Ícaro Lima Magalhães

Ciência da Computação - 11328386

Professor Leonardo Vidal Batista

Introdução ao Processamento Digital de Imagens

03 de abril de 2018

Trabalho I, Relatório

INTRODUÇÃO

Este trabalho contempla a implementação e análise crítica de diversas técnicas de processamento digital de imagens envolvendo filtros, acentuação, manipulação de características específicas de uma imagem, dentre outras técnicas. Para este fim, com base em uma interface com funções básicas para abrir, exibir e manipular imagens, foram implementados diversos algoritmos e métodos que serão detalhados ao longo do relatório. O relatório objetiva analisar o senso crítico, entendimento, implementação e resultados do aluno em relação ao tema apresentado.

MATERIAIS E MÉTODOS USADOS

Na descrição da atividade nos foi descrita a seguinte interface na qual deveríamos basear nossa implementação:

```
open_image(image_path: String)
show_image(image_obj: Image)
save_image(image_obj: Image)
```

Também nos foi dada a opção de fazer uso da biblioteca *OpenCV* para executar estas operações básicas, contanto que todo o resto da implementação fizesse manipulação direta na matriz da imagem, pixel a pixel.

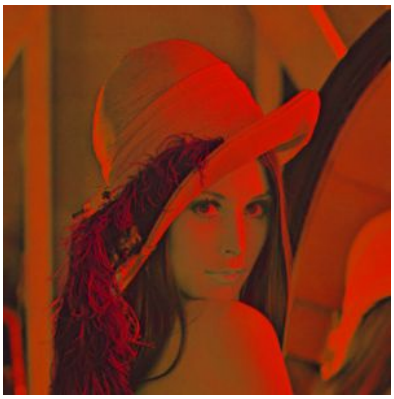
Para o desenvolvimento da atividade optei por não utilizar *OpenCV*. A linguagem escolhida foi *Python 3.6* e o módulo *numpy* foi usado para alguns cálculos simples como média e mediana, melhorando a legibilidade do código.

DETALHES DE IMPLEMENTAÇÃO

Sabendo das informações necessárias para a manipulação de imagens, foi criada uma classe com a finalidade de detalhar e expor a largura, a altura e o acesso aos pixels para que esta possa ser manipulada. Cada filtro individual recebeu seu próprio método e suas funções auxiliares. Todos serão detalhados nesta seção do relatório. Serão apresentados e analisados ambos o algoritmo em forma de pseudocódigo, porém extremamente próximos ao implementado, assim como os resultados obtidos. O dataset usado nos testes será apresentado em sessões futuras. A nível de demonstração usarei a imagem *lena.jpg* manipulada originalmente com dimensão 512x512.

Imagem original: *lena.jpg*
Algumas variações da imagem original foram usadas nos testes

1.1 - *Conversão RGB-YIQ-RGB*


RGB → ← YIQ → ← RGB

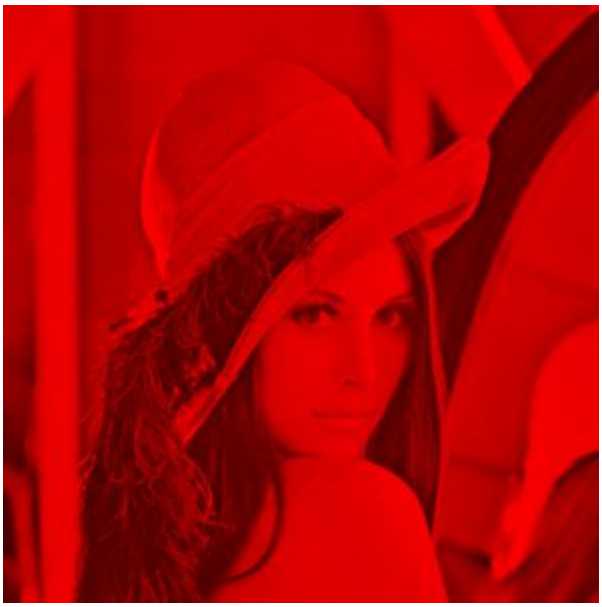
A conversão de ida e de volta entre os dois sistemas de cores basicamente se dá pelo complemento das operações. Ao converter RGB → YIQ e em seguida, sobre a imagem resultante aplicar a conversão inversa houve perda de informação levando a uma imagem levemente distorcida. Pude perceber isso ao extrair a média das bandas e analisar. Pelo que entendi a provável causa desse fenômeno é perda de precisão causada por arredondamentos entre valores de ponto flutuante que ocorre nos truncamentos e limitações das bandas durante

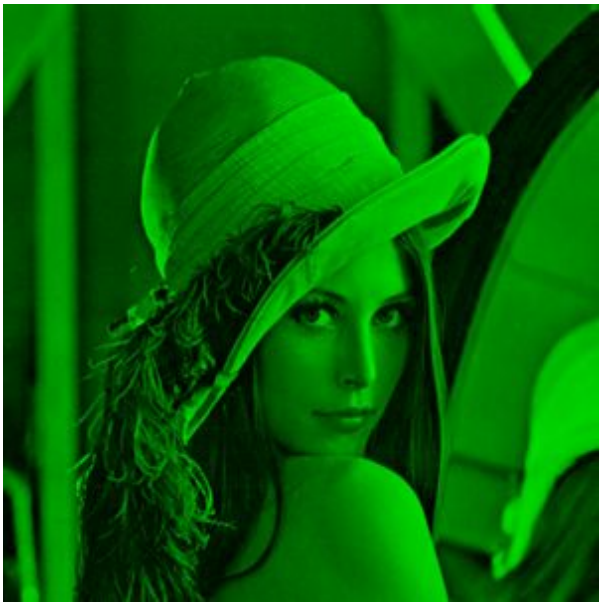
os processos de conversão.

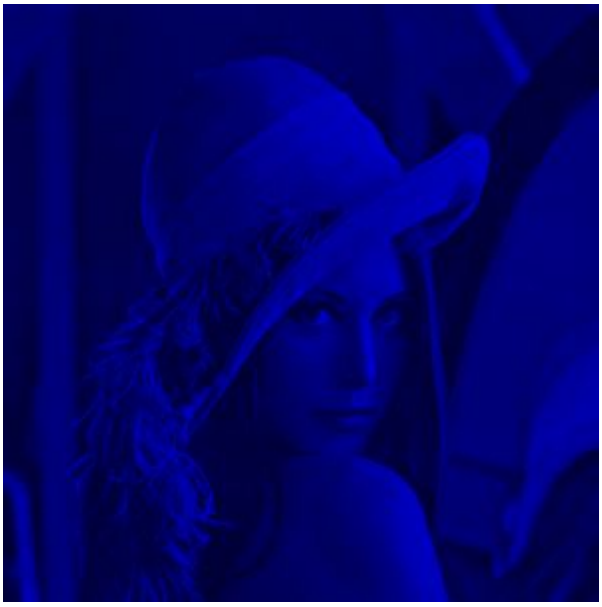
Algoritmos de conversão RGB → YIQ e YIQ→ RGB

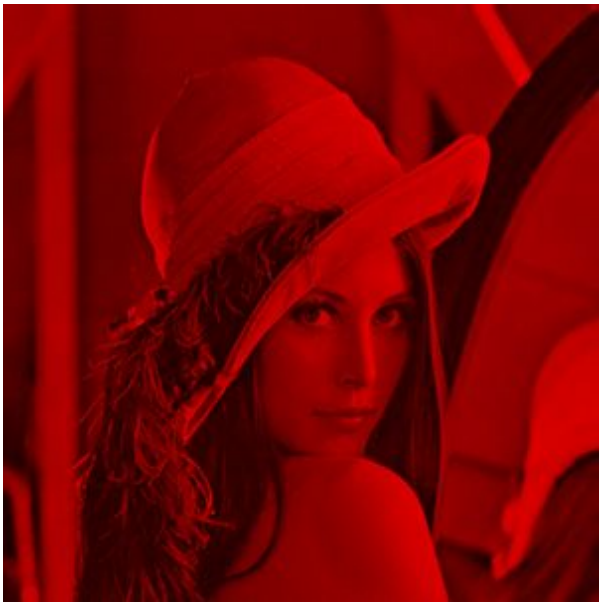
```
for i, j in image_range:
    R, G, B = pixels[i, j]
    Y, I, Q = (
        int(0.299 * R + 0.587 * G + 0.114 * B),
        int(0.596 * R - 0.275 * G - 0.321 * B),
        int(0.212 * R - 0.523 * G + 0.311 * B)
    )
    pixels[i, j] = (Y, I, Q)

for i, j in image_range:
    Y, I, Q = pixels[i, j]
    R, G, B = (
        int(1.000 * Y + 0.956 * I + 0.621 * Q),
        int(1.000 * Y - 0.272 * I - 0.647 * Q),
        int(1.000 * Y - 1.106 * I + 1.703 * Q)
    )
    pixels[i, j] = (R, G, B)
```

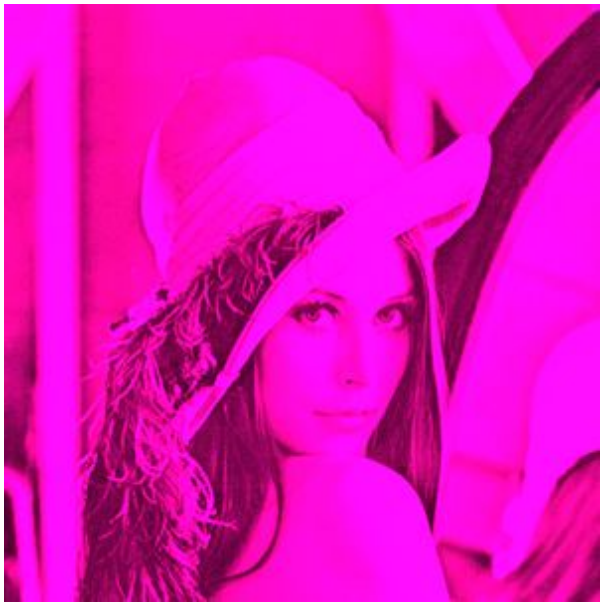
1.2 - Exibição de bandas individuais (R, G e B) como imagens monocromáticas ou coloridas (em tons de R, G ou B, respectivamente)











Bandas R, G, B, Y e Y convertida de volta para RGB, respectivamente

Variações monocromáticas de cada uma, respectivas aos quadros da esquerda

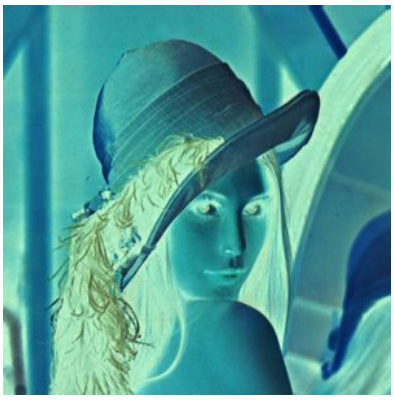
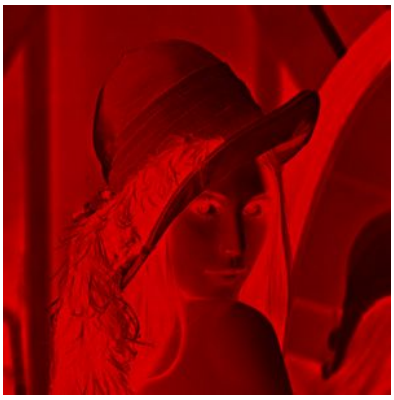
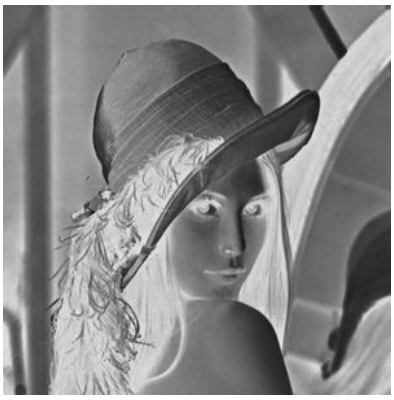
Para cada exemplo busquei manter a imagem com o espectro de cores totalmente limitado à banda em questão, e em seguida, ao converter para escala de cinza podemos notar claramente a diferença entre as bandas individuais na imagem monocromática. Para obter a banda Y foi preciso converter RGB para YIQ e em seguida aplicar a operação análoga somente sobre Y. Extra: Realizei um experimento e notei que ao converter de volta a faixa Y para RGB, a monocromática voltou com um tom rosa e a multicromática voltou exatamente como a monocromática Y em YIQ.

Bandas R, G, B e Y respectivamente Variações monocromáticas de cada um

Algoritmos de extração de bandas R, G, B, Y | monocromáticas

```
for i, j in image_range: # red [IM]
    R, _, _ = pixels[i, j]
    pixels[i, j] = (R, R, R) if monocromatic else (R, 0, 0)

for i, j in image_range:  # green [IM]
    _, G, _ = pixels[i, j]
    pixels[i, j] = (G, G, G) if monocromatic else (0, G, 0)

for i, j in image_range: # blue [IM]
    _, _, B = pixels[i, j]
    pixels[i, j] = (B, B, B) if monocromatic else (0, 0, B)

for i, j in image_range: # blue [IM]
    Y, _, _ = pixels[i, j]
    pixels[i, j] = (Y, Y, Y) if monocromatic else (Y, 0, 0)
```

1.3 - Negativo (intensidade na saída = 255 – intensidade na entrada)

RGB Banda Y Banda Y em RGB

No primeiro caso foi aplicada a operação para tornar a imagem original em negativa. No segundo caso a imagem foi convertida de RGB para YIQ e em seguida aplicada a operação negativa somente na banda Y. Com o resultado da segunda operação, a conversão inversa foi feita e a imagem voltou a RGB em negativo da banda Y, como mostra o terceiro quadro.

Algoritmos de variação em negativo

```
for i, j in image_range:
    R, G, B = pixels[i, j]
    pixels[i, j] = (255 - R, 255 - G, 255 - B)

for i, j in image_range:
    Y, _, _ = pixels[i, j]
    pixels[i, j] = (255 - Y, 0, 0)
```

1.4 e 1.5 - Controle de brilho aditivo e multiplicativo (valor do pixel resultante = valor do pixel original + c ou o valor do pixel original * c , c inteiro)

Controle de brilho aditivo
C = 100

Controle de brilho multiplicativo
C = 2

Apesar de serem dois processos análogos, o controle de brilho aditivo e o multiplicativo apresentaram resultados ligeiramente diferentes, o ruído nas imagens é visível, de modo que no primeiro exemplo o resultado parece muito uniforme e suave, já no segundo, bastante acentuado e com efeitos mais locais que globais.

Algoritmo de controle de brilho aditivo ou multiplicativo

```
for i, j in image_range:
    # Assign to the operand var the respective operator for
    #   additive or multiplicative cases [IM]
```

```
    operand = sum() if additive else product()
    R, G, B = pixels[i, j]
    R, G, B = limited_rgb(R operand.over(C), G operand.over(C), B operand.over(C))
    pixels[i, j] = (R, G, B)
```

1.6. Limiarização aplicada sobre Y, com limiar m e duas opções: a) m escolhido pelo usuário; b) m = média de valores da banda Y;

Limiarização sobre Y
M = Média de valores da banda Y

Limiarização sobre Y
M = 100

No caso de M = média dos valores da banda Y obtemos um resultado que preserva mais detalhes da imagem. No experimento, a média dos valores da banda Y gira em torno de 134, enquanto no segundo experimento, ao setar M = 100, ou seja, com um valor menor que a média anterior, notamos perda de detalhes em relação ao primeiro experimento.

Algoritmo de limiarização sobre Y

```
# The image should be converted from RGB to YIQ
#   before manipulation [IM]
image = rgb_to_yiq(image)
pixels = image.load()

# If the user has not specified the parameter,
#   the mean Y should be calculated [IM]
if M == None:
    M = mean(array(image)[:, :, 0].flatten())

for i, j in image_range:
```

```
    Y, I, Q = pixels[i, j]
    pixels[i, j] = ((255 if Y >= M else 0), 0, 0)
```

1.7. Filtros de Média e Mediana de ordem (tamanho do kernel) escolhida pelo usuário;

Filtro da média
sobre *lena_salt_and_peper.png*

Filtro da mediana
sobre *lena_salt_and_peper.png*

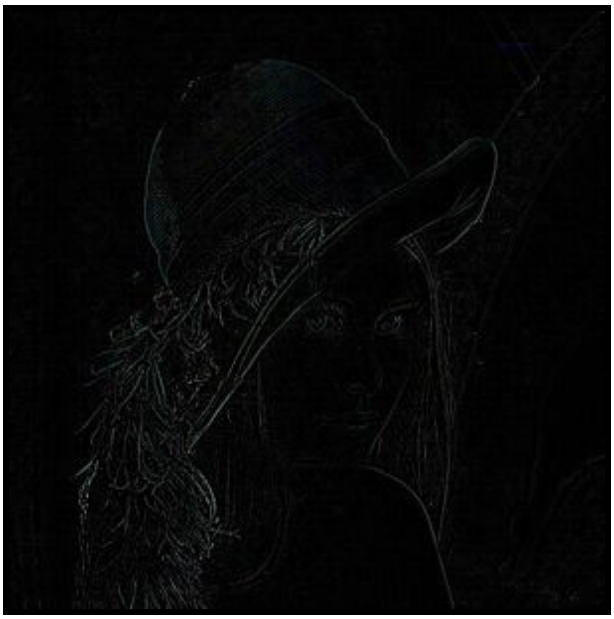
Neste experimento podemos notar a remoção do efeito *salt and peper* levemente aplicado sobre a imagem original, porém com mais definição no caso do filtro da mediana.

Filtros da média e mediana

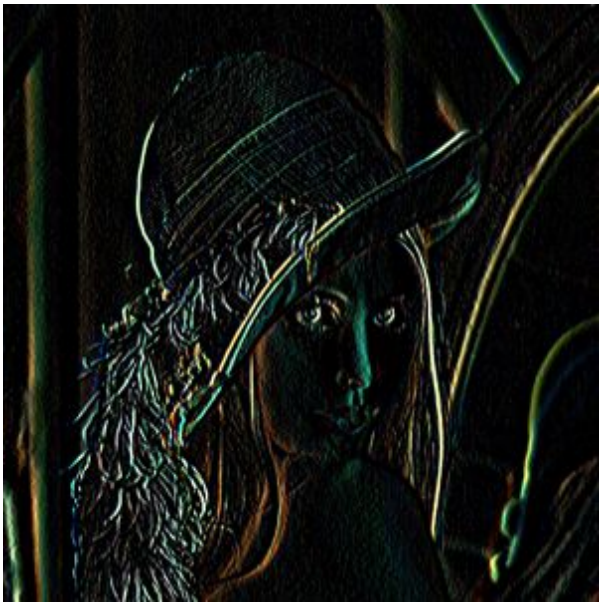
```
def integer_mask_mean(component) → mean(component).as(int)

for i, j in image_range:
    # 2d mask NxN [IM]
    mask = image[i: i + kernel_size, j: j + kernel_size]
    pixels[j, i] = (
        integer_mask_mean(mask[:, :, 0]), # mean over R
        integer_mask_mean(mask[:, :, 1]), # mean over G
        integer_mask_mean(mask[:, :, 2])  # mean over B
    )


def integer_mask_median(component) → median(component).as(int)

for i, j in image_range:
    # 2d mask NxN [IM]
```

```
mask = image[i: i + kernel_size, j: j + kernel_size]
pixels[j, i] = (
    integer_mask_median(mask[:, :, 0].flatten()), # median over R
    integer_mask_median(mask[:, :, 1].flatten()), # median over G
    integer_mask_median(mask[:, :, 2].flatten())  # median over B
)
```

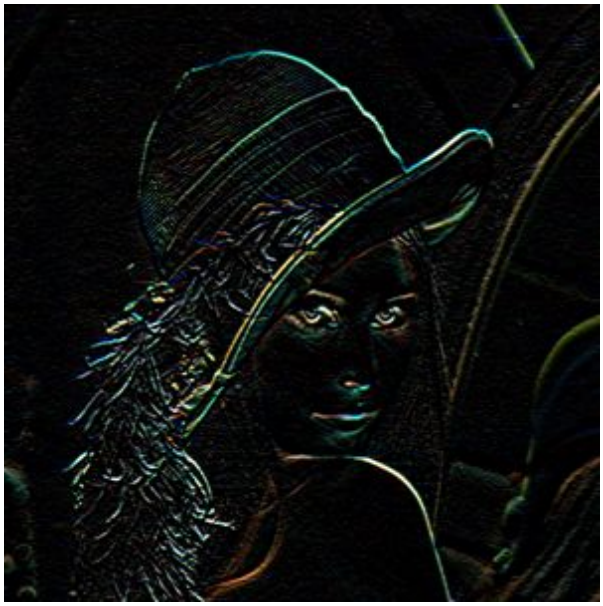
1.8 e 1.9 - Filtros de detecção de bordas Sobel e Laplaciano e dois filtros personalizados que devem ser descritos no relatório



Laplaciano

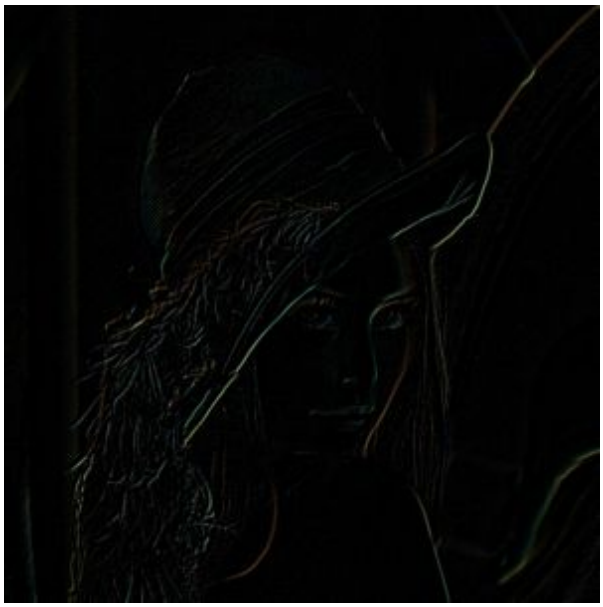


Sobel sobre o eixo X



Sobel sobre o eixo Y

Diferente do Laplaciano, os filtros Sobel aplicados na imagem acentuaram muito mais os contornos. Interessante notar a diferença no ombro da modelo nos filtros Sobel em X e em Y, deixando claro a acentuação dos contornos em X. Note também à esquerda das duas imagens, um elemento da imagem some completamente em X pois se trata de uma mancha preta na vertical. Esta fica mais visível quando o filtro é aplicado em Y.



Custom 1 Custom 2

Ao aplicar os filtros notamos que o primeiro traz à imagem mais definição, no caso, trata-se de um filtro de sharpening, porém adiciona ruído à imagem. Já o segundo filtro aplicado lembra muito o Laplaciano, porém com menos bordas à vista, porém bem mais definidas e suavizadas.

Função de convolução 2D implementada para aplicar as máscaras Laplaciana, Sobel sobre X, Sobel sobre Y, Custom1 e Custom2 e algoritmo de aplicação de máscara e convolução 2D

```
[ 0, -1,  0]        [-1,  0,  1]        [-1, -2, -1]        [ 0, -1,  0]        [0,  0,  0]
[-1,  4, -1]        [-2,  0,  2]        [ 0,  0,  0]        [-1,  5, -1]        [0,  1,  0]
[ 0, -1,  0]        [-1,  0,  1]        [ 1,  2,  1]        [ 0, -1,  0]        [0,  0, -1]

   laplace             sobel x             sobel y             custom 1            custom 2
```

```
for i, j in itertools.product(range(2, rows - 2), range(2, columns - 2)):
    newMask = image.asMatrix()[i: i + 3, j: j + 3]

    masked_R = sum(mask * newMask[:, :, 0])
    masked_G = sum(mask * newMask[:, :, 1])
    masked_B = sum(mask * newMask[:, :, 2])

    pixels[j, i] = (masked_R, masked_G, masked_B)
```

DATASET USADO

O dataset da atividade foi composto por quatro imagens nas resoluções 512x512, 256x256 e 128x128 pixels:




DIFICULDADES ENCONTRADAS E OBSERVAÇÕES

Pelo fato das operações usadas no desenvolvimento desta atividade sobre matrizes serem da ordem de complexidade de $O(N^2)$, ao aplicar kernels de tamanho N a complexidade escala para $O(N^3)$, tornando o processo de convolução e aplicação de filtros / kernels mais demorado para imagens grandes. Isso se tornou visível pois o resultado dos últimos filtros não é apresentado de imediato neste trabalho, demorando cerca de dois ou três segundos cada. Isso nos dá a impressão que estas são operações impraticáveis em dispositivos com pouco recurso em hardware. Para resolver este problema existem diversas bibliotecas que fazem manipulação direta e que tiram proveito do paralelismo para executar este tipo de processamento conseguem executar em uma fração do tempo consumido por algoritmos iterativos como os implementados na atividade, pois se tratam de cálculos vetoriais e matriciais que tiram proveito das propriedades do paralelismo. Auxílio de GPUs também são bem vindos neste tipo de processamento.

Outro ponto interessante é lidar com o sistema de cores nativo do OpenCV, que usa BGR ao invés de RGB e pode causar confusão em pessoas que nunca tiveram um contato com os conceitos de processamento digital de imagens, oferecendo resultados estranhos ao se manipular imagens em BGR ao invés de RGB, exigindo mais camadas de conversões entre espaços de cores. Este foi um dos motivos que me fez optar por só usar componentes nativos da linguagem e abrir mão do OpenCV na atividade.

CONCLUSÃO

O trabalho traz diversas técnicas interessantes e aprofunda conceitos muito importantes do processamento digital de imagens, abrindo bastante margem para experimentação. Os resultados foram satisfatórios e estão de acordo com o esperado, os problemas oferecidos pelo desenvolvimento da atividade forçam o aluno a destrinchar as operações a nível de pixel e

entender o impacto de cada uma delas sobre a imagem, analisando suas qualidades e efeitos. Conhecer estes conceitos nos torna mais independentes, críticos e conscientes em relação à importância do processamento digital de imagens em diversos campos da ciência, assim como em atividades cotidianas como, por exemplo, a aplicação de um filtro em uma foto e seu impacto na performance de uma aplicação.