Relatório: Segmentação de Imagens Baseada em Texturas

Lucas Nogueira
lfn19@inf.ufpr.br

Pedro Beber
pbqv20@inf.ufpr.br

Abril de 2025

Sumário

1 Introdução

Este relatório apresenta uma análise das técnicas empregadas para segmentação de imagens baseada em texturas, conforme implementado no código Python fornecido. A segmentação de imagens é um processo fundamental em visão computacional que visa dividir uma imagem em regiões significativas que compartilham características semelhantes. Particularmente, a segmentação baseada em texturas é relevante quando as regiões de interesse não podem ser distinguidas apenas por intensidade ou cor, mas apresentam padrões estruturais distintos.

O foi utilizada uma abordagem que combina extração de características de textura através de múltiplos filtros e uma técnica de agrupamento não supervisionado para identificar regiões com texturas similares. Esta metodologia é especialmente útil em aplicações como análise de imagens médicas, sensoriamento remoto, inspeção industrial e reconhecimento de padrões, apesar de apresentar resultados inferiores a de outras técnicas utilizando *Machine Learning*, a técnicas clássicas implementadas chegam em resultados com menos processamento e sem a necessidade de treinamento prévio com conjuntos de dados. O trabalho foi desenvolvido no Google Colab e está disponível aqui.

2 Metodologia

A metodologia implementada pode ser dividida nas seguintes etapas principais:

2.1 Pré-processamento da Imagem

É feito a leitura de uma imagem em escala de cinza utilizando a biblioteca OpenCV. Esta escolha de representação concentra a análise nos padrões de textura presentes na imagem, independentemente das informações de cor.

```
imagem = cv2.imread(input, cv2.
    IMREAD_GRAYSCALE)
```

2.2 Extração de Características de Textura

2.2.1 Aplicação de Filtros de Textura

Foi implementada a função `sliding_kernel_mean` que aplica diferentes kernels (filtros) à imagem de entrada sem sobreposição, calculando a média das

operações em cada bloco. Esta abordagem permite capturar características locais de textura em diferentes regiões da imagem.

```
def sliding_kernel_mean(matrix,
    kernel):
  m_rows, m_cols = matrix.shape
  k_rows, k_cols = kernel.shape
  result = []
  for i in range(0, m_rows, k_rows)
   :
      for j in range(0, m_cols,
   k_cols):
        block = matrix[i:i+k_rows
   , j:j+k_cols]
        if block.shape != kernel.
   shape:
          continue
        multiplied = block *
   kernel
        mean_value = multiplied.
   mean()
        result.append(mean_value)
  return np.array(result)
```

2.2.2 Biblioteca de Kernels

Foi criada uma variedade de kernels para capturar diferentes aspectos das texturas:

1. **Filtro Laplaciano**: Detecta mudanças de segunda ordem, útil para realçar áreas de rápida mudança de intensidade.

```
laplacian = np.array([
  [0, -1, 0],
  [-1, 4, -1],
  [0, -1, 0]])
```

2. **Filtros de Borda Horizontal e Vertical**: Detectam transições de intensidade em direções específicas.

```
h_edge = np.array([
  [-1, -1, -1],
  [0, 0, 0],
  [1, 1, 1]])
```

```
v_edge = np.array([
  [-1, 0, 1],
  [-1, 0, 1],
  [-1, 0, 1]])
```

3. **Operadores de Scharr Horizontais e Verticais**: Aproximações melhoradas dos gradientes direcionais, mais robustos a ruídos.

```
h_scharr = np.array([
  [3, 0, -3],
  [0, 0, 0],
  [-3, -10, -3]])

v_scharr = np.array([
  [3, 10, 3],
  [0, 0, 0],
  [-3, -10, -3]])
```

A função `run_kernel_lib` aplica todos estes filtros à imagem, gerando um vetor de características para cada bloco da imagem:

```
def run_kernel_lib(matrix):
  # definição de kernels
  return np.array([
   sliding_kernel_mean(matrix,k)
   for k in [laplacian,h_edge,
   v_edge,h_scharr,v_scharr]])
```

2.3 Abordagem Multi-escala

Optou-se por manter um banco de kernels em uma única escala, então para obter um resultado multi-escala, os filtros foram aplicados em diferentes resoluções da imagem. Isto é realizado através da criação de uma pirâmide Gaussiana que progressivamente reduz a resolução da imagem de entrada:

```
result = [run_kernel_lib(imagem)]
layer = imagem.copy()

for i in range(4):
```

```
    layer = cv2.pyrDown(cv2.
     GaussianBlur(layer,(3,3),cv2.
     BORDER_DEFAULT))
    one_dimension_processed_img =
     run_kernel_lib(layer)
    result.append(
     one_dimension_processed_img)
```

Este procedimento permite capturar características de textura em diferentes escalas espaciais, o que é particularmente útil para identificar padrões que podem se manifestar em diferentes níveis de detalhe.

2.4 Segmentação por Clustering

Após a extração das características de textura, o código utiliza o algoritmo K-means para agrupar regiões com texturas similares:

```
kmeans = KMeans(n_clusters=2,
    random_state=42).fit(
    result_kernels)
```

O K-means é um algoritmo de clustering não supervisionado que agrupa os dados em um número predefinido de clusters (neste caso, 2), baseando-se na distância euclidiana entre os pontos no espaço de características. O resultado é um mapa de rótulos que pode ser visualizado como uma imagem segmentada:

```
labels = kmeans.labels_.reshape((x,
    y))
```

3 Visualização dos Resultados

3.1 Exemplos de resultados

Nessa secção é apresentado os resultados obtidos com um subconjunto do conjunto de 16 imagens de teste, e considerações. Na figura 1, é exibido o resultado da segmentação de uma paisagem urbana.

É possível verificar que a maioria da região dos prédios foi segmentada corretamente, apenas algumas regiões foram desconsideradas pelo método. Como hipótese, sugere-se que a textura das regiões que não foram segmentadas corretamente, serem semelhantes ao céu, que contem uma textura lisa.

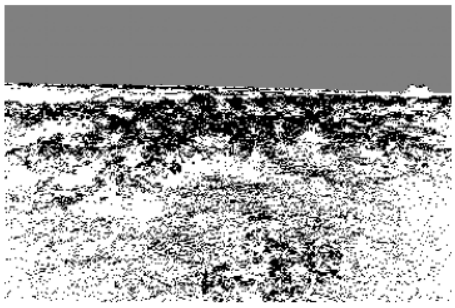
(a) Imagem original

(b) Imagem depois da segmentação

Figura 1: Resultados da segmentação: (a) a imagem original e (b) a imagem segmentada utilizando o método proposto

A seguir segue outra imagem de exemplo a ser segmentada (Figura 2, dessa vez utilizando 3 clusters, correspondente aos três tipos de textura identificados na imagem. O método foi eficiente em identificar duas texturas de arbustos, vegetação rasteira e céu. No entanto, apresenta certas inconsistências para certos pixeis da vegetação rasteira, não segmenta completamente os arbustos e considera que as árvores do plano de fundo são vegetação rasteira. Um ponto positivo marcante foi a diferenciação da textura do céu em comparação a vegetação.

(a) Imagem original



(b) Imagem depois da segmentação

Figura 2: Resultados da segmentação: (a) a imagem original e (b) a imagem segmentada utilizando o método proposto

Por fim, segue-se a análise de uma figura (Figura 3) mais heterogênea, contendo ao todo 6 texturas identificadas. As texturas são as seguintes: vegetação, carro, excluindo-se rodas que é outra textura, estradas e céu.

A segmentação realizada demonstrou desempenho satisfatório, especialmente na identificação das regiões de vegetação e céu. Esses elementos, por possuírem características de textura e coloração relativamente homogêneas,

(a) Imagem original

(b) Imagem depois da segmentação

Figura 3: Resultados da segmentação: (a) a imagem original e (b) a imagem segmentada utilizando o método proposto

foram classificados corretamente na maior parte das imagens, evidenciando a adequação do método para distinguir áreas naturais do cenário.

Entretanto, ao analisar a segmentação do carro, observa-se que a variação de luminosidade impactou negativamente a identificação das suas texturas. Regiões do carro submetidas a diferentes intensidades de luz — como áreas refletidas ou sombreadas — apresentaram resultados inconsistentes, sendo, por vezes, classificadas de maneira incorreta. Isso demonstra uma limitação do método em lidar com objetos cuja aparência visual é fortemente dependente das condições de iluminação.

Com relação às rodas do carro, o algoritmo de segmentação obteve êxito na maior parte dos casos, conseguindo destacá-las do restante do veículo. Todavia, foi percebida certa confusão entre as rodas e as sombras projetadas pelo chassi, principalmente devido à semelhança de intensidade e textura entre essas regiões. Esse comportamento indica que áreas escuras e de baixa variação de textura tendem a ser agrupadas em um mesmo segmento, mesmo quando pertencem a diferentes elementos da cena.

4 Análise Teórica das Técnicas Empregadas

4.1 Filtros para Análise de Textura

Os filtros utilizados no código pertencem à categoria de detectores de borda e operadores diferenciais, que são fundamentais para a análise de textura:

- **Laplaciano**: Este operador de segunda ordem detecta mudanças na

taxa de variação da intensidade, sendo particularmente eficaz em realçar detalhes finos e bordas em várias direções.

- **Filtros de Borda Direcional**: Os kernels de borda horizontal e vertical implementados são simplificações do operador Sobel, capazes de detectar variações de intensidade em direções específicas. Estas características são importantes para texturas com orientações predominantes.

- **Operadores de Scharr**: São versões melhoradas dos operadores de Sobel, oferecendo melhor precisão na detecção de gradientes direcionais, especialmente em imagens com ruído.

4.2 Análise Multi-escala

A análise multi-escala implementada através da pirâmide Gaussiana é uma técnica poderosa por várias razões:

1. **Invariância à escala**: Permite detectar texturas semelhantes que aparecem em diferentes tamanhos.

2. **Captura de padrões hierárquicos**: Algumas texturas são compostas por padrões em diferentes níveis de detalhe, que só podem ser adequadamente caracterizados analisando múltiplas escalas.

3. **Redução de ruído**: As versões de menor resolução naturalmente suavizam ruídos de alta frequência, o que pode melhorar a robustez da segmentação.

4.3 Clustering K-means para Segmentação

O K-means é uma escolha adequada para este problema por várias razões:

1. **Simplicidade e eficiência**: É um algoritmo computacionalmente eficiente e conceitualmente simples.

2. **Não supervisionado**: Não requer dados rotulados, o que é vantajoso em aplicações de segmentação onde anotações manuais podem ser custosas.

3. **Baseado em distância**: Agrupa pontos baseados em sua similaridade no espaço de características, o que se alinha bem com o objetivo de encontrar regiões com texturas semelhantes.

No entanto, o K-means também apresenta limitações:

1. **Número fixo de clusters**: Requer especificação prévia do número de clusters (regiões).
2. **Sensibilidade à inicialização**: Diferentes inicializações podem levar a resultados diferentes.
3. **Forma dos clusters**: Assume clusters esféricos no espaço de características, o que pode não ser apropriado para todos os tipos de textura.

5 Conclusão

Neste trabalho, foi apresentada e analisada uma abordagem clássica de segmentação de imagens baseada em texturas, empregando filtros convolucionais para extração de características e o algoritmo K-means para agrupamento não supervisionado. Os experimentos realizados evidenciaram a eficiência dessa metodologia na distinção de regiões com padrões texturais bem definidos, especialmente em elementos como vegetação e céu, frequentemente presentes em cenas naturais.

A análise multi-escala demonstrou ser uma estratégia relevante para lidar com padrões de diferentes tamanhos, ampliando a capacidade de generalização dos filtros utilizados. Ainda assim, fica claro que a integração de informações adicionais, tais como cor, contexto espacial ou métodos supervisionados mais avançados, pode aprimorar o desempenho em cenários mais complexos.

Em síntese, o estudo mostra que técnicas clássicas de segmentação ainda apresentam valor em certas aplicações, especialmente pela simplicidade e ausência de necessidade de dados rotulados para treinamento. Sugere-se como trabalhos futuros a investigação de abordagens híbridas, que combinem métodos tradicionais e modernos, visando obter segmentações mais robustas e precisas em imagens com variados graus de complexidade textural.

6 Repositório GIT

Repositório para acesso ao código do projeto: `https://github.com/PedroBVidal/T1_texture_segmentation`.