Implementação do cálculo de esqueleto de objetos em imagens binárias

Carlos Eduardo Fontaneli, RA - 769949

26 de agosto de 2022

Resumo

O presente trabalho tem como objetivo implementar e analisar a técnica de cálculo de esqueleto de objeto em imagens binárias. Ademais, utilizou-se a linguagem python para a implementação dos códigos necessários.

Palavras-chaves: cálculo, esqueleto, objeto, python.

Introdução

Este documento conta com a implementação da técnica de cálculo de esqueleto de objeto em imagens binárias. Tal procedimento é a transformação de um objeto binário em uma sequência de pixels com largura 1, a qual representará a forma do objeto original. Dessa forma, é possível extrair uma descrição simples e eficiente de formas, além disso esse procedimento permite a identificação de posição de pessoas ou seres em imagens.

1 Objetivos

1.1 Cálculo do esqueleto em imagens binárias

Neste presente trabalho, buscou-se analisar o cálculo de esqueleto de objetos em imagens binárias. Para tanto, implementou-se uma função que calcula o esqueleto de um objeto(com valores de cores diferentes de zero) em uma imagem binária de fundo preto.

1.2 Aplicação em diferentes formas

Após a implementação do cálculo de esqueleto, buscou-se aplicá-lo em diferentes formas para análise de sua eficácia e acurácia. Portanto, selecionou-se 3 formas diferentes para aplicação do cálculo de esqueleto.

2 Metodologia

Em busca de implementar o cálculo de esqueleto há diversas abordagens possíveis, neste presente trabalho utilizou-se a abordagem de implementação utilizando morfologia.

Seja p_1 um dado pixel da imagem e sua vizinhança de p_n pixels dada da seguinte forma:

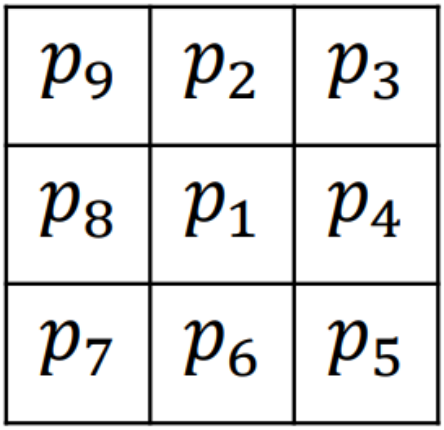

Figura 1 – Vizinhança de uma dado pixel

Dessa forma, para cada pixel do objeto(pixel da imagem com valor diferente de 0), o objetivo é demarcar para remoção os pixels que satisfazem as condições dos seguintes passos:

Passo 1:
(a) $2 <= N(p_1) <= 6$, onde $N(p_1)$ é o número de vizinhos não nulos de p_1;
(b) $T(p_1)$, onde $T(p_1)$ é o número de transições 0-1 na sequência de pixels vizinhos;
(c) $p_2 p_4 p_6 = 0$;
(d) $p_4 p_6 p_8 = 0$.

Após a conferência das condições acima para todos os pixels do objeto e a demarcação daqueles que a atendem, remove-se tais pixels e na sequência aplica-se o segundo passo que segue:

Passo 2:
(a) $2 <= N(p_1) <= 6$, onde $N(p_1)$ é o número de vizinhos não nulos de p_1;
(b) $T(p_1)$, onde $T(p_1)$ é o número de transições 0-1 na sequência de pixels vizinhos;
(c) $p_2 p_4 p_8 = 0$;
(d) $p_2 p_6 p_8 = 0$.

Feito isso, aplica-se novamente o processo de demarcação e remoção dos pixels que satisfazem as condições. Tais passos são repetidos até que não haja nenhum pixel do objeto que atenda nenhuma das condições.

2.1 Implementação do cálculo de esqueleto

Para a implementação do cálculo de esqueleto de objeto em imagens binárias foi necessário criar uma função que receberá uma imagem binária que conterá um objeto(com pixels de valores diferentes de 0) e fundo preto.

Dessa forma, em busca de aplicar os passos do cálculo faz-se necessário aplicar um padding de 1 pixel na imagem original para tratar os pixels da borda da imagem original. Na sequência, começa-se o laço de aplicação dos 2 passos sobre cada pixel da imagem.

Assim sendo, começa-se a iteração sobre os pixels da imagem, confere-se se o pixel selecionado pertence ao objeto. Depois, separa-se os vizinhos daquele pixel para análise posteriore dos passos. Com os vizinhos separados, começa-se a conferência de cada condição do passo em questão.

Após tais processos, demarca-se os pixels que atendem as condições e, após a iteração completa do passo em questão, remove-se tais pixels. Estes procedimentos são aplicados para ambos os passos em sequência e, por fim, quando não há mais demarcação de pixels a serem removidos o laço principal é encerrado.

Abaixo segue o código em Python resultante de tal processo:

```
def skeleton_shape(img):
  num_rows, num_cols = img.shape
  img_padded = np.pad(img, ((1, 1), (1, 1)),
                              mode='constant')
  change = 1
  # index dos vizinhos
  nei_list = [(-1, 0), (-1, 1), (0, 1), (1, 1),
              (1, 0), (1, -1), (0, -1), (-1,-1)]
  while(change):
    change = 0
    # PASSO 1
    removed_pixels_ind = []
    for row in range(1, num_rows+1):
      for col in range(1, num_cols+1):
        # conferindo se o pixel pertence ao objeto
        if img_padded[row][col] != 0:

          # separando os vizinhos
          neighbours = []
          non_null_neighbours = 0
          for nei_index in nei_list:
            # contagem de vizinhos n o nulos
            if img_padded[row + nei_index[0]][col + nei_index[1]] != 0:
              non_null_neighbours += 1
            neighbours.append(img_padded[row + nei_index[0]]
                                        [col + nei_index[1]])

          # condicoes do PASSO 1
          # (a)
          if 2 <= non_null_neighbours and non_null_neighbours <= 6:
```

```
            # (b)
            T = 0
            for j in range(len(neighbours) - 1):
              if neighbours[j]  == 0 and neighbours[j+1] != 0:
                T += 1
            if neighbours[7]  == 0 and neighbours[0] != 0:
              T += 1
            if T == 1:

              # (c)
              if neighbours[0] == 0 or \
                 neighbours[2] == 0 or \
                 neighbours[4] == 0:

                # (d)
                if neighbours[2] == 0 or \
                   neighbours[4] == 0 or \
                   neighbours[6] == 0:

                  removed_pixels_ind.append((row, col))
                  change = 1 # alterando o criterio de parada

# REMOVENDO OS PIXELS
for pixel in removed_pixels_ind:
  img_padded[pixel[0]][pixel[1]] = 0

# PASSO 2 (mesmo procedimento do passo 1 at  a condicao (b))
removed_pixels_ind = []
for row in range(1, num_rows+1):
  for col in range(1, num_cols+1):
    if img_padded[row][col] != 0:

      neighbours = []
      non_null_neighbours = 0
      for nei_index in nei_list:
        if img_padded[row + nei_index[0]][col + nei_index[1]] != 0:
          non_null_neighbours += 1
        neighbours.append(img_padded[row + nei_index[0]]
                                    [col + nei_index[1]])

    # conferindo as condicoes
      if 2 <= non_null_neighbours and non_null_neighbours <= 6:

        T = 0
        for j in range(len(neighbours) - 1):
          if neighbours[j]  == 0 and neighbours[j+1] != 0:
            T += 1
        if neighbours[7]  == 0 and neighbours[0] != 0:
          T += 1
```

```
            if T == 1:

              # (c)
              if neighbours[0] == 0 or \
                 neighbours[2] == 0 or \
                 neighbours[6] == 0:

                # (d)
                if neighbours[0] == 0 or \
                   neighbours[4] == 0 or \
                   neighbours[6] == 0:
                  removed_pixels_ind.append((row, col))
                  change = 1

    for pixel in removed_pixels_ind:
      img_padded[pixel[0]][pixel[1]] = 0

  # Retornando a imagem resultante nas proporcoes da imagem original
  return img_padded[1:-1][1:-1]
```

3 Resultados

3.1 Imagens binárias utilizadas

Para a aplicação do cálculo de esqueleto de objetos em imagens binárias foram utilizadas as seguintes imagens:

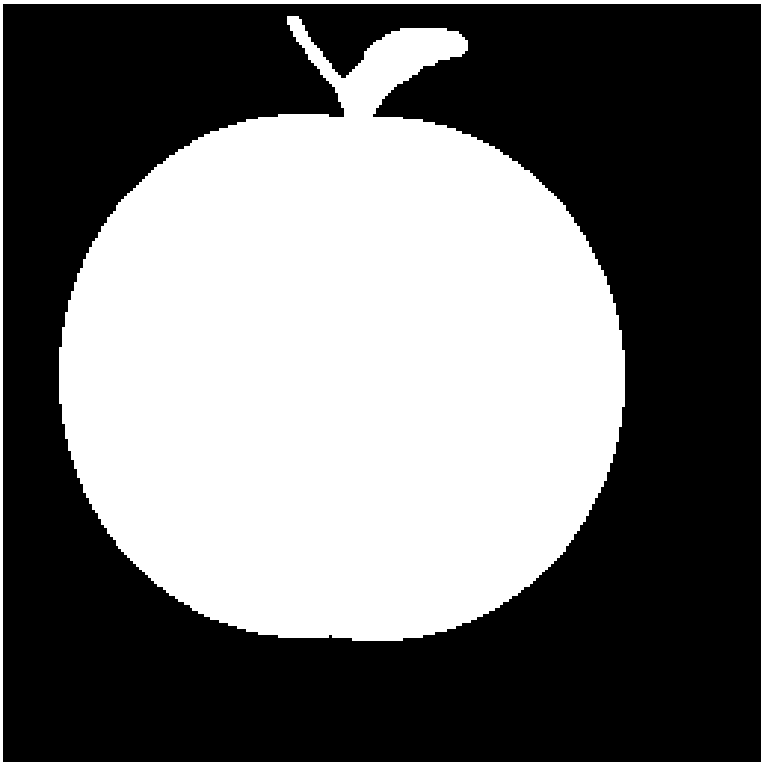

Figura 2 – Imagem binária de uma forma de maçã

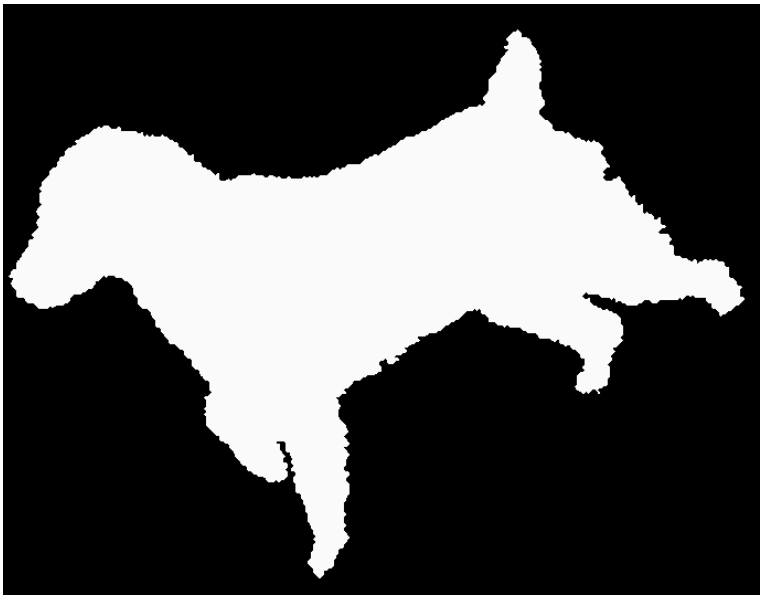

Figura 3 – Imagem binária de uma forma de cachorro

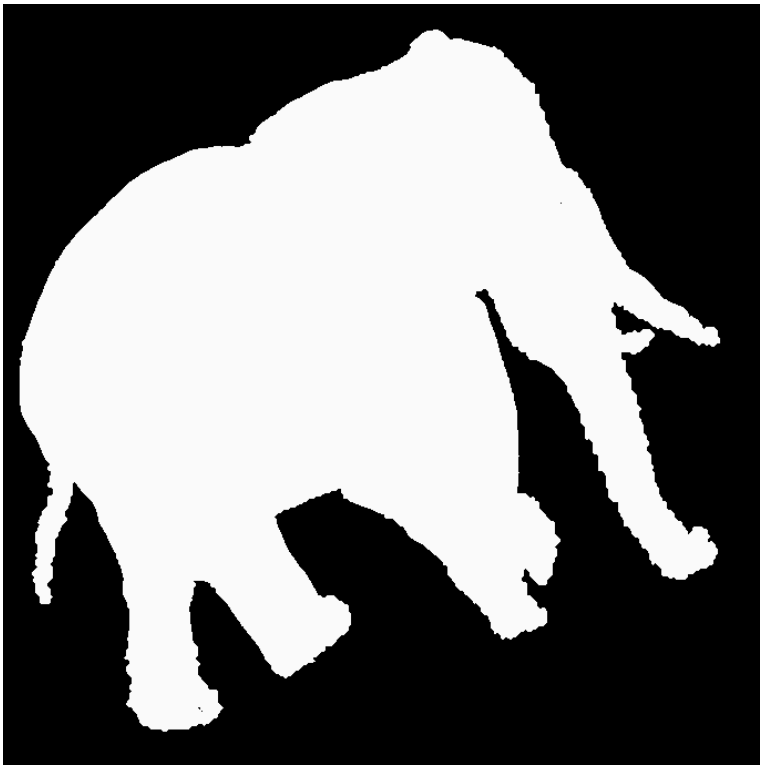

Figura 4 – Imagem binária de uma forma de elefante

3.2 Esqueletos obtidos

Após a aplicação do cálculo de esqueleto nas imagens binárias, obteve-se as seguintes imagens resultantes:

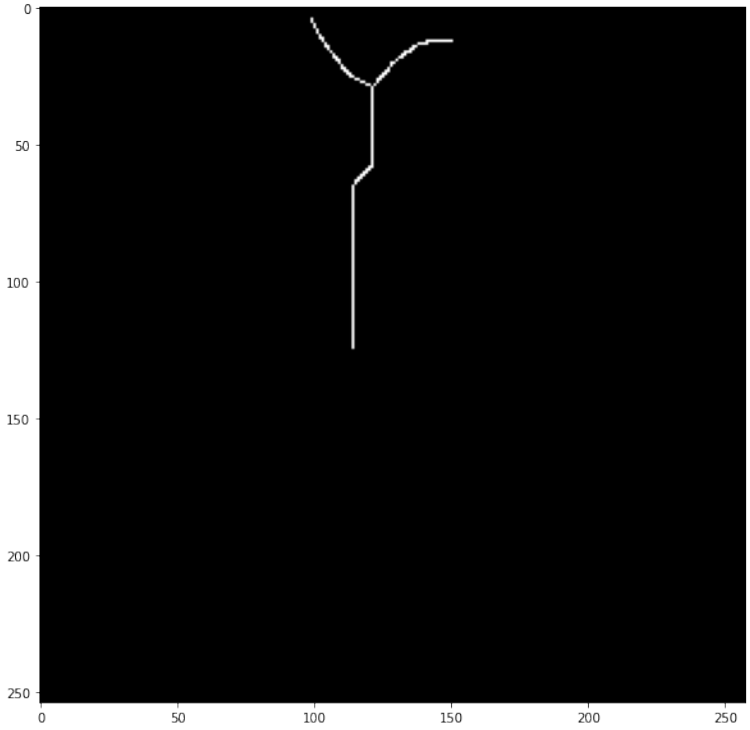

Figura 5 – Esqueleto resultante de uma forma de maçã

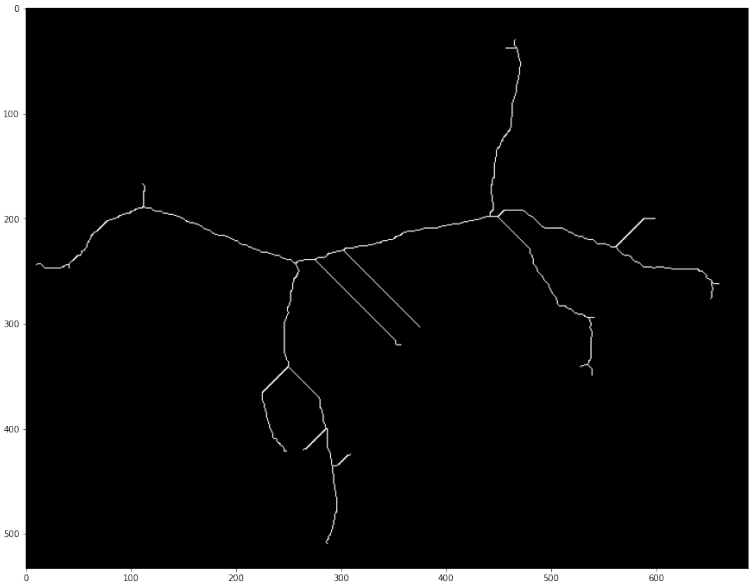

Figura 6 – Esqueleto resultante de uma forma de cachorro

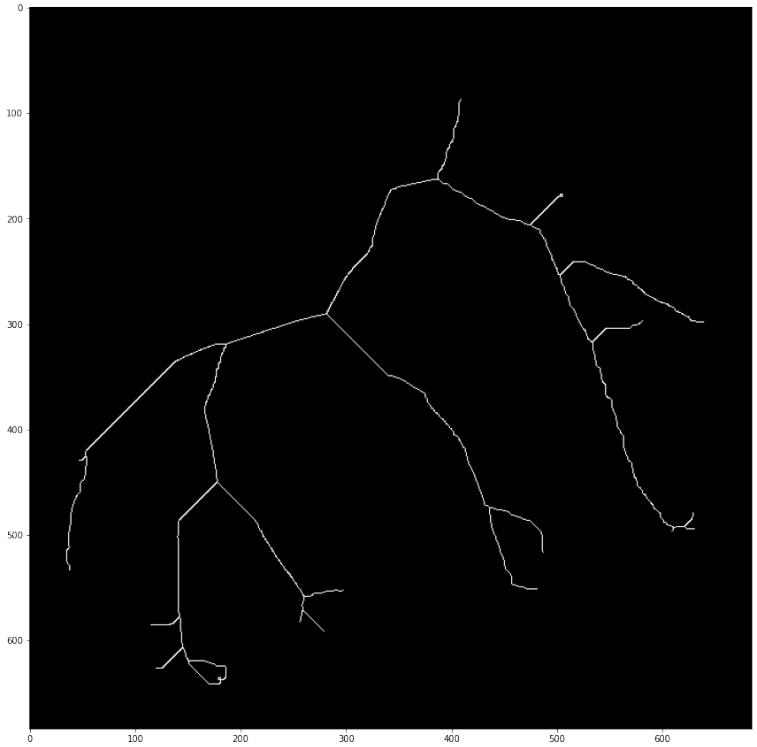

Figura 7 – Esqueleto resultante de uma forma de elefante

Considerações finais

Para o esqueleto resultante da forma de maçã é possível observar que o esqueleto representa de forma satisfatória o objeto original. É possível perceber que o esqueleto começa da porção superior do objeto e demarca corretamente o corpo da maçã e as pontas do caule. Ademais, o esqueleto demarca também corretamente a curvatura do caule e suas pontas.

Para o esqueleto da forma de cachorro é possível perceber a demarcação do corpo do cachorro. Dessa forma, é vísivel as demarcações das pernas, do rabo e da cabeça do cachorro. Entretanto, é evidente as diversas demarcações incorretas sobre o esqueleto, onde há ramificações pequenas do esqueleto que não representam a forma do cachorro. Este comportamento é devido a alta sensibilidade a pequenas variações na forma do objeto do algoritmo baseado em morfologia. Portanto, como a imagem do cachorro possui muitas variações pequenas em seu contorno, o esqueleto resultante acaba por encorporando essas pequenas variações.

O esqueleto resultante da forma do elefante demonstrou um resultado similar ao do cachorro, é possível ver a forma do elefante como um todo, seu rabo, suas pernas, sua trompa e até o chifres da trompa do elefante são perceptíveis no esqueleto. Contudo, o mesmo problema das pequenas variações no contorno ocorra aqui também.

De forma geral, pode-se afirmar que o algoritmo baseado em morfologia apresenta um resultado satisfatória e se mostrou capaz de extrair um esqueleto que resume bem a forma da imagem original. Contudo,