Segmentação Baseada em Textura usando Filtros Direcionais

Luan Marko Kujavski GRR20221236
João Pedro Vicente Ramalho GRR20224169
Heloisa Benedet Mendes GRR20221248
Departamento de Informática
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
{lmk22, jpvr22, hbm22}@inf.ufpr.br

***Resumo*—Este trabalho propõe um método para segmentação de imagens baseado em características de textura utilizando filtros direcionais aplicados em três escalas. São empregados kernels nas orientações horizontal, vertical, diagonais (45° e 135°) e circular, processando as imagens em três estágios de escala. As janelas de pixels são descritas pela média dos valores filtrados e agrupadas via distância euclidiana com limiar fixo. O método foi validado em imagens de um dataset próprio.**

I. Introdução

A segmentação baseada em textura é essencial para identificar regiões homogêneas em visão computacional. Este trabalho explora o uso de filtros espaciais direcionais clássicos para extrair características de textura, sem recorrer a técnicas de aprendizado profundo.

A proposta é dividida em três etapas principais:

1) Aplicação de filtros direcionais em múltiplas escalas.
2) Extração de vetores de características por janelas de pixels.
3) Agrupamento das regiões com base na similaridade.

Este relatório descreve a construção do método, abordando os filtros utilizados, o processamento multi-escala, a extração de características e o processo de agrupamento e segmentação.

O sistema foi implementado em Python 3.10, utilizando amplamente a biblioteca *Open Source Computer Vision Library* (OpenCV), e validado com um conjunto próprio de imagens que apresenta uma grande variedade de classes e texturas. O projeto é reprodutível e está disponível no repositório GitHub do projeto.

II. Metodologia

Esta seção descreve os recursos utilizados na implementação do método de segmentação de imagens proposto. A partir desses elementos, a estratégia é aplicada conforme a estrutura em três etapas apresentada anteriormente.

A. Conjunto de Filtros

Foram implementados cinco filtros com as direções Horizontal, Vertical, Diagonal 45°, Diagonal 135° e Circular. Cada filtro é aplicado em três escalas distintas: 3×3, 5×5 e 7×7. As definições dos filtros foram organizadas em um arquivo YAML, conforme exemplificado a seguir:

```
filtros:
  horizontal:
    kernels: [3,5,7]
    filtros:
      kernel_3: [...]  % escala 3x3
      kernel_5: [...]  % escala 5x5
      kernel_7: [...]  % escala 7x7
  vertical:
    kernels: [3,5,7]
    filtros:
      kernel_3: [...]
      kernel_5: [...]
      kernel_7: [...]
      ...
```

B. Processamento Multi-Escala

Para o processamento em múltiplas escalas, foi adotada uma abordagem sequencial:

- Redimensionamento inicial da imagem para R×R pixels, sendo R o tamanho de redimensionamento escolhido;
- Convolução com os kernels direcionais na imagem correspondente a cada escala.

C. Extração de Características

A extração de características requer a determinação de diversos parâmetros, que são armazenados no arquivo `args.json` após o início do programa. Os principais parâmetros desta etapa sãa o tamanho da janela (W) e a redimensão da imagem (R), que seguem as seguintes restrições:

$$R \geq W \tag{1}$$

$$R \,\%\, W = 0 \tag{2}$$

$$\frac{R^2}{W^2} \leq 256 \tag{3}$$

A Equação 3 limita o número de classes a 256, já que o sistema representa cada classe por um valor distinto de intensidade de cinza.

Para cada filtro definido no arquivo YAML (composto por múltiplos kernels), é aplicada a função `filter2D` sobre a imagem redimensionada. O resultado é então dividido em

janelas de $W \times W$ pixels, nas quais é calculada a média dos valores filtrados.

$$mean_i = \frac{1}{W^2} \sum_{(x,y) \in W} (I * K_i)(x, y) \tag{4}$$

O cálculo é formalizado na Equação 4, em que I representa a imagem processada e K_i o kernel do filtro aplicado. Esse processo gera, para cada janela, um vetor de características com N elementos, onde N corresponde ao número de filtros aplicados.

A extração de características foi implementada no script `extract_textures.py`, que recebe a imagem, aplica os filtros definidos e gera os arquivos de saída `train_features.json` e `test_features.json`.

D. Agrupamento e Segmentação

No `segmentation.py`, carregamos os arquivos de features gerados anteriormente e agrupamos as regiões com base na distância euclidiana. Para cada par de vetores de características $\mathbf{f}$ e $\mathbf{g}$, calculamos:

$$d = \|\mathbf{f} - \mathbf{g}\|_2 = \sqrt{\sum_{i=1}^{n} (f_i - g_i)^2} \tag{5}$$

Regiões cujas distâncias satisfazem $d < T$ (onde T é o limiar definido na linha de comando) são atribuídas à mesma classe. Caso alguma região não seja agrupada com qualquer outra, ela recebe uma nova classe única (singleton).

A saída do processo de segmentação é uma imagem em tons de cinza, na qual cada região recebe um valor de intensidade proporcional à sua classe. Esse mapa de classes é normalizado para o intervalo [0, 255] e então sobreposto à imagem original, lado a lado, facilitando a comparação visual.

III. Resultados e Discussão

O método foi aplicado no dataset com 16 imagens e se mostrou bastante eficaz, com uma boa identificação e separação de texturas homogêneas.

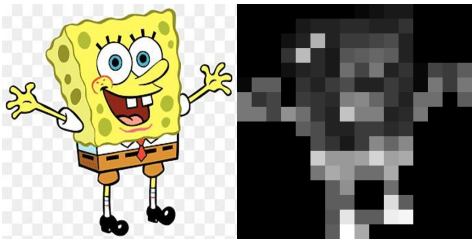
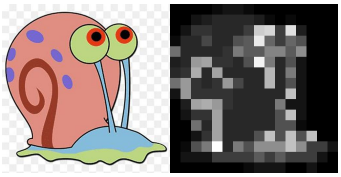
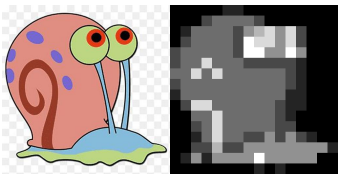


Figura 1: Bob Esponja – W= 32, T= 0.1

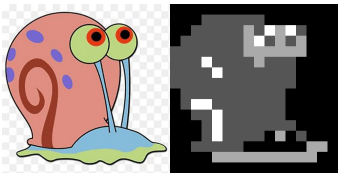
O sistema é altamente sensível ao valor do limiar. Alterações nesse parâmetro influenciam diretamente a quantidade de informação preservada: limiares mais altos resultam em menos detalhes retidos. As imagens a seguir ilustram esse efeito: utilizando a mesma imagem, o mesmo procedimento e o mesmo valor de W, observa-se que, à medida que o limiar aumenta, a quantidade de detalhes visíveis no desenho diminui.



(a) Garry – W= 32, T= 0.1

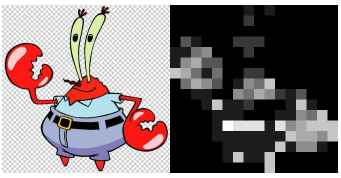


(b) Garry – W= 32, T= 0.2

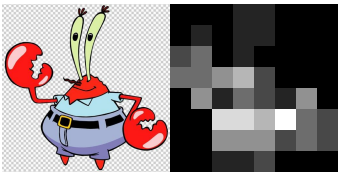


(c) Garry – W= 32, T= 0.3

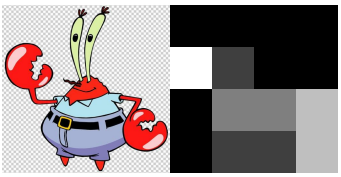
Além disso, o tamanho de W utilizado impacta significativamente o resultado. As imagens a seguir mostram que, à medida que W aumenta, o número de classes identificadas diminui de forma acentuada.



(a) Sirigueijo – W= 32, T= 0.2



(b) Sirigueijo – W= 64, T= 0.2



(c) Sirigueijo – W= 128, T= 0.2

Todas essas relações empíricas são sintetizadas no gráfico representado na Imagem 4, que apresenta de forma clara a curva de decaimento na preservação de detalhes em função de T e a queda exponencial no número de classes identificadas conforme W aumenta.

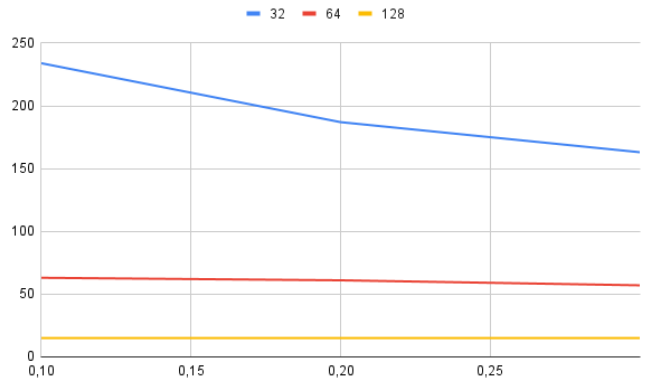


Figura 4: Variação do número de classes identificadas em função de T e W

Esses achados não apenas validam a eficiência do método proposto, mas também fornecem diretrizes práticas para sua aplicação em cenários reais, onde o balanceamento entre precisão e generalização é fundamental.

IV. Classificação por Textura

Extrapolando o desenvolvimento proposto, foi implementado o script `classify.py` responsável pela classificação de imagens com base nas texturas previamente extraídas. Durante a execução, aplica-se o parâmetro `split_factor` para a divisão dos dados em conjuntos de treino e teste. O script também é compatível com o dataset tiny-imagenet, que contém aproximadamente 100 mil imagens de treinamento distribuídas entre 200 classes, além de 50 imagens de teste para cada classe.

V. Conclusão

Conclui-se que o sistema proposto é capaz de identificar e segmentar regiões com base em padrões de textura, utilizando apenas métodos computacionais clássicos, sem a necessidade de técnicas de aprendizado profundo. Os experimentos demonstraram que tanto o valor do limiar quanto o tamanho da janela W exercem influência direta na qualidade da segmentação, afetando a preservação de detalhes e a quantidade de classes identificadas. De modo geral, o método mostrou-se robusto para diferentes texturas e padrões, confirmando sua eficácia nas condições testadas.

Repositório

O código está disponível em:
https://github.com/joaop-vr/computer_vision