

Métodos Para Segmentação dos Pulmões em Imagens de Tomografia Computadorizada

Sumário

Introdução

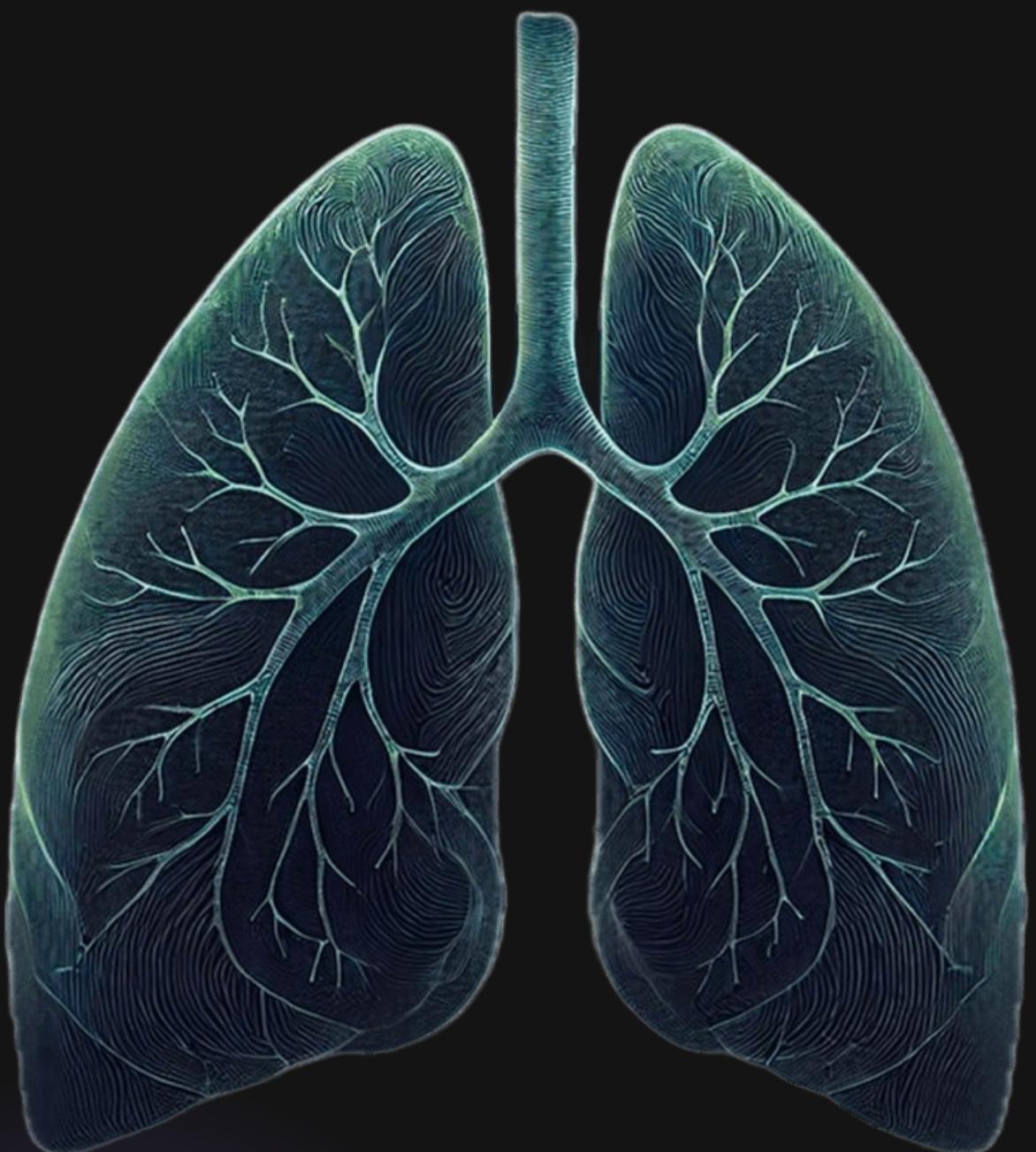
Fundamentação Teórica

Implementação

Resultados

Conclusão

Introdução



Segmentação Pulmonar em Tomografias Computadorizadas (TC):

- A segmentação de imagens médicas é fundamental para análise e diagnóstico;
- Tomografias Computadorizadas fornecem imagens detalhadas dos pulmões, auxiliando no diagnóstico de doenças respiratórias;
- Métodos automáticos de segmentação reduzem erros e aceleram a análise.

Problema e Objetivo

Desafios da segmentação manual:

- Processo demorado (exames podem ter até 1000 imagens);
- Subjetividade (resultados podem variar entre especialistas);
- Erros na delimitação podem afetar diagnósticos.

Objetivo:

- Testar utilização de métodos clássicos de segmentação;
- Desenvolver um algoritmo baseado em contornos ativos para segmentação automática;
- Aumentar precisão e reduzir tempo na análise de TC;
- Melhorar a consistência dos resultados.

Abordagem

Método de Contornos Ativos Crisp

Adaptativo (MCA-CA):

- Técnica que deforma uma curva inicial até se ajustar às bordas do pulmão;
- Utiliza minimização de energia para encontrar os contornos exatos;
- Aplicado em 2D e 3D, sendo avaliado em relação a outros métodos.

Vantagens do MCA-CA:

- Maior precisão na segmentação;
- Redução da necessidade de intervenção manual;
- Aplicação potencial em sistemas de auxílio ao diagnóstico.

Fundamentação Teórica

Segmentação de Imagens e Contornos Ativos

O que é segmentação de imagens?

- Processo de separar regiões de interesse em uma imagem;
- Essencial para análise de exames médicos, como TC do tórax.

Métodos de Contornos Ativos (MCA):

- Técnicas que ajustam uma curva deformável para encontrar as bordas de um objeto;
- O MCA Crisp Adaptativo aprimora métodos anteriores, tornando a segmentação mais precisa e eficiente.

Unidades Hounsfield e Classificação de Tecidos

Imagens DICOM e Unidades Hounsfield (UH)

- As imagens de TC são compostas por pixels com valores em UH, que representam densidades dos tecidos;
- Cada tecido tem um intervalo específico de UH:
 - Ar (pulmão): -1000 a -500 UH;
 - Água/tecidos moles: ~0 UH;
 - Osso: 600 a 2000 UH.

Importância para a segmentação:

O algoritmo utiliza valores de UH para distinguir os pulmões de outras estruturas.

Detecção de Bordas com o Operador Sobel

Identificando contornos em imagens de TC:

- O Operador Sobel é um método clássico para detecção de bordas;
- Calcula variações de intensidade dos pixels para encontrar transições entre regiões.

No algoritmo de segmentação:

- Ajuda a definir os contornos pulmonares;
- Utilizado na energia externa do MCA-CA para melhorar a precisão.

Vantagens do Método Crisp Adaptativo

Principais melhorias do MCA-CA:

- Considera informações de textura e densidade dos tecidos;
- Adapta-se melhor a variações na forma dos pulmões;
- Minimiza erros comuns em segmentação manual.

Aplicações na medicina:

- Detecção de doenças pulmonares (exemplo: DPOC, fibrose);
- Auxílio na tomada de decisão clínica.

Métodos Alternativos de Segmentação

Crescimento de Regiões com a Semente

Fora do Pulmão:

- Baseado na similaridade entre pixels adjacentes;
- Inicia com sementes fora da região pulmonar e expande para pixels semelhantes;
- Critérios de similaridade incluem intensidade, textura e cor;
- Processo iterativo que para quando não há mais pixels elegíveis ou um tamanho limite é atingido.
- **Vantagens:** Controle da região segmentada.
- **Desvantagens:** Sensível a ruídos e dependente da escolha da semente.

Divisão e Fusão de Regiões:

- Divide recursivamente a imagem em regiões menores até atender um critério de homogeneidade;
- Critérios comuns~são variação de intensidade e média da região;
- Se um critério é satisfeito, as regiões são fundidas.
- **Vantagens:** Boa precisão em imagens homogêneas.
- **Desvantagens:** Complexidade computacional e sensibilidade a ruído.

Métodos Alternativos de Segmentação

Watershed:

- Baseado em bacias hidrográficas, tratando a imagem como uma superfície tridimensional;
- Identifica regiões de mínimos locais e expande suas fronteiras até atingir uma linha divisória.
- **Vantagens:** Segmenta bem estruturas com contornos bem definidos.
- **Desvantagens:** Pode supersegmentar regiões sem filtragem adequada.

Limiarização com Média Móvel:

- Adapta dinamicamente o limiar com base na média da vizinhança local;
- Indicada para imagens com iluminação não uniforme.
- **Vantagens:** Simples e eficiente em imagens com variação de brilho.
- **Desvantagens:** Pode falhar em regiões de baixo contraste.

Métodos Alternativos de Segmentação

Limiarização Múltipla:

- Define vários limiares para separar diferentes regiões de intensidade na imagem;
- Eficiente para imagens com histograma multimodal.
- **Vantagens:** Boa segmentação de imagens com múltiplas regiões de interesse.
- **Desvantagens:** Sensível a ruídos e alto custo computacional.

Limiarização com Propriedades Locais:

- Ajusta o limiar de cada pixel considerando características da vizinhança;
- Ideal para imagens com variação espacial de intensidade.
- **Vantagens:** Boa segmentação em imagens com iluminação desigual.
- **Desvantagens:** Pode ser mais demorada que a limiarização global.

Métodos Alternativos de Segmentação

Otsu:

- Define automaticamente um limiar que minimiza a variância intra-classe e maximiza a variância inter-classe;
- Indicado para imagens bimodais (com dois picos distintos no histograma).
- **Vantagens:** Automático e eficaz para imagens com contraste adequado.
- **Desvantagens:** Menos eficiente para imagens com distribuição uniforme de intensidades.

Sauvola:

- Variante da limiarização adaptativa;
- Calcula um limiar para cada pixel usando a média e o desvio padrão da vizinhança.
- **Vantagens:** Eficiente para imagens com variação local de intensidade.
- **Desvantagens:** Pode exigir ajuste fino de parâmetros para diferentes imagens.

Descrição do Algoritmo

Entrada do Algoritmo

Dados de entrada:

- Imagens no formato DICOM de Tomografia Computadorizada do tórax;
- Cada pixel contém valores em Unidades Hounsfield (UH) indicando a densidade dos tecidos.

Por que isso é importante?

A segmentação se baseia nos valores de UH para distinguir pulmões de outras estruturas.

Inicialização da Curva

Definição das regiões de interesse:

- A imagem é dividida em regiões esquerda e direita, onde os pulmões estão localizados;
- A curva inicial é posicionada automaticamente com base na distribuição de UH.

Características da inicialização:

- Utiliza uma grade de pontos organizados em um polígono regular;
- O centro da curva inicial é determinado pelos valores de UH entre -1000 e -500 (áreas aeradas do pulmão).

Cálculo da Energia Externa e Interna

Energia Externa (Ecrisp):

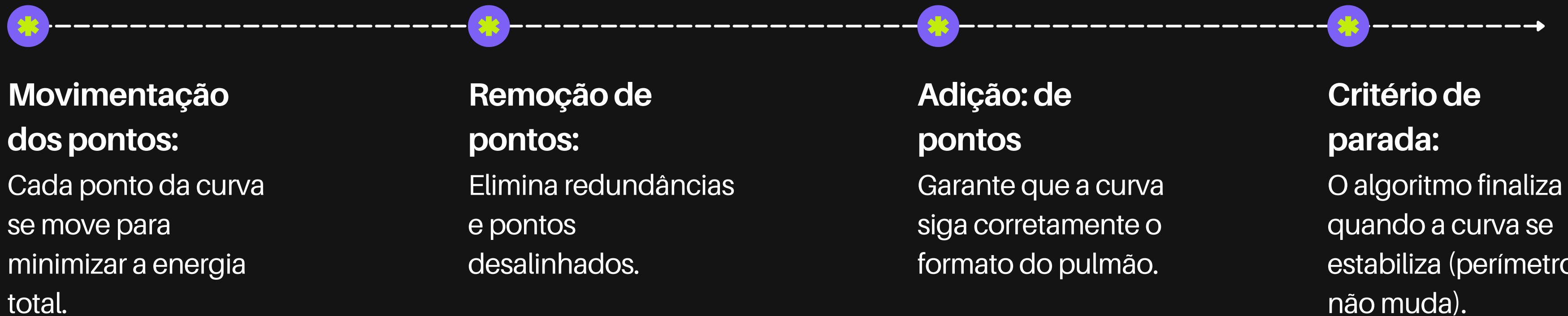
- Determinada pelo Operador Sobel, que detecta bordas pulmonares;
- Se um pixel tem baixa densidade (~ar) e não é borda, recebe energia externa menor.

Energia Interna (Eintadap):

- Mantém a curva suave e contínua, evitando distorções;
- Composta por:
 - Força de Continuidade – mantém pontos igualmente espaçados;
 - Força Balão Adaptativa – ajusta a curva para capturar a forma correta do pulmão.

Processo Iterativo de Ajuste da Curva

- Evolução da curva até a segmentação final:



Vantagem: Adaptação dinâmica à estrutura pulmonar mesmo em imagens ruidosas.

Implementação

Leitura da Imagem e Pré-Processamento

- Leitura do arquivo DICOM, arquivo *carregar.py*, usando biblioteca *pydicom*.

```
# carregar.py

def carregar_imagem(input_path: str) -> np.ndarray:
    """
    Carrega imagem do tipo DICOM e retorna a imagem em Hounsfield Units (HU).
    """
    pass
```

Definição dos parâmetros

```
class MCACrisp:  
    def __init__(  
        self,  
        image_path,  
        y_min,  
        y_max,  
        x_min,  
        x_max,  
        quantidade_pixels=30,  
        raio=30,  
        w_cont=0.6,  
        w_adapt=0.1,  
        d_max=10.0,  
        area_de_busca=9,  
        alpha=20,  
        early_stop=0.2,  
        ~~~~~):
```

Classificação das áreas da imagem

- Arquivo *classificao.py*;
- Função que classifica cada pixel com base nos intervalos de UH, formando uma matriz de densidades 9x9 a partir dos vizinhos de -4 a 4.

```
def calcula_ocorrencias_classes(src: MatLike) -> np.ndarray:  
    """  
    Calcula a quantidade de ocorrências de cada classe para cada pixel usando  
    operações vetorializadas.  
    """  
    def probabilidade_classes(ocorrencias: MatLike) -> np.ndarray:  
        """  
        Calcula as probabilidades das classes usando operações vetorializadas.  
        Recebe como entrada as ocorrencias calculada na função  
        calcula_ocorrencia_classes.  
        """
```

Cálculo da energia externa

- Arquivo *energia.py*;
- Aplicação do operador Sobel para cálculo da energia externa em cada pixel. Se estiver dentro dos limites definidos empiricamente, a energia é zero. Esse valor auxilia na definição de onde a curva deve “grudar” na imagem, ressaltando os contornos do pulmão.

```
def energia_externa(
    imagem: np.ndarray,
    probabilidade: np.ndarray,
    probabilidade3: float = 0.2,
    probabilidade4: float = 0.15,
) -> np.ndarray:
    """
        Calcula a energia externa da imagem aplicando o operador Sobel para obter
        a magnitude do gradiente. Em seguida, utiliza os limiares (probabilidade3 e
        probabilidade4) para criar uma máscara que define onde a energia será
        efetivamente
        considerada (definindo-a como zero fora desses limiares).
    """
```

Cálculo da energia externa

```
def energia_interna_adaptativa(  
    curva: np.ndarray, indice: int, w_adapt: float = 0.1, w_cont: float = 0.6  
) -> np.floating:  
    """
```

Calcula a energia interna adaptativa para um ponto da curva, combinando a força adaptativa (força_adaptativa) e a força de continuidade (força_continuidade), cada uma ponderada por seus respectivos pesos (w_adapt e w_cont).

```
    """
```

```
    def energia_total(  
        curva: np.ndarray,  
        indice: int,  
        energia_crisp: np.ndarray,  
        w_adapt: float = 0.1,  
        w_cont: float = 0.6,  
    ) -> np.floating:  
        """
```

Calcula a energia total de um ponto da curva como a soma da energia interna adaptativa e da energia externa (obtida a partir da matriz energia_crisp na posição correspondente).

```
        """
```

Cálculo da energia externa

```
def minimiza_energia(  
    curva: np.ndarray,  
    indice: int,  
    energia_crisp: np.ndarray,  
    area_de_busca: int = 9,  
    w_cont=0.6,  
    w_adapt=0.1,  
) -> np.ndarray:  
    """
```

Avalia todos os pontos candidatos dentro de uma área de busca (por exemplo, 9x9) ao redor de um ponto da curva. Para cada candidato, calcula a energia total (interna + externa) e seleciona o ponto que optimiza (minimiza essa energia, retornando o deslocamento que melhor se adapta ao contorno).

Inicialização da Curva

- Arquivo *curva.py*;
- Calcula o ponto inicial dentro dos limites preestabelecidos para as duas regiões do pulmão (esquerda e direita) e inicializa a curva dentro do raio com a quantidade de pontos dada. Ambas as informações são passadas como parâmetro.
- Dada uma imagem 512x512, a região esquerda está no intervalo de pixels [0, 256] e a direita [256, 512] no eixo das abscissas e ambas possuem [180, 360] no eixo das ordenadas. Todos esses são passados como parâmetros: *y_min*, *y_max*, *x_min* e *x_max*.

Inicialização da Curva

```
def crisp_inicial(  
    imagem: np.ndarray, lim_infY: int, lim_supY: int, lim_infX: int,  
    lim_supX: int  
) -> np.ndarray:  
    """
```

Recebe uma imagem em formato de array NumPy e recorta a região definida pelos limites superiores e inferiores (tanto em X quanto em Y). A função identifica a posição onde há maior concentração de pixels dentro da faixa de intensidade [-1000, -500] e retorna as coordenadas correspondentes.

```
    """
```

```
    def inicializa_curva(  
        ponto: np.ndarray,  
        raio: int = 30,  
        quantidade_pixels: int = 30,  
) -> np.ndarray:  
        """
```

Recebe o ponto inicial do eixo x e y calculado pela função `crisp_inicial`, calcula a curva e retorna a duas listas de pontos que representam a curva.

```
        """
```

Remoção de pontos

- Arquivo *curva.py*;
- Remove pontos repetidos da curva ou que não estão dentro da angulação mínima (passada como parâmetro - alpha) permitida entre vizinhos.

```
#curva.py

def remover_pontos(curva: np.ndarray, alpha: float = 20) -> np.ndarray:
    """
    Recebe pontos da curva e um ângulo mínimo para remover pontos da curva.
    """
```

Adição de pontos

- Arquivo *curva.py*;
- Adiciona pontos entre pontos vizinhos que estão a uma distância maior que uma mínima preestabelecida (passada como parâmetro - *d_max*)

```
#curva.py
```

```
def adicionar_pontos(curva, imagem, d_max):
```

```
    """
```

Recebe pontos da curva, a imagem de pulmão em UH e a distância mínima para adicionar pontos na curva.

```
    """
```

Ciclo Iterativo

- Arquivo *contorno_ativo.py*;
- Itera sobre a curva de segmentação e a atualiza até que ocorra a convergência ou que se atinja o número máximo de iterações.

Resumo do Fluxo do Algoritmo

Passos principais:

1. Carregamento da imagem DICOM;
2. Identificação das regiões pulmonares;
3. Inicialização da curva;
4. Cálculo das energias externa e interna;
5. Processo iterativo: ajuste, remoção e adição de pontos;
6. Parada quando a curva estabiliza;
7. Geração da máscara segmentada.

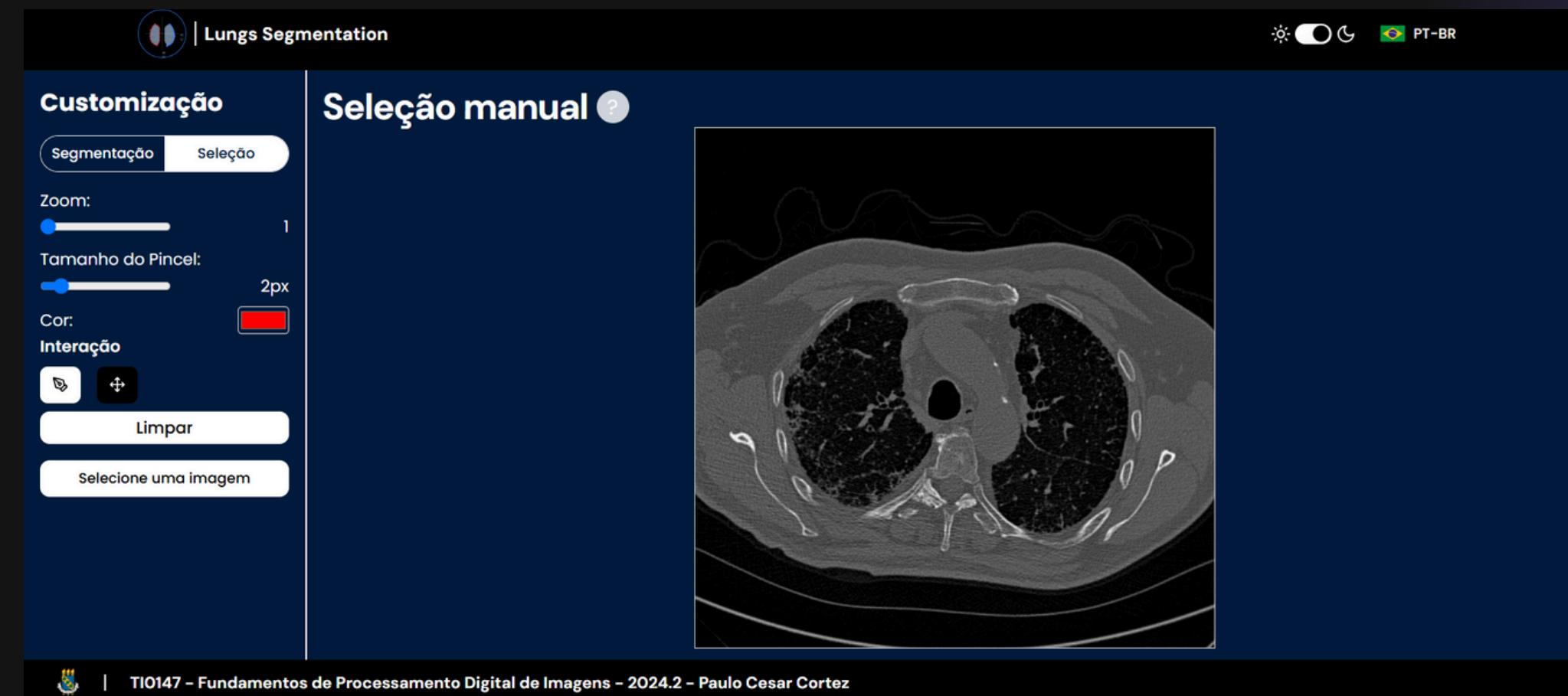
Interface

- Página Inicial:



Interface

- Segmentação Manual:

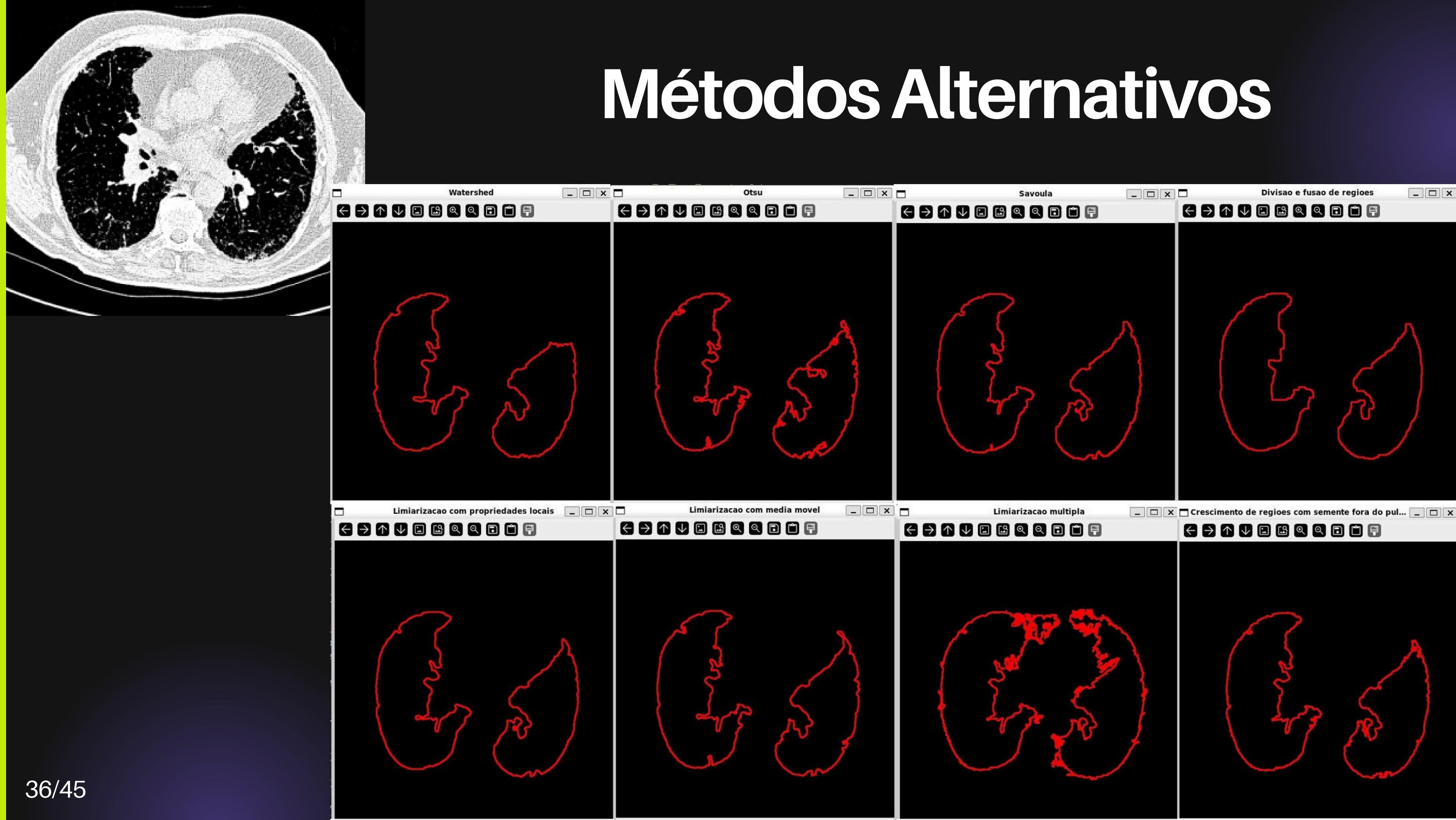


- Segmentação Automática:



Resultados

Métodos Alternativos



Algoritmo Crisp

- **Divisão da Imagem** - Segmentação ocorre em duas partes:
 - ◆ Esquerda (0 a 256 pixels);
 - ◆ Direita (256 a 512 pixels).
- **Parâmetros Principais:**
 - ◆ Área inicial: 60 pixels de raio;
 - ◆ Peso do contorno: 0.6;
 - ◆ Peso de adaptação: 0.3;
 - ◆ Área de busca: 9 pixels;
 - ◆ Máxima expansão: 6 pixels;
 - ◆ Precisão e parada: $\alpha = 20$, $\text{early_stop} = 0.001$.



Conclusão

Resumo das Equipes

- Back-end
- Front-end
- Documentação
- Gerência

Back-end

1. Implementação do Algoritmo de Contornos Ativos
Crisp 2D para segmentação pulmonar;
2. Desenvolvimento da API de integração;
3. Aplicação de métodos alternativos de segmentação;
4. Teste e validação do algoritmo implementado.

Front-end

1. Criação de protótipo de UI;
2. Elaboração da interface;
3. Desenvolvimento da interface funcional;
4. Integração da interface com API.

Documentação

1. Definição do escopo do projeto;
2. Criação do manual de uso;
3. Elaboração da documentação final;
4. Execução e criação do relatório de testes;
5. Desenvolvimento da apresentação em slides.

Gerência

1. Controle e versionamento com GitHub;
2. Definição e acompanhamento do cronograma;
3. Gestão de riscos e mitigação de problemas;
4. Revisões de códigos/documentos e alinhamento entre as equipes.

Comentários Finais

- Os métodos clássicos performaram relativamente bem para imagens específicas;
- Sem a utilização de aprendizado de máquina para ajuste fino dos parâmetros do método principal ele não consegue performar tão bem quanto esperado.

Melhorias Futuras

- Utilizar métodos de aprendizado de máquina para refinar o algoritmo a fim de melhorar sua performance geral;
- Testar e avaliar a performance do algoritmo em pulmões não saudáveis;
- Testar e avaliar a performance o algoritmo em uma base de imagens mais ampla.

Obrigado!