Processamento digital de imagens

Leonardo Borck da Silveira

Luiz Felipe Cipriani Morfelle

Norton Batista Abdala

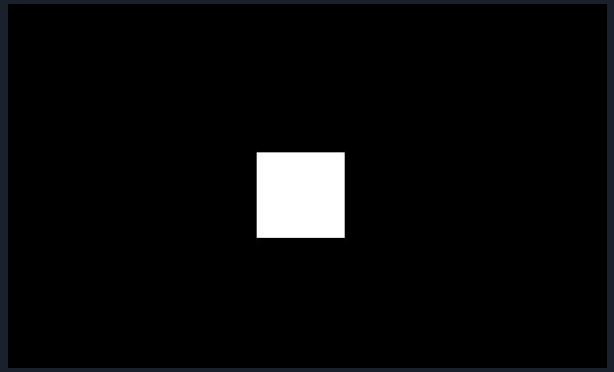
Operações aritméticas

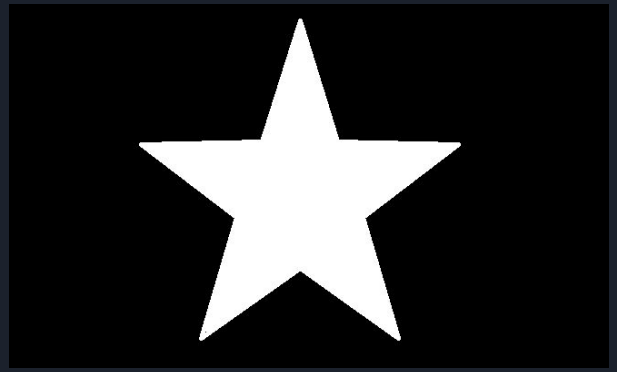
- p(x, y) = f(x, y) * g(x, y)
- d(x, y) = f(x, y) - g(x, y)
- s(x, y) = f(x, y) + g(x, y)
- v(x, y) = f(x, y) / g(x, y)

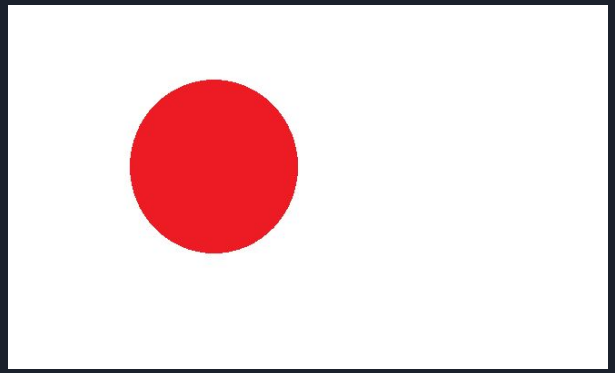
Considerações

O algoritmo utilizou 3 imagens de entrada para gerar duas imagens de saída, utilizando das operações aritméticas requisitadas

Imagens Base







Resultados

(CÍRCULO – ESTRELA) + QUADRADO

ESTRELA – (ESTRELA * (QUADRADO / CÍRCULO)

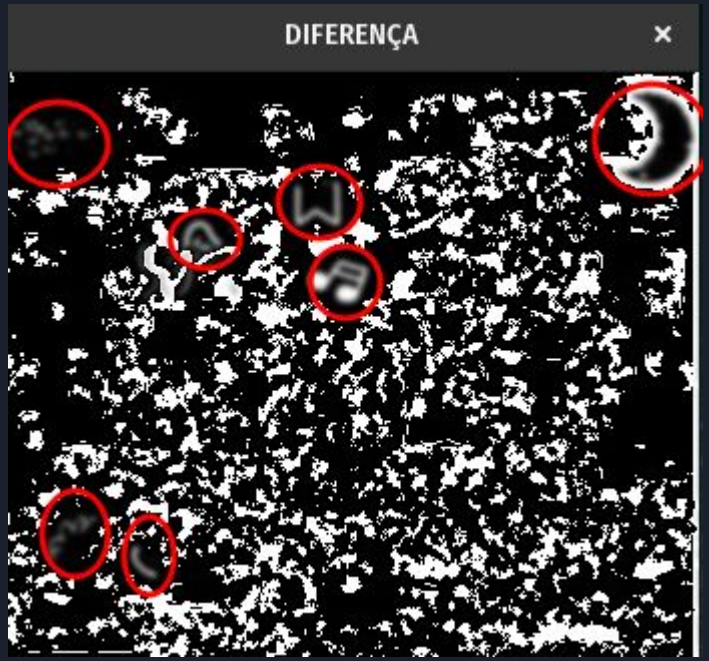
Trecho de código

```
#Loop passando em cada pixel da imagem
for i in range(image1.shape[0]):
    for j in range(image1.shape[1]):
        #Recolhendo os pixels de cada imagem
        img1 = image1[i, j]
        img2 = image2[i, j]
        img3 = image3[i, j]

        result[i, j] = (img1 - img3) + img2
```

```
#Loop passando em cada pixel da imagem
for i in range(image1.shape[0]):
    for j in range(image1.shape[1]):
        #Recolhendo os pixels de cada imagem
        img1 = image1[i, j]
        img2 = image2[i, j]
        img3 = image3[i, j]
        result[i, j] = img3 - (img3 * (img2 / img1))
```

Imagens Base - 7 Erros (Subtração)

Resultados - 7 Erros (Subtração)



Trecho de código

```
# Laço pelos pixels da Imagem
# .shape[0] são as linhas e .shape[1] as colunas
for x in range(jogo1.shape[0]):
    for y in range(jogo1.shape[1]):
        result[x, y] = (jogo2[x, y] - jogo1[x, y])
```

Isolar canais (rgb)

- imagem R - mantém o valor de R e zera G e B
- imagem G - mantém o valor de G e zera R e B
- imagem B - mantém o valor de B e zera G e R

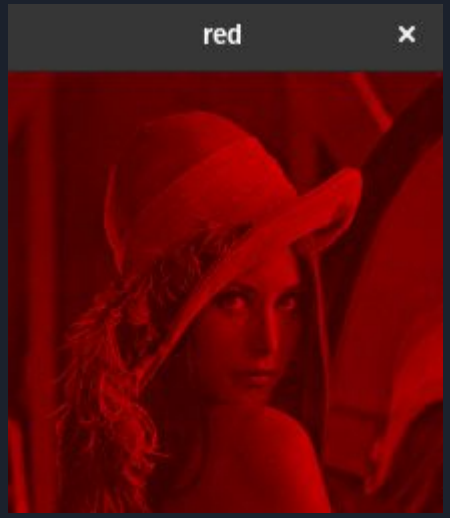
Considerações

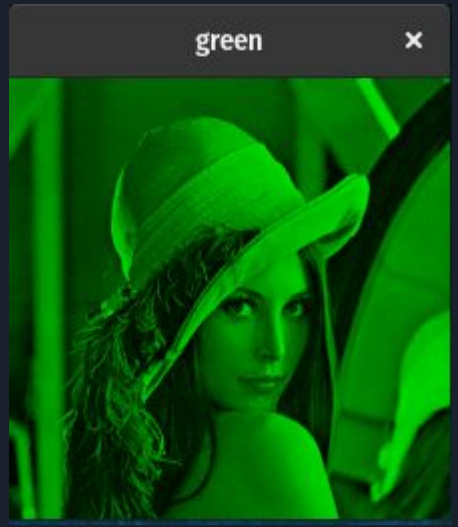
O algoritmo utilizou a imagem da Lena como entrada para gerar 3 imagens de saída, cada uma representando um canal de cor RGB

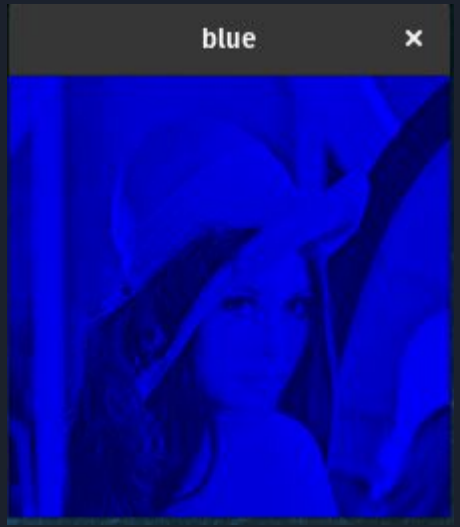
Foi relativamente simples a implementação deste algoritmo apenas necessitando zerar os 2 canais de cores para isolar o canal escolhido.

Imagem Base

Resultado







Trecho de código

```
# BLUE
for x in range(img.shape[0]):
    for y in range(img.shape[1]):
        pixelblue = imgb[x, y]
        pixelblue[1] = 0
        pixelblue[2] = 0
        imgb[x, y] = pixelblue
```

Conversão em tons de cinza

- Média aritmética
- Média ponderada

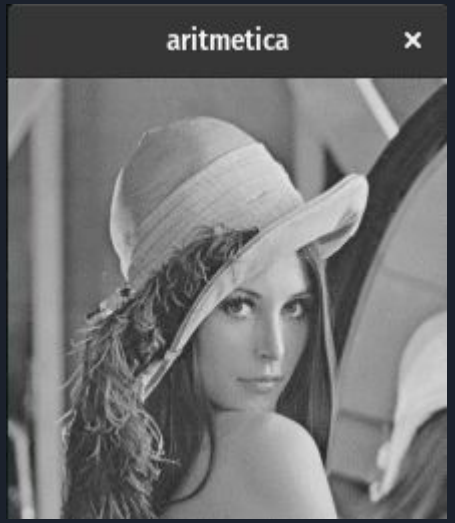
Considerações

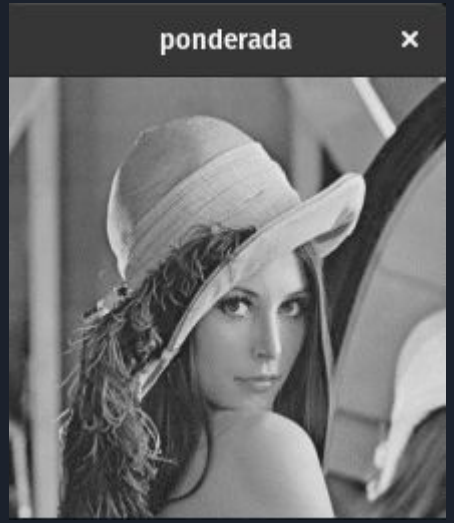
O algoritmo utilizou a imagem da Lena como entrada para gerar 2 imagens de saída, uma utilizando a média ponderada e a outra a média aritmética.

Uma dificuldade encontrada foi no algoritmo de média aritmética que apresentou um bug ao somar e dividir os canais, pois os valores de cada canal precisaram ser multiplicados por 1. (Já havíamos comentado com o professor durante a aula).

Imagem Base

Resultado





Trecho de código

```
# Ponderada B G R
for x in range(ponderada.shape[0]):  # Altura
    for y in range(ponderada.shape[1]):  # Largura
        pixel = ponderada[x, y]
        pixel2 = (0.114*pixel[0] + 0.587*pixel[1] + 0.299*pixel[2])
        ponderada[x, y] = pixel2

# Aritimetica B G R
for x in range(aritmetica.shape[0]):  # Altura
    for y in range(aritmetica.shape[1]):  # Largura
        pixel = aritmetica[x, y]
        pixel2 = (1*pixel[0] + 1*pixel[1] + 1*pixel[2])/3
        aritmetica[x, y] = pixel2
```

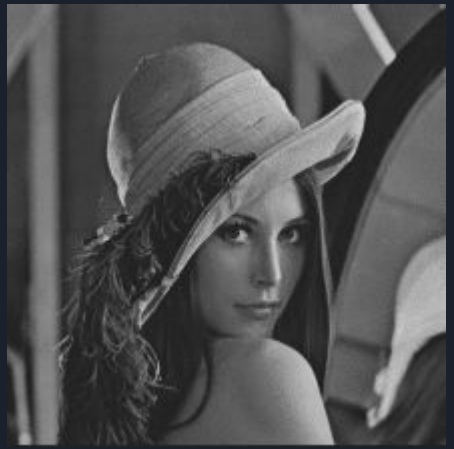
Limiarização (threshold)

Considerações

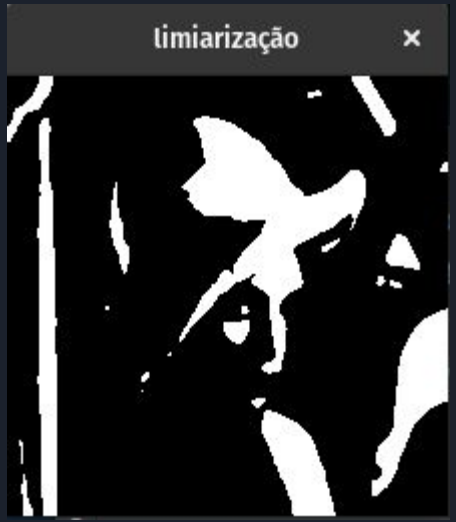
O algoritmo utilizou a imagem da Lena como entrada para gerar 1 imagem de saída.

Foi escolhido o valor de 160 como valor de corte, acima de 160 será transformado BRANCO e igual ou menor que 160 será transformado preto.

Imagem Base



Resultado



Trecho de código

```
for x in range(suave.shape[0]):
    for y in range(suave.shape[1]):
        if limiar[x, y][0] > 160:  # Tudo
            limiar[x, y] = 255  # branco
        else:
            limiar[x, y] = 0  # preto
```

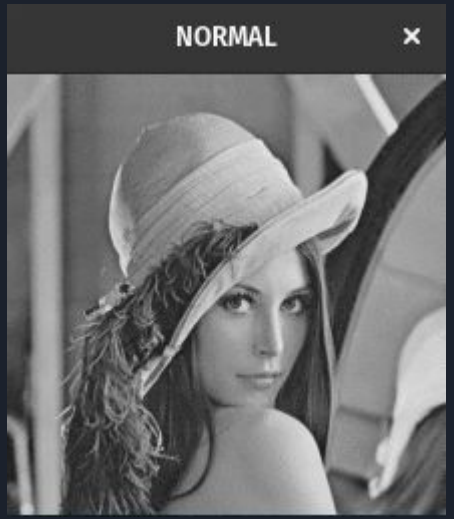
Convolução Genérica

Considerações

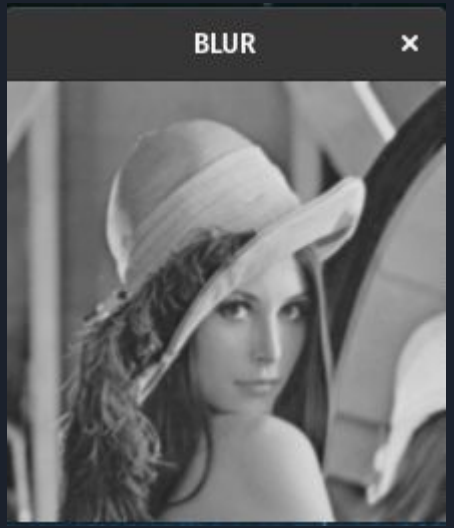
O algoritmo utilizou a imagem da Lena convertida em tons de cinza pela média ponderada como entrada para gerar 1 imagem de saída desfocada.

Máscara utilizada (0.0625, 0.125, 0.0625, 0.125, 0.25, 0.125, 0.0625, 0.125, 0.0625)

Imagem Base



Resultado



Trecho de código

```
#  FUNÇÃO DA CONVOLUÇÃO
pixel2 = ((0.0625*pixel_esquerdacima)+(0.125*pixel_cima)+(0.0625*pixel_direitacima) +
          (0.125*pixel_esquerda)+(0.25*pixel)+(0.125*pixel_direita) +
          (0.0625*pixel_esquerdabaixo)+(0.125*pixel_baixo)+(0.0625*pixel_direitabaixo))

temp[linha, coluna] = pixel2
```

Morfologia Matemática

- Dilatação
- Erosão
- Limite externo
- Limite interno
- Abertura
- Fechamento

Dilatação

O algoritmo utilizou a imagem de um “j” branco em um fundo preto(a pedido do professor). A imagem precisa ser binarizada(limiarizada) para funcionar o processo.

O processo expandiu a imagem original.

O kernel utilizado foi mostrado em aula pelo professor e é em formato de cruz.

Resultado

Trecho de código

```
if pixel != 0:  # Se o pixel central da imagem base não for 0
    pixel_cima = 255  # Replica os pixels iguais a 255 do kernel
    pixel_baixo = 255  # neste caso formato de cruz
    pixel_esquerda = 255
    pixel_direita = 255

    temp[linha-1, coluna] = pixel_cima
    temp[linha+1, coluna] = pixel_baixo
    temp[linha, coluna-1] = pixel_esquerda
    temp[linha, coluna+1] = pixel_direita
```

Erosão

O algoritmo utilizou a imagem de um “j” branco em um fundo preto. O processo contraiu a imagem original.

A erosão também precisa que a imagem base seja binarizada.

O kernel utilizado foi mostrado em aula pelo professor e é em formato de cruz.

Resultado

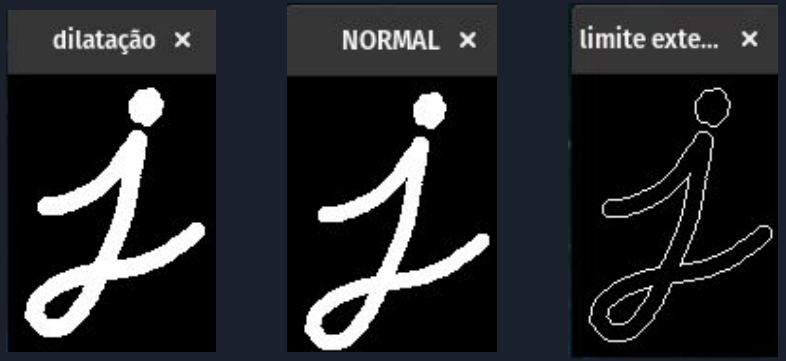
Trecho de código

```
# A IMAGEM BASE PRECISA SER IGUAL AO KERNEL NAS 5 POSIÇÕES IGUAIS A 255.
if pixel != 0 and pixel_cima != 0 and pixel_baixo != 0 and pixel_esquerda != 0 and pixel_direita != 0 :
    pixel = 255  # Se for igual a 255 então o pixel central da imagem base irá ser 255
else:  # Se não será 0
    pixel = 0

temp[linha, coluna] = pixel
```

Limite externo

O algoritmo utilizou a imagem de um “j” branco em um fundo preto e a sua versão dilatada. O processo demonstrou a diferença entre a imagem dilatada e a imagem original.

Resultado



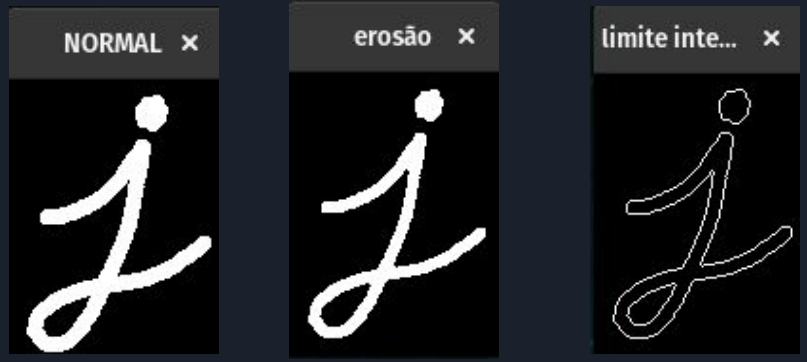
Trecho de código

```
for linha in range(1, img.shape[0]-1):  # Altura
    for coluna in range(1, img.shape[1]-1):  # Largura
        pixel = img[linha, coluna]  # Pixel original
        pixeld = dilatada[linha, coluna]  # Pixel dilatado

        result = pixeld - pixel  # Pixel dilatado - pixel original

        temp[linha, coluna] = result  # Salva na imagem temp
```

Limite interno

O algoritmo utilizou a imagem de um “j” branco em um fundo preto e a sua versão erodida. O processo demonstrou a diferença entre a imagem erodida e a imagem original.

Resultado



Trecho de código

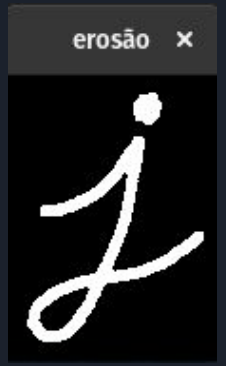
```
for linha in range(1, img.shape[0]-1):  # Altura
    for coluna in range(1, img.shape[1]-1):  # Largura
        pixel = img[linha, coluna]  # Pixel original
        pixele = erodida[linha, coluna]  # Pixel erodido

        result = pixel - pixele  # Pixel original - pixel erodido

        temp[linha, coluna] = result  # Salva na imagem temp
```

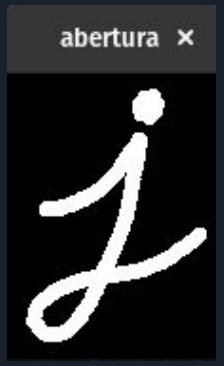
Abertura

O algoritmo utilizou a imagem erodida de um “j” branco em um fundo preto. O processo suavizou as bordas da imagem.

É feita a dilatação a partir da imagem erodida.

Resultado





Trecho de código

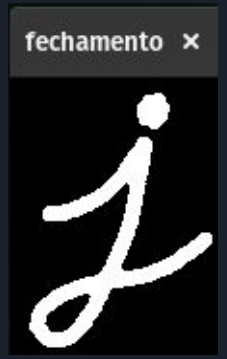
```
if pixel != 0:  # Se o pixel central da imagem base não for 0
    pixel_cima = 255  # Replica os pixels iguais a 255 do kernel
    pixel_baixo = 255  # Neste caso formato de cruz
    pixel_esquerda = 255
    pixel_direita = 255

    temp[linha-1, coluna] = pixel_cima
    temp[linha+1, coluna] = pixel_baixo
    temp[linha, coluna-1] = pixel_esquerda
    temp[linha, coluna+1] = pixel_direita
```

Fechamento

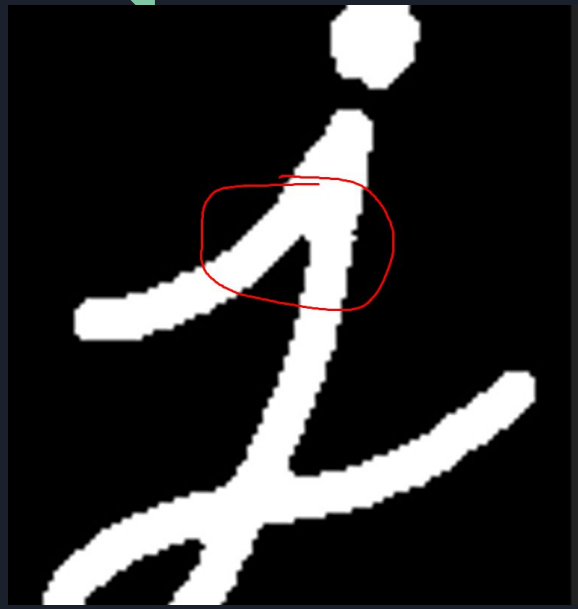
O algoritmo realizou a erosão a partir da imagem dilatada de um “j” branco em um fundo preto. O processo irá Irá preencher ou fechar os vazios.

É feito a erosão a partir da imagem dilatada.

Resultado



Resultado



FECHAMENTO

ABERTURA

IMAGEM NORMAL

Trecho de código

```
# A IMAGEM BASE PRECISA SER IGUAL AO KERNEL NAS 5 POSIÇÕES IGUAIS A 255.
if pixel != 0 and pixel_cima != 0 and pixel_baixo != 0 and pixel_esquerda != 0 and pixel_direita != 0:
    pixel = 255  # Se for igual a 255 então o pixel central da imagem base irá ser 255
else:  # Se não será 0
    pixel = 0;
temp[linha, coluna] = pixel
```

Considerações finais

O trabalho foi ótimo para aplicarmos o conteúdo mostrado em aula e realmente aprendermos como funciona na prática, nosso grupo conseguiu realizar todas as atividades somente tivemos mais dificuldade com a implementação dos algoritmos de dilatação e erosão, mas após o professor explicar novamente o conteúdo foi possível realizar a atividade também.

Agora que terminamos surgiram algumas dúvidas:

Podemos dizer que os processos de dilatação e erosão realizam alterações nas bordas das imagens base?

O que acontece quando tem overflow nos pixels da imagem, como é tratado?