sampling - Sigmoidal
% 5 - LFP_Q - Sigmoidal, com vizinhanca quadrangular a partir de um raio
%
lfpSamplingFunction = 4;

% LBP
lbpNumSamples = 8;
lbpRadius = 1;

% Test parameters
findBestValues = true;
finingCycles = 2;

% Test algorithm
%
if lfpSamplingFunction == 2 % Gaussian
    lfpBeta = 0;
else
    lfpBellWidth = 0;
end;

for i = 1 : length(testCaseList)
    testCaseIndex = testCaseList(i);
    selectedTestCase = testCase(testCaseIndex, :);

    testPath = selectedTestCase{1};
    slashOccurr = strfind(testPath, '\\');
    numOccurr = size(slashOccurr, 2);
    testName = testPath(slashOccurr(numOccurr-2)+2 : slashOccurr(numOccurr)-1);
    message = sprintf('Test name: %s', testName);
    disp(' ');
    disp(' ');
    disp('******');
    disp('******');
    disp(message);

    % Load Images
    imageSet = LoadOutexImages(testCaseIndex, selectedTestCase);
    selectedTestCase{3} = size(imageSet, 1);

    % Execute tests
    if findBestValues == true
        % set parameters to default
        beta = lfpBeta;
        bell = lfpBellWidth;
        bins = lfpNumBins;
    end;
end;

```

```

for cycleNumber = 1 : finingCycles
    switch lfpSamplingFunction
        case 2 % Gaussian function - best bell width
            bestBellWidths = LFPBestBell(imageSet, testCaseIndex,
selectedTestCase, lfpWindowSize, lfpNumNeighbors, lfpRadius, bins, cycleNumber,
lfpSamplingFunction);
            bell = bestBellWidths(1);
        otherwise % best beta
            bestBetas = LFPBestBeta(imageSet, testCaseIndex,
selectedTestCase, lfpWindowSize, lfpNumNeighbors, lfpRadius, bins, cycleNumber,
lfpSamplingFunction);
            beta = bestBetas(1);
    end;

    bestBins = LFPBestBins(imageSet, testCaseIndex, selectedTestCase,
lfpWindowSize, lfpNumNeighbors, lfpRadius, beta, bell, cycleNumber,
lfpSamplingFunction);
    bins = bestBins;
end;
end;

% ROCCurve(imageSet, testCaseIndex, selectedTestCase, lbpNumSamples, lbpRadius,
lfpWindowSize, lfpNumNeighbors, lfpRadius, lfpBeta, lfpBellWidth, lfpNumBins,
lfpSamplingFunction);
end;

% Finish the test
disp('*****');
fprintf('\n');
disp('OutexTest finished.');
beep;

```

## APÊNDICE D – CÓDIGO FONTE EXCLUSIVO PARA TREINO COM O VISTEX

```
%%%%%%%%%%%%%%%
% LFPBestBeta - Determina o melhor beta para o LFP e determinado caso de
% teste.
%%%%%%%%%%%%%%%
function result = LFPBestBeta(imgSet, testInUse, janela, numNeighbors, radius,
numBins, fileVersion, lfpSamplingFunction, trainIDs, trainClassIDs)

testPath = testInUse{1};
numImages = testInUse{3};
classifier = testInUse{6};

fprintf('\n');
disp('=====');
message = sprintf('LFP Best Beta - %s\nParams:\nfunction: %d, winSize: %d,
numNeighbors: %d, radius: %d, numBins: %d, iteration: %d\n', testPath,
lfpSamplingFunction, janela, numNeighbors, radius, numBins, fileVersion);
disp(message);
disp('-----');

% Beta list
if lfpSamplingFunction == 3 % triangular function
    betaList = [1:1:256]';
else
    betaList = [0.100:0.050:2.500]';
end

threshold = 0;
lfp_hist = zeros(numImages, numBins);
sensitivityTable = zeros(length(betaList), 3);

for betaIndex = 1 : size(betaList);
    beta = betaList(betaIndex);

    % Generates LFP matrix
    lfp_matrix = BuildLFPMatrix(imgSet, janela, numNeighbors, radius, beta, 0,
lfpSamplingFunction);

    % Generates LFP histogram
    disp('Generating LFP histogram...');
    tic
    for imgIndex = 1 : numImages
        lfp_hist(imgIndex, :) = LFP_Hist(lfp_matrix{imgIndex}, numBins);
    end;
    toc

    % Classification process
    resultClassifier = ClassifyData(testInUse, lfp_hist, threshold, trainIDs,
trainClassIDs);
    sensitivityTable(betaIndex, 1) = beta;
    sensitivityTable(betaIndex, 2) = resultClassifier(:, 4);
end;

% Display the results
disp('-----');
disp('LFP Best Beta');
message = sprintf('Test case: %s', testPath);
disp(message);
message = sprintf('Function: %d', lfpSamplingFunction);
disp(message);
message = sprintf('Num bins: %d', numBins);
disp(message);
```

```

fprintf('\n');

[sensitivity row] = max(sensitivityTable(:,2));
bestBeta = sensitivityTable(row, 1);

classifierName = 'Chi-square';
if (classifier == 1)
    classifierName = 'Log-likelihood';
end;

message = sprintf('%s - Best beta: %f, sensitivity: %f', classifierName, bestBeta,
sensitivity);
disp(message);

fprintf('\n');
disp('Sensitivity table:');
fprintf('\tBeta\tSensitivity\n');
disp(sensitivityTable);
disp('-----');

% Results
slashOccurr = strfind(testPath, '\\');
numOccurr = size(slashOccurr, 2);
testName = testPath(slashOccurr(numOccurr-1)+2 : slashOccurr(numOccurr)-1);

% Plot and Saves Sensitivity Graph
h = figure;
plot(sensitivityTable(:,1), sensitivityTable(:,2));
title(classifierName);
ylabel('Sensitivity');
xlabel('Beta value');
grid;
fileName = sprintf('%s_Beta_classifier%d_func%d_ver%d', testName, classifier,
lfpSamplingFunction, fileVersion);
saveas(h, fileName, 'jpg');
close(h);

% Saves the results in an excel file
filename = sprintf('Outex_BestBeta_classifier%d_func%d_ver%d.xls', classifier,
lfpSamplingFunction, fileVersion);
mainSheet = 'Best_Beta';
rangeSpecifier = 'A%d:F%d';

% Header
data = [{'Test Name'}, {'Bins'}, {'Best Beta'}, {'Sensitivity'}];
range = sprintf(rangeSpecifier, 1, 1);
xlswrite(filename, data, mainSheet, range);

% Summary
data = [{testName}, {numBins}, {bestBeta}, {sensitivity}];
range = sprintf(rangeSpecifier, 2, 2);
xlswrite(filename, data, mainSheet, range);

% Detailed Data
message = sprintf('Saving results in %s', filename);
disp(message);

detailSheet = testName;
xlswrite(filename, sensitivityTable, detailSheet);
result = bestBeta;

end

%%%%%%%%%%%%%

```

```
% LFPBestBins - Determina o melhor numero de bins para o LFP e determinado
% caso de teste.
%%%%%%%%%%%%%%%
function result = LFPBestBins(imgSet, testInUse, janela, numNeighbors, radius,
beta, bellWidth, fileVersion, lfpSamplingFunction, trainIDs, trainClassIDs)

testPath = testInUse{1};
numImages = testInUse{3};
classifier = testInUse{6};

fprintf('\n');
disp('=====');
message = sprintf('LFP Best Bins - %s\nParams:\nfunction: %d, winSize: %d,
numNeighbors: %d, radius: %d, beta: %f, bell width: %f, iteration: %d', testPath,
lfpSamplingFunction, janela, numNeighbors, radius, beta, bellWidth, fileVersion);
disp(message);
disp('-----');

% number of bins
numBinsList = [256, 250:-10:110, 100:-5:10, 5]';

% Generates LFP matrix
lfp_matrix = BuildLFPMatrix(imgSet, janela, numNeighbors, radius, beta, bellWidth,
lfpSamplingFunction);

% calculates sensitivity
threshold = 0;
sensitivityTable = zeros(length(numBinsList), 2);

for binsIndex = 1 : length(numBinsList);
    numBins = numBinsList(binsIndex);
    lfp_hist = zeros(numImages, numBins);

    % Generates LFP histogram
    disp('Generating LFP histogram...');

    tic
    for imgIndex = 1 : numImages
        lfp_hist(imgIndex, :) = LFP_Hist(lfp_matrix{imgIndex}, numBins);
    end;
    toc

    % Classification process
    resultsClassifier = ClassifyData(testInUse, lfp_hist, threshold, trainIDs,
trainClassIDs);
    sensitivityTable(binsIndex, 1) = numBins;
    sensitivityTable(binsIndex, 2) = resultsClassifier(:, 4);
end;

% Display the results
disp('-----');
disp('LFP Best Bins');
message = sprintf('Test case: %s', testPath);
disp(message);
message = sprintf('Function: %d', lfpSamplingFunction);
disp(message);
message = sprintf('Beta: %f, Bell width: %f', beta, bellWidth);
disp(message);
fprintf('\n');

[sensitivity row] = max(sensitivityTable(:,2));
bestBins = sensitivityTable(row, 1);

classifierName = 'Chi-square';
if (classifier == 1)
    classifierName = 'Log-likelihood';
end;
```

```

message = sprintf('%s - Best bins: %d, sensitivity: %f', classifierName, bestBins,
sensitivity);
disp(message);

fprintf('\n');
disp('Sensitivity table:');
fprintf('\tBins\tSensitivity\n');
disp(sensitivityTable);
disp('-----');

% Results
slashOccurr = strfind(testPath, '\\');
numOccurr = size(slashOccurr, 2);
testName = testPath(slashOccurr(numOccurr-1)+2 : slashOccurr(numOccurr)-1);

% Plot and Saves Sensitivity Graph
h = figure;
plot(sensitivityTable(:,1), sensitivityTable(:,2));
title(classifierName);
ylabel('Sensitivity');
xlabel('Number of bins');
grid;
fileName = sprintf('%s_Bins_classifier%d_func%d_ver%d', testName, classifier,
lfpSamplingFunction, fileVersion);
saveas(h, fileName, 'jpg');
close(h);

% Saves the results in an excel file
filename = sprintf('Outex_BestBins_classifier%d_func%d_ver%d.xls', classifier,
lfpSamplingFunction, fileVersion);
mainSheet = 'Best_Bins';
rangeSpecifier = 'A%d:E%d';

% Header
data = [{'Test Name'}, {'Beta'}, {'BellWidth'}, {'Num Bins'}, {'Sensitivity'}];
range = sprintf(rangeSpecifier, 1, 1);
xlswrite(filename, data, mainSheet, range);

% Summary
data = [{testName}, {beta}, {bellWidth}, {bestBins}, {sensitivity}];
range = sprintf(rangeSpecifier, 2, 2);
xlswrite(filename, data, mainSheet, range);

% Detailed Data
message = sprintf('Saving results in %s', filename);
disp(message);

detailSheet = testName;
xlswrite(filename, sensitivityTable, detailSheet);

result = bestBins;

end

%%%%%%%%%%%%%
% LFPBestBell - Determina o melhor valor de sigma para a gaussiana empregada
% pelo LFP em determinado caso de teste.
%%%%%%%%%%%%%
function result = LFPBestBell(imgSet, testInUse, janela, numNeighbors, radius,
numBins, fileVersion, lfpSamplingFunction)

testPath = testInUse{1};

```

```

numImages = testInUse{3};
classifier = testInUse{6};

fprintf('\n');
disp('=====');
message = sprintf('LFP Best Bell Width - %s, winSize: %d, numBins: %d, iteration: %d\n',
    testPath, janela, numBins, fileVersion);
disp(message);

% Beta list
bellWidthList = [0.5:0.1:1, 2:0.5:15]';
threshold = 0;

lfp_hist = zeros(numImages, numBins);
sensitivityTable = zeros(length(bellWidthList), 3);

for bellWidthIndex = 1 : size(bellWidthList);
    bellWidth = bellWidthList(bellWidthIndex);

    % Generates LFP matrix
    lfp_matrix = BuildLFPMatrix(imgSet, janela, numNeighbors, radius, 0, bellWidth,
        lfpSamplingFunction);

    % Generates LFP histogram
    disp('Generating LFP histogram...');

    tic
    for imgIndex = 1 : numImages
        lfp_hist(imgIndex, :) = LFP_Hist(lfp_matrix{imgIndex}, numBins);
    end;
    toc

    % Classification process
    resultClassifier = ClassifyData(testInUse, lfp_hist, threshold);
    sensitivityTable(bellWidthIndex, 1) = bellWidth;
    sensitivityTable(bellWidthIndex, 2) = resultClassifier(:, 4);
end;

% Display the results
disp('-----');
disp('LFP Best Bell Width');
message = sprintf('Test case: %s', testPath);
disp(message);
message = sprintf('Num bins: %d', numBins);
disp(message);
fprintf('\n');

[sensitivity row] = max(sensitivityTable(:,2));
bestBell = sensitivityTable(row, 1);

classifierName = 'Chi-square';
if (classifier == 1)
    classifierName = 'Log-likelihood';
end;

message = sprintf('%s - Best bell width: %f, sensitivity: %f', classifierName,
    bestBell, sensitivity);
disp(message);

fprintf('\n');
disp('Sensitivity table:');
fprintf('\tBell width\tSensitivity\n');
disp(sensitivityTable);
disp('-----');

% Results

```

```

slashOccurr = strfind(testPath, '\\');
numOccurr = size(slashOccurr, 2);
testName = testPath(slashOccurr(numOccurr-1)+2 : slashOccurr(numOccurr)-1);

% Plot and Saves Sensitivity Graph
h = figure;
plot(sensitivityTable(:,1), sensitivityTable(:,2));
title(classifierName);
ylabel('Sensitivity');
xlabel('Bell width');
grid;
fileName = sprintf('%s_Bell_classifier%d_func%d_ver%d', testName, classifier,
lfpSamplingFunction, fileVersion);
saveas(h, fileName, 'jpg');
close(h);

% Saves the results in an excel file
filename = sprintf('Outex_BestBell_classifier%d_func%d_ver%d.xls', classifier,
lfpSamplingFunction, fileVersion);
mainSheet = 'Best_Bell';
rangeSpecifier = 'A%d:D%d';

% Header
data = {[{'Test Name'}, {'Bins'}, {'Best Bell'}, {'Sensitivity'}]};
range = sprintf(rangeSpecifier, 1, 1);
xlswrite(filename, data, mainSheet, range);

% Summary
data = {[{testName}, {numBins}, {bestBell}, {sensitivity}];
range = sprintf(rangeSpecifier, testCaseIndex+1, testCaseIndex+1);
xlswrite(filename, data, mainSheet, range);

% Detailed Data
message = sprintf('Saving results in %s', filename);
disp(message);

detailSheet = testName;
xlswrite(filename, sensitivityTable, detailSheet);

result = bestBell;

end

%%%%%%%%%%%%%%%
% ClassifierChiSquare - Faz a classificacao de um conjunto de teste
% utilizando a metrica chi-quadrado.
%%%%%%%%%%%%%%%
function result = ClassifierChiSquare(testInUse, trains, tests, trainClassIDs,
testClassIDs, thresholds)

summarizeHist = testInUse{5};

trainNum = size(trains, 1);
testNum = size(tests, 1);
DM = zeros(testNum, trainNum);

for i = 1 : testNum;
    test = tests(i,:);
    DM(i,:) = distMATCHisquare(trains, test)';
end;

% Remove as comparacoes diretas entre a mesma amostra
if summarizeHist == false

```

```

for i = 1 : size(DM, 1)
    for j = 1 : size(DM, 1)
        if i == j
            DM(i, j) = 999;
        end
    end;
end

if (isempty(thresholds))
    thresholds = 0;
end;

rocData = zeros(length(thresholds), 3);

for i = 1 : length(thresholds)
    classificationResult = ClassifyOnNN(DM, trainClassIDs, testClassIDs,
thresholds(i));

    rocData(i, 1) = thresholds(i);
    rocData(i, 2) = classificationResult(1); % min distance
    rocData(i, 3) = classificationResult(2); % max distance
    rocData(i, 4) = classificationResult(3); % sensitivity
    rocData(i, 5) = classificationResult(4); % specificity
end;

result = rocData;

end

%%%%%%%%%%%%%%%
% ClassifierLogLikelihood - Faz a classificacao de um conjunto de teste
% utilizando a metrica estatistica G.
%%%%%%%%%%%%%%%
function result = ClassifierLogLikelihood(testInUse, trains, tests, trainClassIDs,
testClassIDs, thresholds)

summarizeHist = testInUse{5};

trainNum = size(trains, 1);
testNum = size(tests, 1);
DM = zeros(testNum, trainNum);

for i = 1 : testNum;
    test = tests(i,:);
    DM(i,:) = LogLikelihood(trains, test)';
end;

% Remove as comparacoes diretas entre a mesma amostra
if summarizeHist == false
    for i = 1 : size(DM, 1)
        for j = 1 : size(DM, 1)
            if i == j
                DM(i, j) = 999;
            end
        end;
    end
end

if (isempty(thresholds))
    thresholds = 0;
end;

```

```

rocData = zeros(length(thresholds), 3);

for i = 1 : length(thresholds)
    classificationResult = ClassifyOnNN(DM, trainClassIDs, testClassIDs,
thresholds(i));

    rocData(i, 1) = thresholds(i);
    rocData(i, 2) = classificationResult(1); % min distance
    rocData(i, 3) = classificationResult(2); % max distance
    rocData(i, 4) = classificationResult(3); % sensitivity
    rocData(i, 5) = classificationResult(4); % specificity
end;

result = rocData;

end

%%%%%%%%%%%%%%%
% ClassifyData - Executa o processo de classificacao de um conjunto de
% teste.
% classifier:
% 0 - ChiSquare, 1 - Log likelihood
%%%%%%%%%%%%%%%
function resultMatrix = ClassifyData(testInUse, histogram, thresholdList, trainIDs,
trainClassIDs)

testPath = testInUse{1};
testList = testInUse{2}{1};
summarizeHist = testInUse{5};
classifier = testInUse{6};

% Classification process
minDist = intmax;
maxDist = 0;

classifierName = 'Chi-square';
if (classifier == 1)
    classifierName = 'Log likelihood';
end

message = sprintf('Classifying data - Classifier: %s', classifierName);
disp(message);
tic
resultMatrix = zeros(length(thresholdList), 5);
disp(' ');

numTest = testList(1);

testIDs = trainIDs;
testClassIDs = trainClassIDs;

% classification
trains = histogram(:, :);
tests = histogram(:, :, :);

if (classifier == 0)
    classificationResult = ClassifierChiSquare(testInUse, trains, tests,
trainClassIDs, testClassIDs, thresholdList);
else
    classificationResult = ClassifierLogLikelihood(testInUse, trains, tests,
trainClassIDs, testClassIDs, thresholdList);
end;

```

```

tempMinDist = classificationResult(:, 2);
tempMaxDist = classificationResult(:, 3);

sensitivity = classificationResult(:, 4);
specificity = classificationResult(:, 5);

message = sprintf('Test: %d', numTest);
disp(message);
message = sprintf('sensitivity: %f', sensitivity);
disp(message);

minDist = min(minDist, tempMinDist);
maxDist = max(maxDist, tempMaxDist);

resultMatrix = resultMatrix + classificationResult;
endTime = toc;
resultMatrix = resultMatrix;

resultMatrix(:, 2) = minDist;
resultMatrix(:, 3) = maxDist;

end

%%%%%%%%%%%%%%%
% FindBestParams - Encontra o melhor beta e numero de bins para o LFP.
%%%%%%%%%%%%%%%
function [beta, bins] = FindBestParams(imageSet, selectedTestCase, lfpWindowSize,
lfpNumNeighbors, lfpRadius, lfpSamplingFunction, lfpBeta, lfpBellWidth, lfpNumBins,
trainIDs, trainClassIDs)

finingCycles = 1;

% set parameters to default
beta = lfpBeta;
bell = lfpBellWidth;
bins = lfpNumBins;

for cycleNumber = 1 : finingCycles
switch lfpSamplingFunction
    case 2 % Gaussian function - best bell width
        bestBellWidths = LFPBestBell(imageSet, selectedTestCase, lfpWindowSize,
lfpNumNeighbors, lfpRadius, bins, cycleNumber, lfpSamplingFunction);
        bell = bestBellWidths(1);
    otherwise % best beta
        bestBetas = LFPBestBeta(imageSet, selectedTestCase, lfpWindowSize,
lfpNumNeighbors, lfpRadius, bins, cycleNumber, lfpSamplingFunction, trainIDs,
trainClassIDs);
        beta = bestBetas(1);
    end;

    bestBins = LFPBestBins(imageSet, selectedTestCase, lfpWindowSize,
lfpNumNeighbors, lfpRadius, beta, bell, cycleNumber, lfpSamplingFunction, trainIDs,
trainClassIDs);
    bins = bestBins;
end;

end

%%%%%%%%%%%%%%%
% FindVistexParams - Faz o treinamento do beta e numero de bins para o LFP,
% variando o numero de amostras que compoem o conjunto de treino.
%%%%%%%%%%%%%%%

```

```

function FindVistexParams(selectedTestCase, maxNumSamples, reverseOrder,
lfpWindowSize, lfpNumNeighbors, lfpRadius, lfpSamplingFunction, lfpBeta,
lfpBellWidth, lfpNumBins)

testPath = selectedTestCase{1};
slashOccurr = strfind(testPath, '\\');
numOccurr = size(slashOccurr, 2);
testName = testPath(slashOccurr(numOccurr-2) : slashOccurr(numOccurr)-1);
message = sprintf('Test name: %s', testName);
disp(' ');
disp(' ');
disp('******');
disp('******');
disp(message);

% Execute test using original samples order (trainning -> test samples)
lineIndex = 0;
data = [];

numSamplesList = [1:3:31, 32, 33, 34:3:64]

for i = 1 : size(numSamplesList, 2)
    numSamples = numSamplesList(i);

    [trainIDs, trainClassIDs] = GetTrainningTestSets(reverseOrder, testPath,
numSamples);

    imageSet = LoadOutexImages(selectedTestCase, trainIDs);
    selectedTestCase{3} = size(imageSet, 1);

    [beta, bins] = FindBestParams(imageSet, selectedTestCase, lfpWindowSize,
lfpNumNeighbors, lfpRadius, lfpSamplingFunction, lfpBeta, lfpBellWidth, lfpNumBins,
trainIDs, trainClassIDs);
    clearvars imageSet;

    strLog = sprintf('numSamples: %d, reverseOrder: %d', numSamples, reverseOrder);
    disp(strLog);

    % numSamples, beta, bins
    lineIndex = lineIndex + 1;
    data(lineIndex, 1) = numSamples;
    data(lineIndex, 2) = beta;
    data(lineIndex, 3) = bins;
end

% XSl data
xlsFileName = 'VistexParams.xls';

if reverseOrder == false
    sheet = sprintf('%dx%d_func%d_original', lfpNumNeighbors, lfpRadius,
lfpSamplingFunction);
else
    sheet = sprintf('%dx%d_func%d_reverse', lfpNumNeighbors, lfpRadius,
lfpSamplingFunction);
end

range = sprintf('A1:C%d', size(data, 1));
xlswrite(xlsFileName, data, sheet, range);

end

%%%%%%%%%%%%%
% LoadOutexImages - carrega os arquivos de imagens.

```

```
%%%%%%%%
function imageSet = LoadOutexImages(testData, trainIDs)

testPath = testData{1};
numClasses = size(trainIDs, 3);
numImgByClass = size(trainIDs, 2);
imageSet = cell(numClasses * numImgByClass, 1);

disp('Loading images...');
imgIndex = 0;

for classIndex = 1 : numClasses
    for imgClassIndex = 1 : numImgByClass
        filename = sprintf('%s\images\%06d.ras', testPath, trainIDs(1,
imgClassIndex, classIndex));

        img = imread(filename);
        img = im2double(img);
        img = (img-mean(img(:)))/std(img(:))*20+128; % image normalization, to
remove global intensity

        imgIndex = imgIndex + 1;
        imageSet{imgIndex} = img;
    end
end;
end

%%%%%%%
% Script principal para execucao dos testes.
%%%%%%%
clc; close all; fclose all;
disp('Running VistexTest...');

% Outex data
databasePath = 'C:\Users\Carlos\Documents\Mestrado\Banco de imagens\';

% selectedTestCase:
% test path, number of tests, number of pictures in the database, operator,
% summarize histogram
selectedTestCase = {strcat(databasePath, 'Vistex\\'), {1:8}, 17280, 2, false, 0};

% Test data
testCaseList = [1];
maxNumSamples = 64;

% LFP
lfpWindowSize = 0; % for use with the LFP sigmoidal (classic form)
lfpNumNeighbors = 8;
lfpRadius = 1;
lfpNumBins = 256;
lfpBeta = 1.005;
lfpBellWidth = 1.005;

% Sampling function:
% 1 - Sigmoidal
% 2 - Gaussian
% 3 - Triangular
% 4 - LFP Sampling - Sigmoidal
% 5 - LFP_Q - Sigmoidal, com vizinhanca quadrangular a partir de um raio
lfpSamplingFunction = 4;

% Test algorithm
```

```
if lfpSamplingFunction == 2 % Gaussian
    lfpBeta = 0;
else
    lfpBellWidth = 0;
end;

FindVistexParams(selectedTestCase, maxNumSamples, false, lfpWindowSize,
lfpNumNeighbors, lfpRadius, lfpSamplingFunction, lfpBeta, lfpBellWidth,
lfpNumBins);
FindVistexParams(selectedTestCase, maxNumSamples, true, lfpWindowSize,
lfpNumNeighbors, lfpRadius, lfpSamplingFunction, lfpBeta, lfpBellWidth,
lfpNumBins);

%{
lfpWindowSize = 16;
lfpRadius = 2;
FindVistexParams(selectedTestCase, maxNumSamples, false, lfpWindowSize,
lfpNumNeighbors, lfpRadius, lfpSamplingFunction, lfpBeta, lfpBellWidth,
lfpNumBins);
FindVistexParams(selectedTestCase, maxNumSamples, true, lfpWindowSize,
lfpNumNeighbors, lfpRadius, lfpSamplingFunction, lfpBeta, lfpBellWidth,
lfpNumBins);

lfpWindowSize = 24;
lfpRadius = 3;
FindVistexParams(selectedTestCase, maxNumSamples, false, lfpWindowSize,
lfpNumNeighbors, lfpRadius, lfpSamplingFunction, lfpBeta, lfpBellWidth,
lfpNumBins);
FindVistexParams(selectedTestCase, maxNumSamples, true, lfpWindowSize,
lfpNumNeighbors, lfpRadius, lfpSamplingFunction, lfpBeta, lfpBellWidth,
lfpNumBins);
%}

% Finish the test
disp('*****');
fprintf('\n');
disp('VistexTest finished.');
beep;
```

## APÊNDICE E – CÓDIGO FONTE EXCLUSIVO PARA TESTE COM O VISTEX

```
%%%%%%%%%%%%%%%
% LFPReference - Executa um teste de referencia do LFP em um caso de teste.
%%%%%%%%%%%%%%%
function testResult = LFPReference(imageSet, testCaseIndex, testInUse, windowSize,
numNeighbors, radius, beta, bellWidth, numBins, lfpSamplingFunction, trainIDs,
trainClassIDs)

testPath = testInUse{1};
numImages = testInUse{3};
classifier = testInUse{6};

% Generates LFP code matrix
disp('Generating LFP matrix...');

tic;
lfp_matrix = BuildLFPMatrix(imageSet, windowSize, numNeighbors, radius, beta,
bellWidth, lfpSamplingFunction);
lfpCodeMatrixTime = toc;

message = sprintf('LFP code matrix built in: %f seconds.', lfpCodeMatrixTime);
disp(message);

% calculates sensitivity
lfp_hist = zeros(numImages, numBins);

disp('Generating LFP histogram...');

tic
for imgIndex = 1 : numImages
    lfp_hist(imgIndex, :) = LFP_Hist(lfp_matrix{imgIndex}, numBins);
end;
lfpHistTime = toc;

message = sprintf('LFP histograms built in: %f seconds.', lfpHistTime);
disp(message);

% Classification process
threshold = 0;
[resultsClassifier, classificationResults] = ClassifyData(testCaseIndex, testInUse,
lfp_hist, threshold, trainIDs, trainClassIDs);

% Display the results
disp('-----');
slashOccurr = strfind(testPath, '\\');
numOccurr = size(slashOccurr, 2);
testName = testPath(slashOccurr(numOccurr-2) + 2 : slashOccurr(numOccurr)-1);

message = sprintf('LFP Reference Results - %s\n', testName);
disp(message);
message = sprintf('Neighbor size: %d', windowSize);
disp(message);
message = sprintf('Num Neighbors: %d', numNeighbors);
disp(message);
message = sprintf('Radius: %d', radius);
disp(message);
message = sprintf('Sampling function: %d', lfpSamplingFunction);
disp(message);
message = sprintf('Bell width: %f', bellWidth);
disp(message);
message = sprintf('Beta: %f', beta);
disp(message);
message = sprintf('Number of bins: %d', numBins);
disp(message);
```

```

message = sprintf('LFP code matrix built in: %f seconds.', lfpCodeMatrixTime);
disp(message);
message = sprintf('LFP histograms built in: %f seconds.', lfpHistTime);
disp(message);

classifierName = 'Chi-Square';
if (classifier == 1)
    classifierName = 'Log likelihood';
end

fprintf('\n');
disp(classifierName);
message = sprintf('Min distance: %f', resultsClassifier(1, 2));
disp(message);
message = sprintf('Max distance: %f', resultsClassifier(1, 3));
disp(message);
message = sprintf('Sensitivity: %f', resultsClassifier(1, 4));
disp(message);
message = sprintf('Specificity: %f', resultsClassifier(1, 5));
disp(message);
disp('-----');

% Save the results
filename = 'OutexReference.xls';
sheet = sprintf('LFP_Reference_f%d', lfpSamplingFunction);
rangeSpecifier = 'A%d:N%d';

% Header
data = {[{'Test Name'}, {'Neighbor size'}, {'Beta'}, {'Number of bins'}, {'LFP code build time'}, {'Histogram build time'}, {'Min Dist (Chi-Sqr)'}, {'Max Dist (Chi-Sqr)'}, {'Sensitivity (Chi-Sqr)'}, {'Specificity (Chi-Sqr)'}, {'Min Dist (LogLik)'}, {'Max Dist (LogLik)'}, {'Sensitivity (LogLik)'}, {'Specificity (LogLik)'}];
range = sprintf(rangeSpecifier, 1, 1);
xlswrite(filename, data, sheet, range);

% Data
[rows, cols] = size(resultsClassifier);
zeroResults = zeros(rows, cols);

if (classifier == 0)
    resultsChiSquare = resultsClassifier;
    resultsLogLikelihood = zeroResults;
else
    resultsChiSquare = zeroResults;
    resultsLogLikelihood = resultsClassifier;
end

message = sprintf('Saving results in %s', filename);
disp(message);

data = [{testPath}, {windowSize}, {beta}, {numBins}, {lfpCodeMatrixTime},
{lfpHistTime}, {resultsChiSquare(1, 2)}, {resultsChiSquare(1, 3)},
{resultsChiSquare(1, 4)}, {resultsChiSquare(1, 5)}, {resultsLogLikelihood(1, 2)},
{resultsLogLikelihood(1, 3)}, {resultsLogLikelihood(1, 4)},
{resultsLogLikelihood(1, 5)}];
range = sprintf(rangeSpecifier, testCaseIndex+1, testCaseIndex+1);
xlswrite(filename, data, sheet, range);

testResult = classificationResults(:,1);

disp('Test Results:');
disp(testResult);

end

```

```

%%%%%%%%%%%%%%%
% LBPReference - Executa um teste de referencia do LBP em um caso de teste.
%%%%%%%%%%%%%%%
function testResult = LBPReference(imageSet, testCaseIndex, testInUse, numSamples,
radius, trainIDs, trainClassIDs)

testPath = testInUse{1};
numImages = testInUse{3};
classifier = testInUse{6};

numBins = 256;
lbpMode = testInUse{4};

switch numSamples
    case 8
        if (lbpMode == 1) % ri
            numBins = 36;
        else if (lbpMode == 2) % riu2
            numBins = 10;
        else
            error('Invalid LBP mode (selected 8 samples)');
        end
    end

    case 16
        if (lbpMode == 1) % ri
            numBins = 4116;
        else if (lbpMode == 2) % riu2
            numBins = 18;
        else
            error('Invalid LBP mode (selected 8 samples)');
        end
    end

    case 24
        if (lbpMode == 1) % ri
            numBins = 699252;
        else if (lbpMode == 2) % riu2
            numBins = 26;
        else
            error('Invalid LBP mode (selected 16 samples)');
        end
    end
end

% Generates LBP code matrix
disp('Generating LBP matrix...');

tic;
lbp_matrix = BuildLBPMatrix(imageSet, numSamples, radius, lbpMode);
lbpCodeMatrixTime = toc;

message = sprintf('LBP code matrix built in: %f seconds.', lbpCodeMatrixTime);
disp(message);

% calculates sensitivity
lbp_hist = zeros(numImages, numBins);

disp('Generating LBP histogram...');

tic;
for imgIndex = 1 : numImages
    lbp_hist(imgIndex, :) = LBP_Hist(lbp_matrix{imgIndex}, numBins);
end;
lbpHistTime = toc;

```

```

message = sprintf('LBP histograms built in: %f seconds.', lbpHistTime);
disp(message);

% Classification process
threshold = 0;
[resultsClassifier, classificationResults] = ClassifyData(testCaseIndex, testInUse,
lbp_hist, threshold, trainIDs, trainClassIDs);

% Display the results
disp('-----');
slashOccurr = strfind(testPath, '\\');
numOccurr = size(slashOccurr, 2);
testName = testPath(slashOccurr(numOccurr-2)+2 : slashOccurr(numOccurr)-1);

message = sprintf('LBP Reference Results - %s\n', testName);
disp(message);
message = sprintf('Neighbor size: %d', numSamples);
disp(message);
message = sprintf('Radius: %f', radius);
disp(message);

switch lbpMode
    case 0
        disp('LBP mode: classic');
    case 1
        disp('LBP mode: ri');
    case 2
        disp('LBP mode: riu2');
end;

message = sprintf('LBP code matrix built in: %f seconds.', lbpCodeMatrixTime);
disp(message);
message = sprintf('LBP histograms built in: %f seconds.', lbpHistTime);
disp(message);

classifierName = 'Chi-Square';
if (classifier == 1)
    classifierName = 'Log likelihood';
end

fprintf('\n');
disp(classifierName);
%message = sprintf('Min distance: %f', resultsClassifier(1, 2));
%disp(message);
%message = sprintf('Max distance: %f', resultsClassifier(1, 3));
%disp(message);
message = sprintf('Sensitivity: %f', resultsClassifier(1, 4));
disp(message);
%message = sprintf('Specificity: %f', resultsClassifier(1, 5));
%disp(message);
disp('-----');

% Save the results
filename = 'OutexReference.xls';
sheet = sprintf('LBP_Reference_f%d', lbpMode);
rangeSpecifier = 'A%d:N%d';

% Header
data = {[{'Test Name'}, {'Num Samples'}, {'Radius'}, {'LBP Mode'}, {'LBP Code build
time'}, {'Histogram build time'}, {'Min Dist (Chi-Sqr)'}, {'Max Dist (Chi-Sqr)'},
{'Sensitivity (Chi-Sqr)'}, {'Specificity (Chi-Sqr)'}, {'Min Dist (LogLik)'}, {'Max
Dist (LogLik)'}, {'Sensitivity (LogLik)'}, {'Specificity (LogLik)'}];
range = sprintf(rangeSpecifier, 1, 1);
xlswrite(filename, data, sheet, range);

% Data

```

```

[rows, cols] = size(resultsClassifier);
zeroResults = zeros(rows, cols);

if (classifier == 0)
    resultsChiSquare = resultsClassifier;
    resultsLogLikelihood = zeroResults;
else
    resultsChiSquare = zeroResults;
    resultsLogLikelihood = resultsClassifier;
end

message = sprintf('Saving results in %s', filename);
disp(message);

data = [{testPath}, {numSamples}, {radius}, {lbpMode}, {lbpCodeMatrixTime},
{lbpHistTime}, {resultsChiSquare(1, 2)}, {resultsChiSquare(1, 3)},
{resultsChiSquare(1, 4)}, {resultsChiSquare(1, 5)}, {resultsLogLikelihood(1, 2)},
{resultsLogLikelihood(1, 3)}, {resultsLogLikelihood(1, 4)},
{resultsLogLikelihood(1, 5)}];
range = sprintf(rangeSpecifier, testCaseIndex+1, testCaseIndex+1);
xlswrite(filename, data, sheet, range);

testResult = classificationResults(:,1);

end

%%%%%%%%%%%%%
% ClassifierChiSquare - Faz a classificacao de um conjunto de teste
% utilizando a metrica chi-quadrado.
%%%%%%%%%%%%%
function result = ClassifierChiSquare(trains, tests, trainClassIDs, testClassIDs,
thresholds)

trainNum = size(trains, 1);
testNum = size(tests, 1);
DM = zeros(testNum, trainNum);

for i = 1 : testNum;
    test = tests(i,:);
    DM(i,:) = distMATChiSquare(trains, test)';
end;

if (isempty(thresholds))
    thresholds = 0;
end;

rocData = zeros(length(thresholds), 3);

for i = 1 : length(thresholds)
    classificationResult = ClassifyOnNN(DM, trainClassIDs, testClassIDs,
thresholds(i));

    rocData(i, 1) = thresholds(i);
    rocData(i, 2) = classificationResult(1); % min distance
    rocData(i, 3) = classificationResult(2); % max distance
    rocData(i, 4) = classificationResult(3); % sensitivity
    rocData(i, 5) = classificationResult(4); % specificity
end;

result = rocData;

end

```

```

%%%%%%%%%%%%%%%
% ClassifierChiSquare - Faz a classificacao de um conjunto de teste
% utilizando a metrica chi-quadrado.
%%%%%%%%%%%%%%%
function result = ClassifierLogLikelihood(trains, tests, trainClassIDs,
testClassIDs, thresholds)

trainNum = size(trains, 1);
testNum = size(tests, 1);
DM = zeros(testNum, trainNum);

for i = 1 : testNum;
    test = tests(i,:);
    DM(i,:) = LogLikelihood(trains, test)';
end;

if (isempty(thresholds))
    thresholds = 0;
end;

rocData = zeros(length(thresholds), 3);

for i = 1 : length(thresholds)
    classificationResult = ClassifyOnNN(DM, trainClassIDs, testClassIDs,
thresholds(i));

    rocData(i, 1) = thresholds(i);
    rocData(i, 2) = classificationResult(1); % min distance
    rocData(i, 3) = classificationResult(2); % max distance
    rocData(i, 4) = classificationResult(3); % sensitivity
    rocData(i, 5) = classificationResult(4); % specificity
end;

result = rocData;

end

%%%%%%%%%%%%%%%
% ClassifyData - Executa o processo de classificacao de um conjunto de
% teste.
% classifier:
% 0 - ChiSquare, 1 - Log likelihood
%%%%%%%%%%%%%%%
function [resultMatrix, testResult] = ClassifyData(testCaseIndex, testInUse,
histogram, thresholdList, trainIDs, trainClassIDs)

testPath = testInUse{1};
testList = testInUse{2}{1};
summarizeHist = testInUse{5};
classifier = testInUse{6};
numTests = size(testList, 2);

% Classification process
minDist = intmax;
maxDist = 0;

classifierName = 'Chi-square';
if (classifier == 1)
    classifierName = 'Log likelihood';

```

```

end

message = sprintf('Classifying data - Classifier: %s', classifierName);
disp(message);
tic
resultMatrix = zeros(length(thresholdList), 5);
disp(' ');

testResult = zeros(numTests);

for testIndex = 1 : numTests
    numTest = testList(testIndex);

    % read picture ID of training and test samples, and read class ID
    % of training and test samples
    testData = sprintf('%s0%02d\test.txt', testPath, numTest-1);

    testPath = testInUse{1};
    [testIDs, testClassIDs] = ReadOutexTxt(testPath, testData);

    % classification
    trains = histogram(trainIDs,:);
    tests = histogram(testIDs,:);

    % If sumerizeHist == true, all histograms belonging to a class will
    % summarized into one same histogram.
    if summarizeHist == true
        newTrainClassIDs = unique(trainClassIDs);
        newTrains = zeros(size(newTrainClassIDs, 2), size(trains, 2)); %
numClasses, numBins

        for i = 1 : length(trainClassIDs)
            trainClassId = trainClassIDs(i);

            newTrainClassIDsIndex = find(newTrainClassIDs == trainClassId);
            newTrains(newTrainClassIDsIndex, :) = newTrains(newTrainClassIDsIndex,
:) + trains(i, :);
        end;

        for i = 1 : size(newTrains, 1)
            newTrains(i, :) = newTrains(i, :) / sum(newTrains(i, :));
        end;
    else
        newTrainClassIDs = trainClassIDs;
        newTrains = trains;
    end;

    if (classifier == 0)
        classificationResult = ClassifierChiSquare(newTrains, tests,
newTrainClassIDs, testClassIDs, thresholdList);
    else
        classificationResult = ClassifierLogLikelihood(newTrains, tests,
newTrainClassIDs, testClassIDs, thresholdList);
    end;

    tempMinDist = classificationResult(:, 2);
    tempMaxDist = classificationResult(:, 3);
    sensitivity = classificationResult(:, 4);
    specificity = classificationResult(:, 5);

    testResult(testIndex) = sensitivity;

    message = sprintf('Test: %d', numTest);
    disp(message);
    message = sprintf('sensitivity: %f', sensitivity);

```

```

    disp(message);

    minDist = min(minDist, tempMinDist);
    maxDist = max(maxDist, tempMaxDist);

    resultMatrix = resultMatrix + classificationResult;
end;
endTime = toc;

resultMatrix = resultMatrix / numTests;

resultMatrix(:, 2) = minDist;
resultMatrix(:, 3) = maxDist;

end

%%%%%%%%%%%%%%%
% LoadOutexImages - carrega os arquivos de imagens.
%%%%%%%%%%%%%%%
function imageSet = LoadOutexImages(testData)

testPath = testData{1};
numImages = testData{3};
imageSet = cell(numImages, 1);
imagePath = '%s\images\%06d.ras';
convertToGray = false;

bmpTest(1) = {'Contrib_TC_00006'};
bmpTest(2) = {'Contrib_TC_00013'};
bmpTest(3) = {'Contrib_TC_00014'};

for i = 1 : length(bmpTest)
    bmpTestPath = bmpTest{i};

    idx = strfind(testPath, bmpTestPath);

    if length(idx) > 0
        imagePath = '%s\images\%06d.bmp';
        convertToGray = true;
    end
end

disp('Loading images...');

normalize = (strfind(testPath, 'Outex_TC_00012') == 0);

for imgIndex = 1 : numImages
    filename = sprintf(imagePath, testPath, imgIndex-1);

    img = imread(filename);

    if (convertToGray == true)
        img = rgb2gray(img);
    end;

    img = im2double(img);

    if normalize
        img = (img-mean(img(:)))/std(img(:))*20+128; % image normalization, to
remove global intensity
    end;
end;

```

```

        imageSet{imgIndex} = img;
    end

end

%%%%%%%%%%%%%
% TestVistexParams - Executa os testes variando o numero de amostras do
% conjunto de treinamento.
%%%%%%%%%%%%%
function TestVistexParams(paramsTable, reverseOrder, sheet, testPath, imageSet,
testCaseIndex, selectedTestCase, lfpWindowSize, lfpNumNeighbors, lfpRadius,
lfpSamplingFunction)

numTests = size(paramsTable, 1);
lfpTestResult = zeros(numTests, 11);
lbpTestResult = zeros(numTests, 11);

for i = 1 : numTests
    numSamples = paramsTable(i, 1);
    beta = paramsTable(i, 2);
    lfpBellWidth = beta;
    bins = paramsTable(i, 3);

    [trainIDs, trainClassIDs] = GetTrainningTestSets(reverseOrder, testPath,
numSamples);

    lfpClassificationResults = LFPReference(imageSet, testCaseIndex,
selectedTestCase, lfpWindowSize, lfpNumNeighbors, lfpRadius, beta, lfpBellWidth,
bins, lfpSamplingFunction, trainIDs, trainClassIDs);
    lfpTestResult(i, 1) = numSamples;
    lfpTestResult(i, 2) = beta;
    lfpTestResult(i, 3) = bins;
    lfpTestResult(i, 4:11) = lfpClassificationResults;

    lbpClassificationResults = LBPRference(imageSet, testCaseIndex,
selectedTestCase, lfpNumNeighbors, lfpRadius, trainIDs, trainClassIDs);
    lbpTestResult(i, 1) = numSamples;
    lbpTestResult(i, 2:9) = lbpClassificationResults;
end

% XSls data
xlsFileName = 'VistexResults.xls';
lfpRange = sprintf('A1:K%d', numTests);
lfpSheet = sprintf('%s-lfp', sheet);
xlswrite(xlsFileName, lfpTestResult, lfpSheet, lfpRange);

lbpRange = sprintf('A1:I%d', numTests);
lbpSheet = sprintf('%s-lbp', sheet);
xlswrite(xlsFileName, lbpTestResult, lbpSheet, lbpRange);

end

%%%%%%%%%%%%%
% Script principal para execucao dos testes.
%%%%%%%%%%%%%
clc; close all; fclose all;
disp('Running VistexTest...');

% Outex data
databasePath = 'C:\\\\Users\\\\Carlos\\\\Documents\\\\Mestrado\\\\Banco de imagens\\\\';

```

```
% testCase(:, :) = {
% test path: string
% {suit test list}: an array with the list of test cases
% number of images: integer
% LBP mode: 0 - classic, 1 - ri, 2 - riu2
% sum histograms: boolean indicate to sum all histograms into a big one
% classifier: 0 - chi-square, 1 - log likelihood
testCase(1, :) = {strcat(databasePath, 'Vistex\\'), {1:8}, 17280, 2, false, 0};

% Test data
testCaseList = [1];

% LFP
lfpWindowSize = 0;

% Sampling function:
% 1 - Sigmoidal
% 2 - Gaussian
% 3 - Triangular
% 4 - LFP Sampling - Sigmoidal
% 5 - LFP_Q - Sigmoidal, com vizinhanca quadrangular a partir de um raio
lfpSamplingFunction = 4;

% Load Vistex Params
maxNummSamples = 64;
xlsParamsFileName = 'C:\Users\Carlos\Projetos\Experimentos\Vistex - Only Trainning
- v1\VistexParams.xls';
dataRange = sprintf('A1:C%d', maxNummSamples);

sheet8x1Orig = '8x1_func4_original';
sheet8x1Rev = '8x1_func4_reverse';

sheet16x2Orig = '16x2_func4_original';
sheet16x2Rev = '16x2_func4_reverse';

sheet24x3Orig = '246x3_func4_original';
sheet24x3Rev = '246x3_func4_reverse';

if ~exist(xlsParamsFileName, 'file')
    error(strcat('File not found: ', xlsParamsFileName));
end;

params8x1Orig = xlsread(xlsParamsFileName, sheet8x1Orig, dataRange);
params8x1Rev = xlsread(xlsParamsFileName, sheet8x1Rev, dataRange);
%
params16x2Orig = xlsread(xlsParamsFileName, sheet16x2Orig, dataRange);
params16x2Rev = xlsread(xlsParamsFileName, sheet16x2Rev, dataRange);

params24x3Orig = xlsread(xlsParamsFileName, sheet24x3Orig, dataRange);
params24x3Rev = xlsread(xlsParamsFileName, sheet24x3Rev, dataRange);
%
testCaseIndex = 1;
selectedTestCase = testCase(testCaseIndex, :);
testPath = selectedTestCase{testCaseIndex};
imageSet = LoadOutexImages(selectedTestCase);

lfpNumNeighbors = 8;
lfpRadius = 1;
TestVistexParams(params8x1Orig, false, 'params8x1Orig', testPath, imageSet,
testCaseIndex, selectedTestCase, lfpWindowSize, lfpNumNeighbors, lfpRadius,
lfpSamplingFunction);
TestVistexParams(params8x1Rev, true, 'params8x1Rev', testPath, imageSet,
testCaseIndex, selectedTestCase, lfpWindowSize, lfpNumNeighbors, lfpRadius,
lfpSamplingFunction);
%{
```

```
lfpNumNeighbors = 16;
lfpRadius = 2;
TestVistexParams(params16x2Orig, false, 'params16x2Orig', testPath, imageSet,
testCaseIndex, selectedTestCase, lfpWindowSize, lfpNumNeighbors, lfpRadius,
lfpSamplingFunction);
TestVistexParams(params16x2Rev, true, 'params16x2Rev', testPath, imageSet,
testCaseIndex, selectedTestCase, lfpWindowSize, lfpNumNeighbors, lfpRadius,
lfpSamplingFunction);

lfpNumNeighbors = 24;
lfpRadius = 3;
TestVistexParams(params24x3Orig, false, 'params24x3Orig', testPath, imageSet,
testCaseIndex, selectedTestCase, lfpWindowSize, lfpNumNeighbors, lfpRadius,
lfpSamplingFunction);
TestVistexParams(params24x3Rev, true, 'params24x3Rev', testPath, imageSet,
testCaseIndex, selectedTestCase, lfpWindowSize, lfpNumNeighbors, lfpRadius,
lfpSamplingFunction);
%}
% Finish the test
disp('*****');
fprintf('\n');
disp('VistexTest finished.');
beep;
```



## APÊNDICE F – CÓDIGO FONTE EXCLUSIVO PARA TREINO E TESTE COM O VISTEX

```

%%%%%%%%%%%%%%%
% ReadOutexTxt gets picture IDs and class IDs from txt for Outex Database
% [filenames, classIDs] = ReadOutexTxt(txtfile) gets picture IDs and class
% IDs from TXT file for Outex Database
%
% Version 1.0
% Authors: Zhenhua Guo, Lei Zhang and David Zhang
% Copyright @ Biometrics Research Centre, the Hong Kong Polytechnic University
%%%%%%%%%%%%%%%
function [filenames, classIDs] = ReadOutexTxt(testPath, txtfile)

useAlternativeFormat = false;
fid = fopen(txtfile, 'r');
firstLine = true;
i = 0;

while 1
    tline = fgetl(fid);

    if ~ischar(tline)
        break;
    end

    if firstLine == true
        if length(strfind(tline, ' ')) == 0 % header - number of imgs
            continue;
        end

        firstLine = false;
    end

    index = findstr(tline, '.');

    if isempty(index)
        index = findstr(tline, ' ');
    end

    i = i+1;
    filenames(i) = str2num(tline(1:index-1))+1; % the picture ID starts from 0, but
the index of Matlab array starts from 1

    index = findstr(tline, ' ');

    if (useAlternativeFormat == true)
        classIDs(i) = str2num(tline(index:end-3)); % Contrib_TC_00000 - CONTRIB
ONLY!
    else
        classIDs(i) = str2num(tline(index:end));
    end;
end

fclose(fid);
end

%%%%%%%%%%%%%%%
% GetTrainningTestSets - Monta duas listas com os identificadores das
% amostras de treino e suas classes correspondentes.

```

```

%%%%%
function [trainIDs, trainClassIDs] = GetTrainningTestSets(reverseOrder, testPath,
numSamples)

% read picture ID of training and test samples, and read class ID
% of training and test samples
trainData = sprintf('%s%03d\\train.txt', testPath, 0);
[trainIDs, trainClassIDs] = ReadOutexTxt(testPath, trainData);

numRows = 8;
numCols = 8;
numImgClasses = 54;

idMatrix = reshape(trainIDs, numRows, numCols, numImgClasses);
classIDsMatrix = reshape(trainClassIDs, numRows, numCols, numImgClasses);

trainingIDsSection = idMatrix(1:(numRows/2), :, :);
trainClassIDsSection = classIDsMatrix(1:(numRows/2), :, :);
testIDsSection = idMatrix((numRows/2)+1:numRows, :, :);
testClassIDsSection = classIDsMatrix((numRows/2)+1:numRows, :, :);

ids=[];
classIDs=[];

for i = 1 : numImgClasses
    % Ids
    tempTrainingIDsSection = trainingIDsSection(:, :, i)';
    trainningIDsLine = tempTrainingIDsSection(:)';

    tempTestIDsSection = testIDsSection(:, :, i)';
    tempTestIDsLine = tempTestIDsSection(:)';
    idsLine = cat(2, trainningIDsLine, tempTestIDsLine);

    if reverseOrder == false % original order
        ids(:, :, i) = idsLine;
    else
        ids(:, :, i) = fliplr(idsLine);
    end

    % class Ids
    tempTrainClassIDsSection = trainClassIDsSection(:, :, i)';
    tempTrainClassIDsLine = tempTrainClassIDsSection(:)';

    tempTestClassIDsSection = testClassIDsSection(:, :, i)';
    tempTestClassIDsLine = tempTestClassIDsSection(:)';

    classIDsLine = cat(2, tempTrainClassIDsLine, tempTestClassIDsLine);

    if reverseOrder == false % original order
        classIDs(:, :, i) = classIDsLine;
    else
        classIDs(:, :, i) = fliplr(classIDsLine);
    end
end

trainIDs = ids(1, 1:numSamples, :);
trainClassIDs = classIDs(1, 1:numSamples, :);

trainIDs = sort(trainIDs(:)');
trainClassIDs = sort(trainClassIDs(:)');
end

```

**APÊNDICE G – PARÂMETROS DO SAMPLED LFP SINTONIZADOS PARA OS EXPERIMENTOS COM O ÁLBUM DE BRODATZ E OUTEX**

Experimento	Vizinhança de análise	Beta	Núm. bins
Brodatz I	(P=8, R=1)	0,6	150
Brodatz I	(P=16, R=2)	0,6	160
Brodatz I	(P=24, R=3)	2,5	190
Brodatz II	(P=8, R=1)	1,25	256
Brodatz II	(P=16, R=2)	0,75	256
Brodatz II	(P=24, R=3)	0,95	256
Brodatz III	(P=8, R=1)	0,8	256
Brodatz III	(P=16, R=2)	0,2	256
Brodatz III	(P=24, R=3)	0,1	256
Outex TC_00010 (amostras 128x128)	(P=8, R=1)	1,55	130
Outex TC_00010 (amostras 128x128)	(P=16, R=2)	0,85	30
Outex TC_00010 (amostras 128x128)	(P=24, R=3)	0,7	256
Outex TC_00010 (amostras 64x64)	(P=8, R=1)	0,75	90
Outex TC_00010 (amostras 64x64)	(P=16, R=2)	0,35	70
Outex TC_00010 (amostras 64x64)	(P=24, R=3)	0,35	256
Outex TC_00012 (amostras 128x128)	(P=8, R=1)	0,6	150
Outex TC_00012 (amostras 128x128)	(P=16, R=2)	1,35	256
Outex TC_00012 (amostras 128x128)	(P=24, R=3)	1,25	256
Outex TC_00012 (amostras 64x64)	(P=8, R=1)	0,7	256
Outex TC_00012 (amostras 64x64)	(P=16, R=2)	1,35	256
Outex TC_00012 (amostras 64x64)	(P=24, R=3)	2,3	256

**APÊNDICE H – PARÂMETROS SINTONIZADOS E SENSIBILIDADE DO  
SAMPLED LFP PARA O EXPERIMENTOS COM O VISTEX**

Sampled LFP	Vizinhança de análise (P=8, R=1)					
	Sequencia ordenada			Sequencia reversa		
Número de amostras de treinamento	beta	Número de bins	Sensibilidade	beta	Número de bins	Sensibilidade
1	0,1	256	37,48%	0,1	256	44,44%
4	0,45	55	49,23%	0,5	65	62,06%
7	0,7	50	49,91%	0,55	210	66,94%
10	0,55	95	56,89%	0,75	190	66,40%
13	0,55	100	58,22%	0,75	190	68,74%
16	0,5	110	60,10%	0,55	75	72,88%
19	0,55	120	61,01%	0,65	80	71,65%
22	0,55	130	62,04%	0,65	90	73,13%
25	0,55	120	63,22%	0,6	90	75,23%
28	0,55	120	63,93%	0,6	90	75,93%
31	0,55	120	64,68%	0,55	130	79,07%
32	0,55	120	65,21%	0,55	140	79,72%
33	0,6	110	64,90%	0,55	140	80,01%
34	0,55	65	65,33%	0,55	140	80,12%
37	0,55	65	66,60%	0,55	130	79,61%
40	0,55	100	69,75%	0,55	80	78,10%
43	0,55	75	70,59%	0,55	110	78,92%
46	0,55	60	70,58%	0,55	110	78,60%
49	0,55	80	72,97%	0,55	130	79,38%
52	0,55	120	75,35%	0,55	100	78,29%
55	0,55	120	76,37%	0,55	130	79,12%
58	0,55	120	77,73%	0,55	120	79,24%
61	0,55	120	78,44%	0,55	120	79,08%
64	0,55	120	79,11%	0,55	120	79,11%

Sampled LFP	Vizinhança de análise (P=16, R=2)					
	Sequencia ordenada			Sequencia reversa		
Número de amostras de treinamento	beta	Número de bins	Sensibilidade	beta	Número de bins	Sensibilidade
1	0,1	256	38,66%	0,1	256	48,42%
4	0,3	256	55,26%	0,25	120	69,13%
7	0,35	256	58,12%	0,2	200	73,37%
10	0,25	70	60,92%	0,15	256	73,82%
13	0,3	100	61,17%	0,25	210	78,17%
16	0,25	170	64,27%	0,15	256	77,86%
19	0,25	210	65,23%	0,3	180	81,23%
22	0,25	170	68,24%	0,3	210	82,68%
25	0,25	250	67,94%	0,25	180	84,40%
28	0,25	170	69,64%	0,25	180	85,32%
31	0,25	170	70,65%	0,25	150	85,79%
32	0,25	170	71,01%	0,25	180	86,33%
33	0,25	170	71,78%	0,3	210	85,53%
34	0,2	240	71,55%	0,3	210	85,53%
37	0,2	240	73,56%	0,3	210	85,21%
40	0,2	50	73,31%	0,3	210	84,87%
43	0,2	50	74,83%	0,3	210	84,81%
46	0,2	50	76,90%	0,3	240	84,77%
49	0,25	180	79,92%	0,25	220	85,24%
52	0,3	220	80,61%	0,25	220	85,20%
55	0,3	220	82,08%	0,3	240	84,32%
58	0,25	170	83,62%	0,3	240	84,06%
61	0,25	170	84,24%	0,3	210	84,40%
64	0,3	210	84,49%	0,3	210	84,49%

**APÊNDICE I –SENSIBILIDADE DO LBP PARA O EXPERIMENTO COM O VISTEX**

LBP riu2	Vizinhança de análise (P=8, R=1)		Vizinhança de análise (P=16, R=2)	
	Sensibilidade		Sensibilidade	
	Sequencia ordenada	Sequencia reversa	Sequencia ordenada	Sequencia reversa
Número de amostras de treinamento				
1	27,24%	29,96%	33,62%	36,41%
4	31,94%	40,23%	42,40%	53,53%
7	35,87%	42,78%	44,87%	55,43%
10	38,37%	44,91%	46,99%	58,40%
13	39,23%	45,80%	48,43%	60,39%
16	39,98%	46,46%	49,25%	61,96%
19	40,86%	48,03%	49,57%	63,05%
22	41,72%	49,05%	50,12%	64,35%
25	42,96%	50,51%	50,89%	65,51%
28	43,73%	51,15%	51,07%	66,54%
31	44,08%	51,49%	51,95%	67,90%
32	44,30%	51,66%	52,07%	68,26%
33	45,08%	51,79%	52,68%	68,06%
34	45,70%	51,66%	53,85%	67,97%
37	46,30%	51,82%	55,37%	67,89%
40	47,72%	52,18%	57,01%	67,70%
43	48,55%	52,85%	58,04%	67,78%
46	49,33%	53,32%	59,97%	67,64%
49	50,45%	53,70%	61,11%	67,54%
52	51,25%	53,75%	62,12%	67,22%
55	52,12%	53,53%	63,78%	67,19%
58	53,05%	53,86%	65,21%	67,25%
61	53,35%	53,86%	66,31%	67,19%
64	54,04%	54,04%	67,17%	67,17%