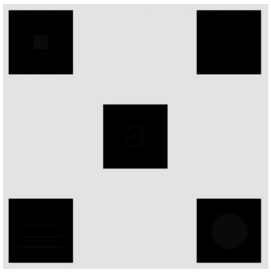
Normalização local de contraste com o uso de filtros de ordem

Felipe Diniz Tomás - RA:110752

I. Introdução

Os histogramas são a base para várias técnicas de processamento de domínio espacial. O processamento de histograma em imagens é usado para aprimoramento de imagem, modificando seu histograma para obter uma melhor aparência visual de um imagem. Os métodos usuais incluem equalização do histograma, e a normalização do histograma, a qual será implementada.

Estes métodos geralmente aumentam o contraste ao distribuir uniformemente intensidades no histograma. Funcionam, centralizando a região de vizinhança em um pixel x na imagem de entrada, de onde os valores de intensidade serão extraídos para a aplicação do método. O resultado é aplicado na posição x da imagem de saída. Esta região de vizinhança é deslocada pixel a pixel para que todos os pixels da imagem de entrada sejam processados.



(a) test.tif

Fig. 1: Casos de testes

Esta imagem possui características interessantes, já que sem contraste esconde símbolos e letras que poderão ser vistas aplicando a normalização de contraste local. Portanto terá clara diferença no uso de diferentes tamanhos de máscara, além da real percepção de mudanças quanto aos seus detalhes e elementos.

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 da normalização de contraste local com o uso de filtro de ordem, foi utilizado a imagem disponível no capítulo 3 do livro Digital Image Processing [1], que está sendo usado como referência para as aulas.

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

III. Código-fonte

O código-fonte, mesmo que simples, foi aplicado os conceitos vistos em salas de aula através dos exemplos práticos, 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("\nDigite o tamanho da máscara que deseja utilizar:")
    mask_size = input()

    while(not mask_size.isdigit()):
        print("\nValor inválido! Digite novamente:")
        mask_size = input()

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

Fig. 2: Função Seleção de imagem e tamanho da máscara.

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 tamanho da máscara definido pelo usuário.

```
# Aplica a normalização

def normalizacao(img, mask_size):
    size = mask_size
    max = maximum(img, np.ones((size, size)))
    min = minimum(img, np.ones((size, size)))
    a = img - min
    b = max - min
    #np.copyto(a, b, where=0)
    #np.place(a, b == 0, 0.0000001)
    np.seterr(invalid ='ignore')
    result = np.true_divide(a, b) * 255
    return result
```

Fig. 3: Função que aplica a normalização.

A figura três mostra a implementação da função de normalização, que irá aplicar o método de contraste local considerando o que foi visto em sala de aula.

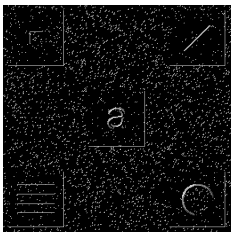
Por fim, a main será responsável por mostrar e salvar as imagens com a biblioteca Opencv, como pode ser visto na figura 4.

```
def main():
    imagem, mask_size, nome_arq = select_img()
    img_normalized = normalizacao(imagem, mask_size)
    print("\nFeche as janelas para continuar...")
    cv2.imshow('Imagem cinza', imagem)
    cv2.imshow('Imagem normalizada', img_normalized)
    cv2.waitKey(0)
    print("\nGostaria de salvar a imagem normalizada?\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(mask_size) +"_"+ aux[0] +'.jpg', img_normalized)
        print("\nSalvo em:\n" + os.path.dirname(__file__))

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

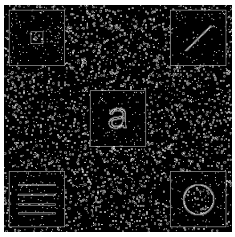
Fig. 4: Função Main.

IV. Imagens resultantes

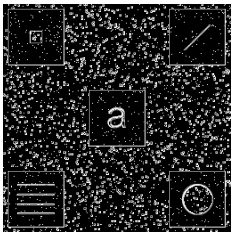
Os resultados foram gerados utilizando diferentes tamanhos de máscara, as figuras a seguir mostram essa experimentação.



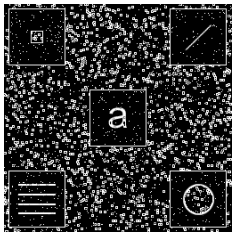
(a) Máscara com tamanho 2.



(b) Máscara com tamanho 3.



(c) Máscara com tamanho 4.



(d) Máscara com tamanho 5.

Fig. 5: Resultados do Caso de teste.

Como destacado anteriormente, o caso de teste escondia em seus quadrados pretos na imagem alguns símbolos e letras, que agora podem ser vistos graças a alteração de contraste. Percebe-se que quanto maior a máscara, mais os símbolos se evidenciam. No entanto, nota-se uma grande granulação branca, efeito causado pelo método de normalização local aplicado nesta imagem.

Portanto, mostra-se útil a aplicação da normalização com filtro de ordem em casos particulares, como desta imagem. 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.