Aplicação de operadores morfológicos

Felipe Diniz Tomás - RA:110752

I. Introdução

A morfologia matemática é uma teoria algébrica que estuda a decomposição de operadores entre reticulados completos em termos de operadores e operações elementares, sendo elas de erosão, dilatação, união, intersecção e complemento. Este campo é uma vertente do processamento não-linear de imagens digitais que analisa e processa estruturas geométricas e topológicas.

Quando tratamos de morfologia matemática, em essência consiste em extrair informações relativas à geometria e à topologia de um conjunto arbitrário em uma imagem, esse processamento é realizado em função de um conjunto, conhecido como elemento estruturante, que é definido, conhecido e comparado ao conjunto arbitrário da imagem.

Inspeciona-se a imagem com o elemento estruturante e quantifica-se o modo com que este elemento se ajusta ou não na imagem, por consequência o tamanho do elemento estruturante influenciam nesta análise.

II. Metodologia

As execuções realizadas foram feitas no seguinte ambiente computacional:

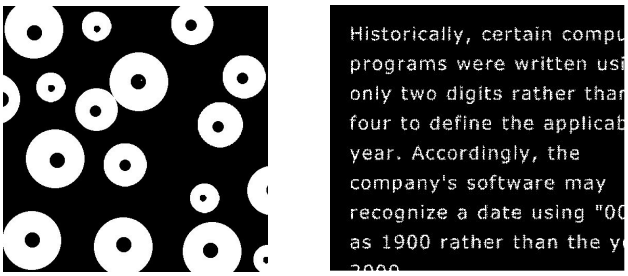
- A IDE utilizada foi o Spyder 5.1.5[1]
- Sistema operacional Windowns 10.
- Python versão 3.10[2]
- Numpy versão 1.21.3 [3].
- Matplotlib versão 3.4.3 [4].
- OpenCv versão 4.5.4-dev [5].

A. Casos de testes

Para testar a implementação dos operadores morfológicos, foi utilizado a imagem disponível no capítulo 9 do livro Digital Image Processing [1], que está sendo usado como referência para as aulas.

Para o operador morfológico de preenchimento de buraco, foi usado a imagem (a) e para operador de limpeza de borda foi usado a imagem (b).

[1]https://www.spyder-ide.org/
[2]https://www.python.org/
[3]https://numpy.org/
[4]https://matplotlib.org/
[5]https://opencv.org/



(a) Fig0907.tif (b) Fig0916.tif

Fig. 1: Casos de testes

Considerando as imagens testes, o resultado esperado seria a imagem (a) com os buracos pretos internos aos círculos brancos fechados, e a imagem (b) com os caracteres que ultrapassem as bordas removidos.

III. Código-fonte

O código-fonte, mesmo que simples, foi aplicado os conceitos vistos em salas de aula através dos exemplos teóricos, além de seguir as dicas e exigências da especificação.

```
def select_img():
    print("============================================")
    print("Digite o número correspondente a imagem que deseja carregar:\n")
    filenames = next(walk(os.path.dirname(__file__) + "/"), (None, None, []))[2]
    files = [ file for file in filenames if file.endswith((".jpg", ".JPG", ".PNG", ".png", ".tif", ".tiff", ".TIF")) ]

    if len(files) == 0:
        print("Não existe arquivos de imagem no diretório raiz.")
        sys.exit()

    for i in range(0, len(files)):
        print('[{}] {}'.format(i, files[i]))
    index = input()

    while not index.isdigit() or int(index) >= len(files) or int(index) < 0:
        print("\nEsta opção não existe, digite novamente.")
        index = input()

    imagem = cv2.imread(os.path.dirname(__file__) + "/" + files[int(index)], cv2.IMREAD_GRAYSCALE)

    print("\nSelecione o operador que deseja utilizar:")
    print("\n[1] Fechamento de buracos.")
    print("[2] Limpeza de borda.")
    operator = input()

    while(not operator.isdigit() or (int(operator) != 2 and int(operator) != 1)):
        print("\nValor inválido! Digite novamente:")
        operator = input()

    return imagem, int(operator), files[int(index)]
```

Fig. 2: Função Seleção de imagem e operador.

Na figura 2, podemos ver a função responsável por ler as imagens disponíveis no diretório raiz ao código-fonte, podendo ser nos formatos jpg, png e tif, selecionando qual será aberta. Além disso nela é escolhido o operador que será aplicado.

```python
# Aplica o preenchimento de buracos

def filling_holes(image):
    seed = np.copy(image)
    seed[1:-1, 1:-1] = image.max()
    mask = image
    filled = reconstruction(seed, mask, method='erosion')
    return filled


# Aplica a limpeza de borda

def border_clearing(image):
    seed = np.copy(image)
    seed[1:-1, 1:-1] = image.min()
    mask = image
    filled = reconstruction(seed, mask, method='dilation')
    result = image - filled
    return result
```

Fig. 3: Funções dos operadores morfológicos.

A figura três mostra a implementação da função de preenchimento e remoção de bordas, que irá aplicar a reconstrução de acordo com o operador.

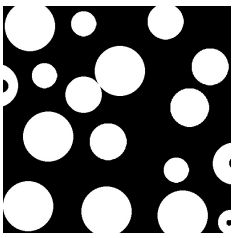
Por fim, a main será responsável por mostrar e salvar as imagens com a biblioteca Opencv, como pode ser visto na figura 4.

```python
def main():
    imagem, operator, nome_arq = select_img()

    if operator == 1:
        img_result = filling_holes(imagem)
        msg = "Imagem preenchida"
        msg_save = "preechida"
        name_save = "buracos"
    else:
        img_result = border_clearing(imagem)
        msg = "Imagem com bordas limpas"
        msg_save = "sem bordas"
        name_save = "bordas"

    print("\nFeche as janelas para continuar...")
    cv2.imshow('Imagem binaria', imagem)
    cv2.imshow(msg, img_result)
    cv2.waitKey(0)
    print("\nGostaria de salvar a imagem " + msg_save + "?\n[1] Sim\n[2] Não")
    op = input()
    while op != '1' or op != '2':
        if op == '2' or op == '1': break
        print("\nEsta opção não existe, tente novamente.")
        op = input()

    #salvando a imagem

    if op == '1':
        aux = nome_arq.split(".")
        cv2.imwrite(os.path.dirname(__file__) + "/" + str(name_save) +"_"+ aux[0] +'.jpg', img_result)
        print("\nSalvo em:\n" + os.path.dirname(__file__))

    print("\nPrograma encerrado.")
if __name__ == "__main__":
    main()
```

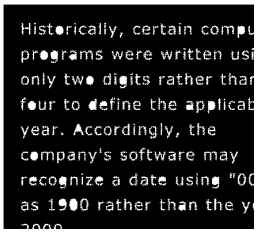
Fig. 4: Função Main.

IV. Imagens resultantes

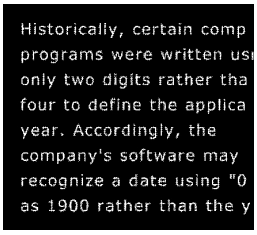
Os resultados foram gerados aplicando os operadores implementados, inclusive sendo possível ver resultados em ambas as imagens de acordo com o operador escolhido. As figura 5 a seguir mostram essa experimentação.



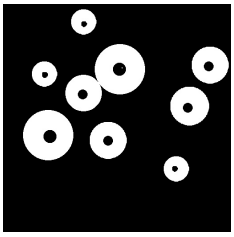
(a) Preenchimento de buraco na fig0916.



(b) Preenchimento de buraco na fig0907.



(c) Limpeza de borda na fig0907.



(d) Limpeza de borda na fig0916.

Fig. 5: Resultados do Caso de teste.

Podemos observar que os buracos foram preenchidos tanto na imagem (a) como na imagem (b), onde existem buracos em alguns caracteres do alfabeto. Já a retirada de elementos que tocam a borda performou como esperado, removendo cada elemento que toca os cantos da imagem, visualmente a imagem (d) deixa mais evidente esse processo.

Portanto, mostra-se útil a aplicação dos operadores morfológicos em imagens binárias. Além disso, graças as bibliotecas disponíveis em python, fica simples implementar tal técnica de processamento digital de imagens.

References

[1] R. C. Gonzalez and R. E. Woods, *Digital image processing*. Upper Saddle River, N.J.: Prentice Hall, 2008.