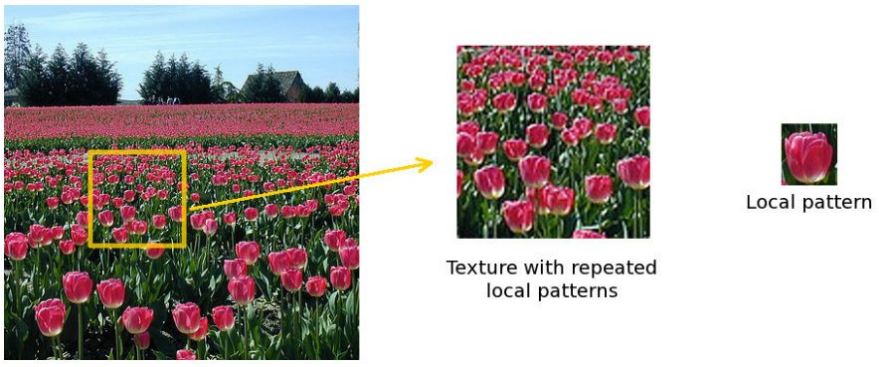
Trabalho 4 - Tópicos em Informática 11
Tema 1: Implementação do descritor de textura Local binary pattern (LBP)
Membros:
Guilherme Nishi Kanashiro, RA: 628298
Leonardo Utida Alcântara, RA: 628182
Rodolfo Krambeck Asbahr, RA: 628042
Tiago Bachiega de Almeida, RA: 628247

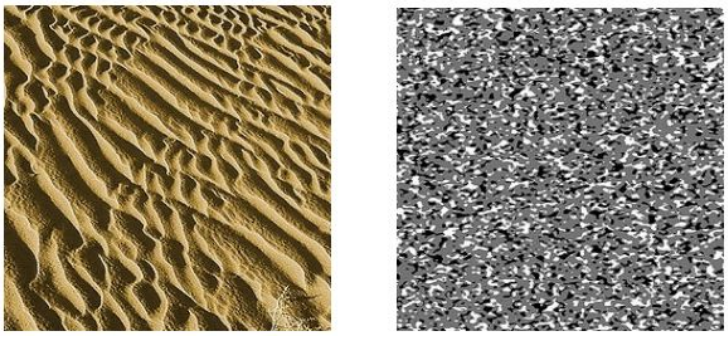
Introdução

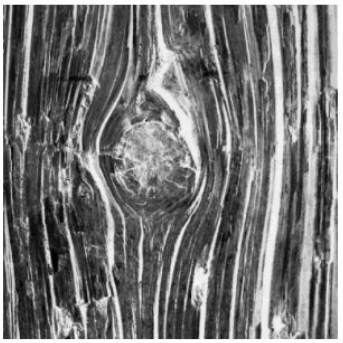
Uma forma muito importante de se caracterizar uma imagem em processamento de imagens é o uso de descritores de textura. Para os seres humanos a textura está relacionado ao tato, como por exemplo, nas diferenças entre superfícies ásperas ou lisas. Já para imagens, texturas são definidas como as diferenças locais nos níveis de intensidade entre um pixel e seus vizinhos.

A textura de uma imagem em geral pode ser classificada em 3 grandes divisões: repetição, estocástica ou ambas. A repetição ocorre quando temos um padrão local que se repete diversas vezes ao longo da imagem. A foto abaixo mostra bem esse conceito em uma imagem de um campo de flores, onde grande parte da textura da imagem foi definida por um padrão local (flor) que se repete diversas vezes.



Já imagens com texturas estocásticas são aquelas em que a textura foi gerada com um fator aleatório, ou seja, não há um padrão de repetição, porém claramente existem linhas ou objetos com variações de intensidade similares ao longo da imagem. Abaixo vemos 4 exemplos desse tipo de textura, os dois primeiros (mais acima) mostram uma imagem de areia e uma imagem de ruído que claramente possuem variações similares nas intensidades e cores dos pixels, porém eles não seguem um padrão. As duas imagens abaixo mostram riscos e as fibras da madeira que também são gerados aleatoriamente, porém nessas imagens vemos que a textura é definida principalmente por fortes diferenças de intensidade ao longo do eixo horizontal.



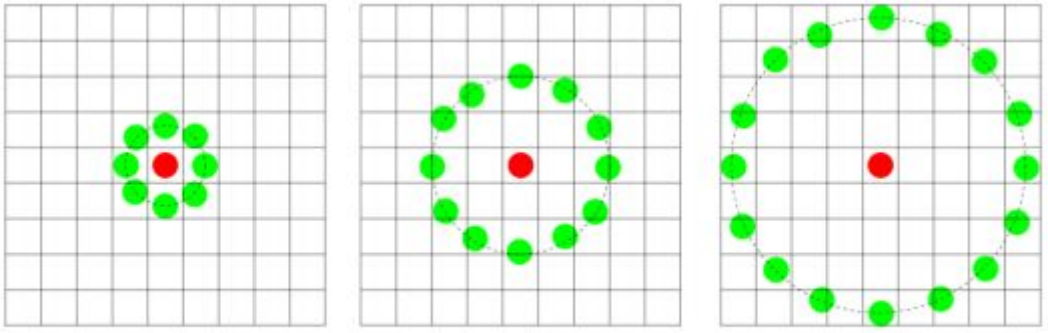


Local Binary Patterns (LBP)

LBP é uma técnica de descrição de textura muito famosa que foi descrita pela primeira vez em 1994 e desde então vem sido muito utilizada para realizar classificação de texturas e classificação de imagens, principalmente pelo fato de ela ser capaz de gerar muitas propriedades que representam uma imagem e que essas propriedades podem ser usadas como atributos de um classificador.

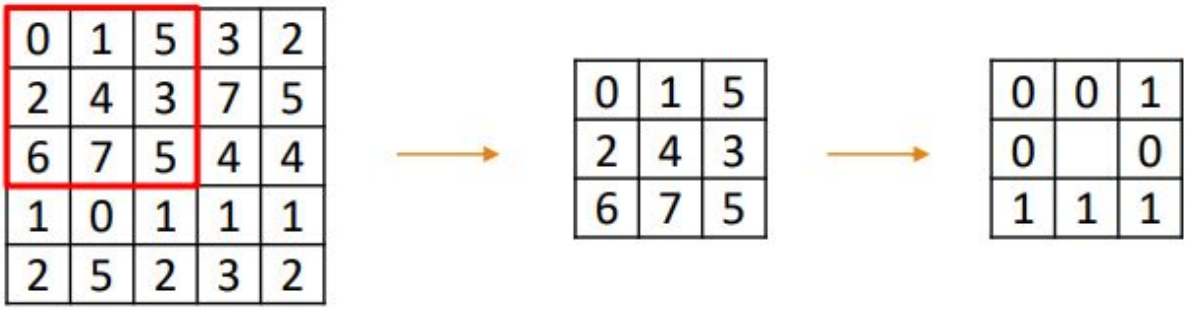
O LBP cria esses atributos estudando a vizinhança de quase todos os pixels da imagem. O LBP considera a vizinhança de um pixel como o conjunto de P pontos de amostragem em um círculo de raio R, sendo o centro desse círculo o ponto a ser estudado. Abaixo temos 3 exemplos muito comuns de vizinhanças que são usadas em processamento de imagens, com múltiplos valores para P e para R. É importante notar que quanto maior o valor de R, ou seja, quando mais distante a amostragem estará do pixel, é necessário que o número P também aumente para conseguirmos ter uma boa amostragem da vizinhança estudada.

Para este trabalho iremos apenas focar no primeiro caso (mais a esquerda), onde analisamos a vizinhança direta do pixel composta pelos 8 pixels que contornam o pixel central.

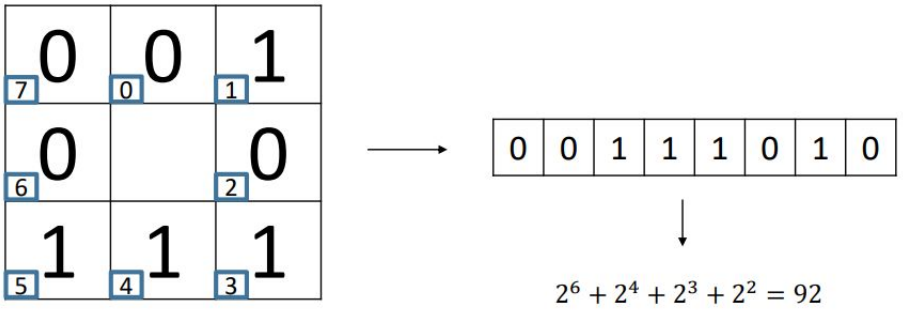


$P = 8, R = 1$ $P = 12, R = 2$ $P = 16, R = 4$

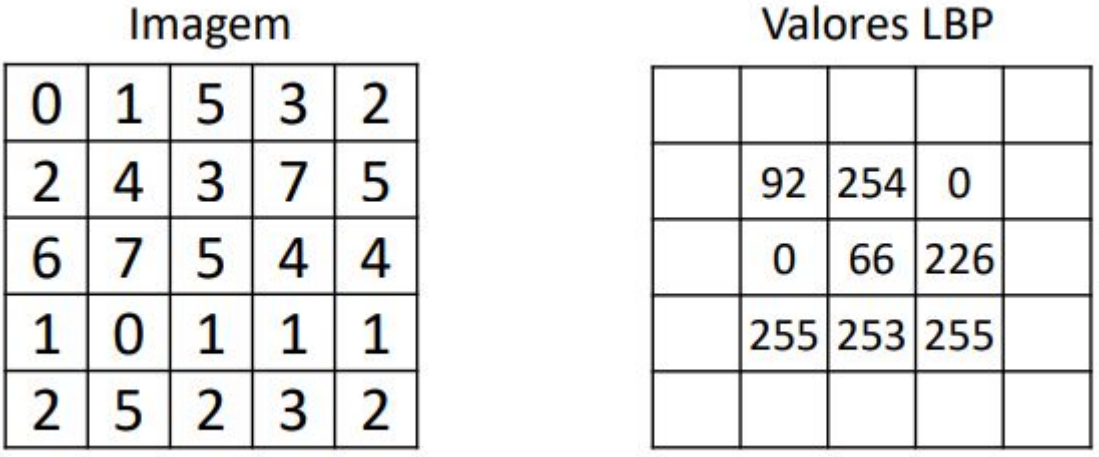
A ideia central do algoritmo é que para uma dada uma imagem de entrada de tamanho [N,M], analisamos cada pixel da imagem (exceto pixels de borda) e em seguida comparamos o valor do pixel central com cada vizinho. Se o pixel central for maior do que o vizinho, associamos o valor 0 com o vizinho, caso contrário, o valor 1 é associado com o vizinho. A imagem abaixo mostra esse processo para o pixel na posição [1,1] e podemos ver que para todos os valores menores que 4 ao redor do pixel central tem o valor 0 associados a eles e para todos os valores maiores ou iguais a 4 temos o valor 1 associado.



O padrão obtido pelo processo descrito acima resulta em uma cadeia de valores de 0s e 1s que pode ser vista como uma sequência de 8 bits. Por padrão definimos que o bit 0 é o vizinho de cima do pixel referência, e que a sequência é percorrida no sentido horário, porém poderíamos ter escolhido outros padrões para definir a cadeia de bit. Com isso, podemos considerar que a sequência obtida representa um número inteiro definido por 8 bits, ou seja no intervalo 0 até 255, que representa um dos tipos de textura local possível de imagens. A imagem abaixo mostra um exemplo da tradução dos valores de 0s e 1s obtidos anteriormente para uma cadeia de bits e finalmente para um valor inteiro.



Criamos então uma nova imagem de tamanho [N-1, M-1] e repetimos o processo acima para todos os pixels da imagem (exceto a borda). Ao fim teremos uma nova matriz com valores que representam o tipo de textura definida ao redor de cada pixel da imagem. A imagem abaixo representa um exemplo desse processo.



Após a criação desta nova matriz, podemos criar um histograma da mesma utilizando 256 caixas para guardar a quantidade que cada um dos 256 tipos de textura se repete. Por exemplo, o valor na caixa 5 representa o número de pixels na imagem possuindo

a vizinhança 00000101 (valor 5 em binário). Em outras palavras, o número de pixels para os quais o pixel é menor do que o pixel ao norte e à direita, e menor do que os demais. Esses 256 valores de quantidades são usados como atributos da imagem na tarefa de classificação.

Algoritmo em python

A figura abaixo mostra as duas principais funções implementadas referente aos cálculos dos atributos do LBP. A função getBitsValue() recebe uma imagem "img" e a posição do pixel central através das variáveis "pos_x" e "pos_y" e então cria a cadeia de bits dada as regras definidas no LBP descritas na seção anterior. A função retorna o inteiro representado pela cadeia de bits gerada naquele pixel.

```
def getBitsValue(img, pos_x, pos_y):
    bits = ''
    middleValue = img[pos_x, pos_y]

    if middleValue > img[pos_x-1,pos_y-1]:
        bits += '0'
    else:
        bits += '1'

    if middleValue > img[pos_x,  pos_y-1]:
        bits += '0'
    else:
        bits += '1'

    if middleValue > img[pos_x+1,pos_y-1]:
        bits += '0'
    else:
        bits += '1'

    if middleValue > img[pos_x+1,pos_y]:
        bits += '0'
    else:
        bits += '1'

    if middleValue > img[pos_x+1,pos_y+1]:
        bits += '0'
    else:
        bits += '1'

    if middleValue > img[pos_x,  pos_y+1]:
        bits += '0'
    else:
        bits += '1'

    if middleValue > img[pos_x-1,pos_y+1]:
        bits += '0'
    else:
        bits += '1'

    if middleValue > img[pos_x-1,  pos_y]:
        bits += '0'
    else:
        bits += '1'

    return int(bits, 2)
```

A função LBP_attributes() recebe uma imagem e aplica a função getBits_value() para todos os pixels da imagem, exceto a borda, e vai salvando os valores retornados em uma nova matriz chamada "newMatrix", da qual são retiradas as quantidades que cada um dos possíveis 256 valores de textura apareceu na imagem. Essas quantidades são salvas na lista "listBins".

```
def LBP_attributes(img):
    num_rows, num_cols = img.shape
    newMatrix = np.zeros((num_rows-2, num_cols-2))

    # Para cada pixel da imagem, menos a borda
    for i in range(1,num_rows - 1):
        for j in range(1,num_cols - 1):
            val = getBitsValue(img, i, j)
            newMatrix[i-1, j-1] = val

    listBins = []
    for i in range(256):
        listBins.append(float(np.sum(newMatrix == i)))
    return listBins
```

O classificador criado neste trabalho tem a função de classificar uma imagem nas classes “floresta” e “nuvens”. Temos 4 diretórios chamados “nuvensTest”, “forestTest”, “nuvensTrain”, “florestaTrain”. Os dois primeiros guardam respectivamente 7 imagens de nuvens e 7 imagens de florestas para serem utilizadas como dataset de teste. Nos dois diretórios seguintes estão 30 imagens de nuvens e 30 imagens de florestas para serem usadas como dataset de treino. As imagens foram retiradas de uma busca no google pelas palavras “nuvens” e “florestas” e todas possuem a resolução de 1280x720.

O algoritmo cria as bases de dados de treino e teste, convertendo as imagens para escala de cinza e realizando uma interpolação para diminuir o tamanho das imagens para 320x180 (a fim de aumentar a velocidade da criação dos atributos com o LBP) e então extrai os 256 atributos de cada imagem com as função LBP_attributes() e salva esses atributos nas bases de dados. Com a base de dados de treino o algoritmo treina um classificador KNN, no qual o valor 1 nas classes representa uma imagem de floresta, enquanto o valor 2 representa uma imagem de nuvem. A imagem abaixo mostra a criação deste classificador.

```
# Realiza o treinamento de um classificador KNN para o dataset de treino criado acima
knn = cv2.ml.KNearest_create()
knn.train(LBP_train_data, cv2.ml.ROW_SAMPLE, labels)
```

O classificador criado acima é utilizado para classificar os dados da base de dados de treino e a acurácia desse experimento treino-teste é calculada e mostrada.

```
# Realiza a classificação dos dados de teste e salva os resultados
ret, result, neighbors, dist = knn.findNearest(LBP_test_data, k=3)
print('Done')
```

```
Done
```

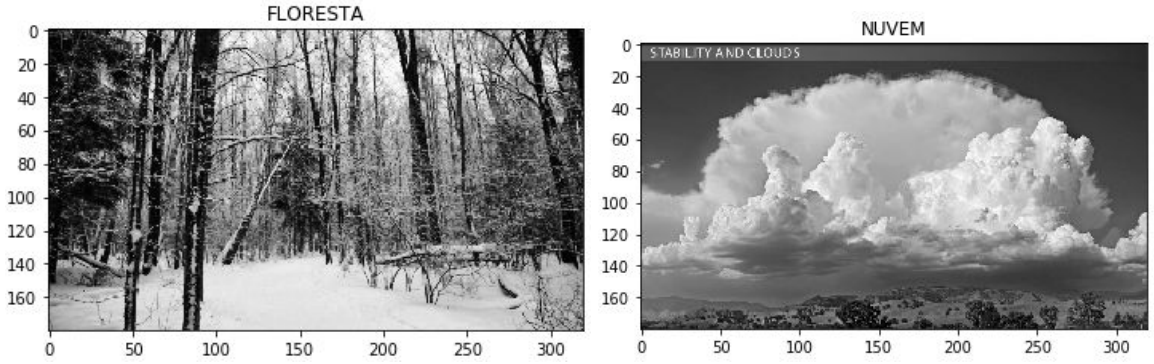
```
print(result)
# 1 = Florestas
# 2 = Nuvens
```

```
[[1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [2.]
 [2.]
 [2.]
 [2.]
 [2.]
 [2.]
 [2.]]
```

```
# Compara o array de resultados com o de classes do test set calculado a
# acurácia do classificador
array_comp = result == labels_test
num_equal = np.sum(array_comp)
print('Acurácia = ', 100*num_equal/len(result))
```

```
Acurácia =  100.0
```

Por fim o código mostra as 14 imagens da base de dados de teste atribuindo como título da imagem as classes obtidas com a classificação. As imagens abaixo mostram a saída dessa parte do código:



Todas as 14 imagens de teste do experimento realizado foram classificadas corretamente. O interessante é que mesmo com um número baixo de imagens de treino o classificador se saiu bem na tarefa. Por fim, vale destacar que a imagem abaixo se encontra no dataset de teste para as florestas e que todas as imagens de floresta no dataset de treino são de florestas **verdes**, porém por se tratar de uma classificação por texturas, a cor não influencia nesse caso, fazendo que a classificação de uma floresta com neve também seja possível.