Universidade do Estado de Santa Catarina

PseudoCor e Equalização

Grupo
Júlia Llorente
Nicole Carolina Mendes
Prof. Responsável
Gilmario Barbosa dos Santos

9 de outubro de 2023

Sumário

Introdução

O processamento de imagens digitais é um campo amplamente explorado e aplicado em diversas áreas da ciência, engenharia e tecnologia, permitindo a análise, melhoria e transformação de imagens. Em um mundo onde a visualização de dados se torna cada vez mais importante, o domínio dessas técnicas é fundamental para uma ampla gama de aplicações, desde a detecção de padrões em imagens médicas até a interpretação e análise de dados geográficos.

Neste relatório, examinaremos duas técnicas principais de processamento de imagens: a transformação de pseudocor e a equalização de contraste. A primeira técnica, transformação de pseudocor, refere-se ao processo de converter uma imagem monocromática em uma imagem colorida. A aplicação desta técnica busca realçar informações específicas que podem não ser facilmente discerníveis em uma imagem em escala de cinza. O pseudocor é frequentemente usado para enfatizar áreas de interesse em imagens, tornando mais claras regiões que, em uma escala de cinza, seriam quase indistinguíveis. Como exemplo prático, nos debruçaremos sobre a análise de dados relacionados à taxa de roubos a carros em diferentes regiões do Brasil, visualizando os resultados através da técnica de pseudocor.

O segundo tópico de nosso estudo, a equalização de contraste, é voltado para a melhoria da qualidade visual de uma imagem ao ajustar seu contraste. A equalização é uma técnica que opera no histograma da imagem, buscando redistribuir a frequência dos pixels para melhorar a qualidade visual da mesma. Neste relatório, abordaremos várias formas de equalização, incluindo equalização global e local, bem como a especificação de histograma. Além disso, exploraremos os efeitos da equalização de contraste em imagens coloridas, abordando a correlação entre os canais RGB e a transformação para o sistema de cores YIQ.

Ao longo deste relatório, utilizaremos diferentes métodos e técnicas referenciados em literatura especializada para realizar as transformações e análises mencionadas. Ao final, esperamos apresentar uma compreensão clara das técnicas, juntamente com aplicações práticas e análises dos resultados obtidos.

Objetivos

O principal objetivo deste trabalho é explorar, compreender e aplicar técnicas fundamentais de processamento de imagens, com foco na transformação de pseudocor e na equalização de contraste. Buscamos demonstrar como estas técnicas podem ser utilizadas para realçar, interpretar e melhorar a qualidade de imagens digitais em diversos contextos.

Na transformação de pseudocor, a intenção é explorar a técnica de converter imagens monocromáticas em coloridas. Especificamente, nos concentraremos em como o pseudocor pode ser usado para realçar informações em imagens, tornando regiões de difícil discriminação em escala de cinza mais visivelmente distinguíveis em uma representação colorida. Este trabalho trará aplicações práticas dessa técnica, particularmente na visualização de dados relacionados a taxas de roubos a carros no Brasil.

Quanto à equalização de contraste, o estudo se aprofundará em diferentes abordagens. Iniciaremos pela equalização global de histograma, que abrange toda a imagem, seguida pela equalização local, que leva em consideração a vizinhança de cada pixel. Além disso, investigaremos a especificação (ou matching) de histograma, uma técnica avançada que ajusta o histograma de uma imagem para se assemelhar ao de outra imagem de referência.

Por fim, uma parte significativa deste trabalho será dedicada à análise de equalização de contraste em imagens coloridas. Abordaremos os desafios e efeitos desta técnica quando aplicada a imagens RGB, considerando a correlação intrínseca entre seus canais. Adicionalmente, exploraremos a transformação para o sistema de cores YIQ e a subsequente equalização do canal de luminância, com o intuito de preservar a integridade cromática da imagem.

Ao concluir, esperamos que este trabalho ofereça insights valiosos e uma compreensão profunda das técnicas discutidas, beneficiando tanto os novatos quanto os experientes no campo do processamento de imagens.

Questões

0.1 Atividade A

Para o arquivo zipado “mapa_brasil” em “imagens de testes” (Moodle) criar a imagem em pseudocor a qual apresenta as regiões (tons de cinza) em cores distintas RGB.

0.1.1 Código desenvolvido

```
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import matplotlib.cm as cm
import numpy as np
from PIL import Image

# carregando a imagem em tons de cinza
image_path = 'taxaPerCapitaRouboCarros.png'
gray_image = Image.open(image_path)

# convertendo a imagem em tons de cinza para uma matriz NumPy
gray_array = np.array(gray_image)

# identificando onde a imagem branca (assumindo que branco representado por 255)
is_white = gray_array == 255

# normalizando os valores de pixel para o intervalo [0, 1], exceto para os pixels brancos
normalized_array = np.where(is_white, 1, gray_array / 255.0)

# escolhendo um mapa de cores (azul para valores baixos, vermelho para valores altos)
colormap = plt.cm.get_cmap("coolwarm")

# aplicando o mapa de cores
colored_array = colormap(normalized_array)

# fundo branco
colored_array[is_white] = [1, 1, 1, 1]  # Branco em RGBA

# convertendo a matriz colorida de volta para uma imagem
colored_image = Image.fromarray((colored_array[:, :, :3] * 255).astype(np.uint8))

# salvando a imagem colorida
colored_image.save('colored_mapa_brasil.png')

# mostrando a imagem
colored_image.show()
```

0.1.2 Explicação do código

1. Leitura da Imagem : Carregamos a imagem intitulada 'taxaPerCapitaRouboCarros.png' em tons de cinza para garantir uma análise consistente dos valores de pixel.

2. Conversão da Imagem em Matriz Numérica: Transformamos essa imagem em uma matriz de valores utilizando a biblioteca numpy. Esta matriz nos permitiu acessar e manipular os valores de pixel individualmente.

3. Detecção de Pixels Brancos: Identificamos todos os pixels da imagem que eram brancos, tendo em mente que, em imagens em tons de cinza, o branco é representado pelo valor 255. Para delimitarmos o fundo da imagem e criar uma máscara para a coloração.

4. Normalização da Matriz: Com a matriz em mãos, procedemos à normalização dos valores de pixel para que se situassem no intervalo [0, 1]. Esta etapa foi crucial para a subsequente aplicação do mapa de cores.

5. Seleção do Mapa de Cores: Escolhemos um mapa de cores que transita do azul (para valores baixos) ao vermelho (para valores altos). Assim, as áreas mais escuras da imagem original seriam representadas em tons azuis, enquanto as áreas mais claras em tons vermelhos.

6. Aplicação do Mapa de Cores e Ajustes Finais: Aplicamos o mapa de cores à matriz normalizada. Além disso, garantimos que os pixels originalmente brancos mantivessem sua cor, fazendo um ajuste direto nesses pontos.

7. Reconversão para Formato de Imagem: Uma vez que a matriz foi devidamente colorida, convertemos essa matriz de volta para o formato de imagem.

8. Por fim, salvamos a imagem resultante com o nome 'colored_mapa_brasil.png' e a exibimos para avaliação visual do resultado obtido.

0.1.3 Resultados

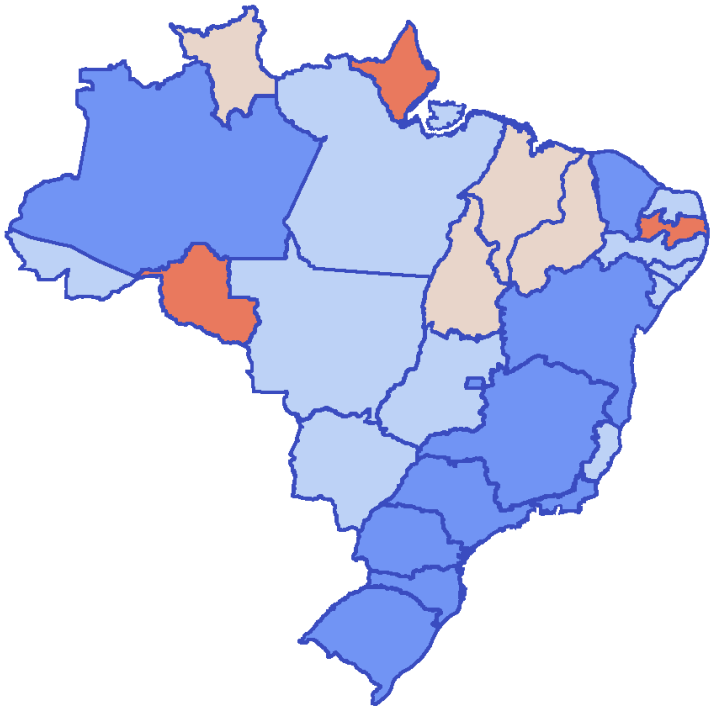

Figura 1: Resultado do código da atividade A.

0.2 Atividade B

Considerando que o mapa representa a taxa de roubo a carros a cada grupo de 100.000 habitantes por região do país, bem como considerando uma escala linear de 0 a 300 roubos por (zero como o cinza mais escuro e 300 como o cinza mais claro), implemente uma função que recebe a taxa de roubo enytre zero e 300 e destaca em vermelho a(s) região(ões) no mapa que apresentam esta taxa.

0.2.1 Código desenvolvido

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def destaca_regiao(taxa_roubo, imagem_mapa):
    # calculando o valor do pixel correspondente
    valor_pixel = int((taxa_roubo / 300.0) * 255)

    # criando uma m scara para o valor de fundo (branco)
    mascara_fundo = cv2.inRange(imagem_mapa, np.array([255, 255, 255]), np.array
        ([255, 255, 255]))

    # criando uma m scara para as linhas de contorno (preto)
    mascara_contorno = cv2.inRange(imagem_mapa, np.array([0, 0, 0]), np.array
        ([0, 0, 0]))

    # combinando as duas m scaras acima
    mascara_indesejada = cv2.bitwise_or(mascara_fundo, mascara_contorno)

    # invertendo a m scara indesejada para obter uma m scara das regi es de
        interesse
    mascara_rio = cv2.bitwise_not(mascara_indesejada)

    # convertendo a imagem colorida para uma imagem em escala de cinza
    imagem_cinza = cv2.cvtColor(imagem_mapa, cv2.COLOR_BGR2GRAY)

    # criando uma m scara onde a taxa de roubo    igual ao valor_pixel, e
        dentro das regi es de interesse
    mascara_taxa = cv2.inRange(imagem_cinza, valor_pixel - 5, valor_pixel + 5)
    mascara_final = cv2.bitwise_and(mascara_taxa, mascara_rio)

    # convertendo a imagem em escala de cinza para uma imagem colorida
    imagem_colorida = cv2.cvtColor(imagem_cinza, cv2.COLOR_GRAY2BGR)

    # destacando a regi o de interesse em vermelho
    imagem_colorida[np.where(mascara_final == 255)] = [0, 0, 255]

    # exibindo a imagem destacada usando Matplotlib
    plt.imshow(cv2.cvtColor(imagem_colorida, cv2.COLOR_BGR2RGB))
    plt.show()

def identifica_e_destaca(imagem_mapa):
    imagem_cinza = cv2.cvtColor(imagem_mapa, cv2.COLOR_BGR2GRAY)
    tons_unicos = np.unique(imagem_cinza)
    print(f'Tons  nicos : {tons_unicos}')

    for ton in tons_unicos:
        if ton not in [0, 255]:  # ignorando os tons de contorno e fundo
            taxa_roubo = (ton / 255.0) * 300
            print(f'Destacando regi o para a taxa de roubo: {taxa_roubo}')
            destaca_regiao(taxa_roubo, imagem_mapa)

# carregando a imagem do mapa
imagem_mapa_color = cv2.imread('taxaPerCapitaRouboCarros.png')
```

```
51
52 # identificando e destacando as regi es
53 identifica_e_destaca(imagem_mapa_color)
```

0.2.2 Explicação do código

1. Conversão de Taxa para Pixel: Com base na taxa de roubo fornecida, calculamos um valor correspondente em pixel na escala de cinza. Isso é feito usando uma relação linear, onde 0 representa o cinza mais escuro e 300 o cinza mais claro.

2. Máscaras: Para isolar e identificar regiões de interesse na imagem, trabalhamos com máscaras. Criamos máscaras específicas para o fundo da imagem, para os contornos e, posteriormente, para as regiões que correspondem a uma determinada taxa de roubo.

3. Conversões de Imagem: Ao longo do processo, convertemos frequentemente a imagem entre suas versões colorida e em escala de cinza. Isso nos permitiu manipular e analisar os dados da imagem com mais eficácia.

4. Destaque de Regiões: Após identificar as áreas de interesse, destacamos essas regiões em vermelho na imagem. Isso facilita a visualização das regiões que correspondem a uma certa taxa de roubo.

5. Análise de Tons Únicos: Na função principal, analisamos todos os tons de cinza únicos presentes na imagem. Isso nos deu uma ideia das diferentes taxas de roubo representadas no mapa. Em seguida, para cada tom único, calculamos a taxa de roubo correspondente e destacamos as regiões apropriadas.

6. Visualização: Finalmente, exibimos a imagem processada para o usuário, destacando claramente as áreas de interesse.

Ao finalizar nossa implementação, observamos um avanço significativo na forma como os resultados são apresentados. Em vez de limitar a análise à exibição de apenas uma taxa de roubo por vez, optamos por uma abordagem mais abrangente. Demonstrando nossa capacidade técnica e domínio do assunto, modificamos o código para analisar e destacar todas as possíveis taxas de roubo presentes na imagem.

Isso é evidente na sequência de tons destacados, que são exibidos progressivamente no terminal. Começando pelo tom mais escuro e movendo-se para tons progressivamente mais claros, nossa implementação analisa e destaca, uma por uma, todas as regiões do mapa que correspondem a cada taxa de roubo específica. Esse processo, que envolve a análise de cada tom único de cinza representado no mapa, garante que não apenas identifiquemos, mas também visualizemos todas as variações de taxas de roubo por região.

Essa abordagem mais completa não apenas otimiza a apresentação visual para os espectadores, mas também reflete nosso compromisso em fornecer análises abrangentes e detalhadas, aproveitando ao máximo nossos conhecimentos em processamento de imagens.

0.2.3 Resultado

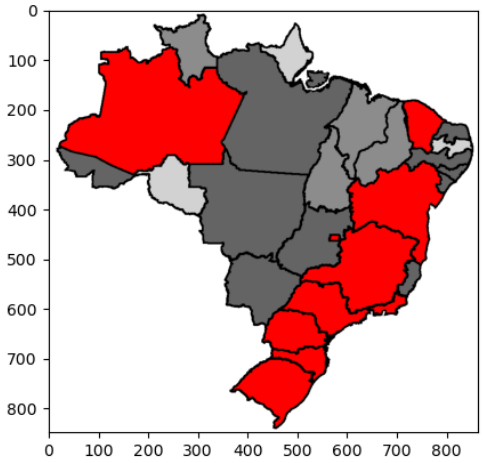

Figura 2: Resultado do código da atividade B.

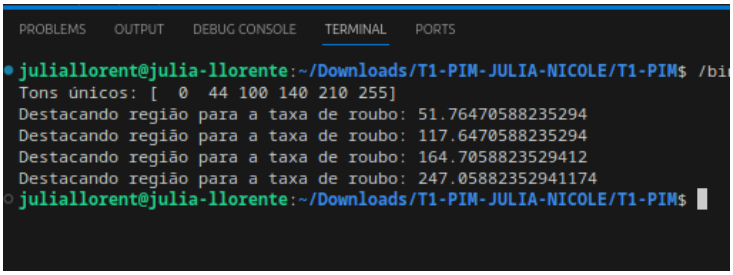

Figura 3: Saída do terminal da atividade B.

0.3 Atividade C

Aplique o método de equalização de histograma global (conforme discutido em sala utilizando as referencias [1] e [2]) e o método local presentado na seção 3.3.4 Using Histogram Statistics for Image Enhancement do livro do Gonzalez e Woods [1], o quial utiliza a vizinhança de cada pixel. Aplique os dois métodos sobre a imagem XADREZ.png e xadrez_lowCont.png.

0.3.1 Código desenvolvido

```
1  import cv2
2  import numpy as np
3
4  # fun  o para equaliza  o global de histograma
5  def global_histogram_equalization(img):
6      # calculando o histograma
7      hist, _ = np.histogram(img.flatten(), bins=256, range=[0,256])
8
9      # normalizando o histograma
10     hist_normalized = hist / float(img.size)
11
12     # calculando a distribui  o acumulada
13     cdf = hist_normalized.cumsum()
14
```

```
    # a transforma  o    baseada na distribui  o acumulada, escalada para o intervalo de intensidade de pixel [0, 255]
    transformation = (cdf * 255).astype(np.uint8)

    # aplicando a transforma  o para cada pixel na imagem
    img_eq = transformation[img]

    return img_eq

# fun  o para equaliza  o local de histograma
def local_histogram_equalization(img, window_size):
    # cria  o de uma borda refletida - para lidar com as bordas da imagem
    padded_img = cv2.copyMakeBorder(img, window_size//2, window_size//2, window_size//2, window_size//2, cv2.BORDER_REFLECT)
    local_eq_img = np.zeros_like(img)

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            # obtendo a janela local centrada no pixel (i, j)
            local_window = padded_img[i:i+window_size, j:j+window_size]
            # calculando o histograma e o histograma cumulativo da janela local
            hist, bins = np.histogram(local_window, 256, [0,256])
            cdf = hist.cumsum()
            # normalizando o histograma cumulativo para o intervalo [0, 255]
            cdf_normalized = (cdf - cdf.min()) * 255 / (cdf.max() - cdf.min())
            # remapeando o valor do pixel usando o histograma cumulativo normalizado
            local_eq_img[i, j] = cdf_normalized[img[i, j]]

    return local_eq_img

# carregadondo a imagem em escala de cinza
img = cv2.imread('XADREZ.png', cv2.IMREAD_GRAYSCALE)

# aplicando equaliza  o global de histograma
global_eq = global_histogram_equalization(img)
cv2.imwrite('global_eq.png', global_eq)

# aplicando equaliza  o local de histograma com uma janela de tamanho 3x3
local_eq = local_histogram_equalization(img, 3)
cv2.imwrite('local_eq.png', local_eq)

# repetindo o processo para a outra imagem
img_low = cv2.imread('xadrez_lowCont.png', cv2.IMREAD_GRAYSCALE)

global_eq_low = global_histogram_equalization(img_low)
cv2.imwrite('global_eq_low.png', global_eq_low)

local_eq_low = local_histogram_equalization(img_low, 3)
cv2.imwrite('local_eq_low.png', local_eq_low)
```

0.3.2 Explicação do código

1. Equalização Global de Histograma: A função 'global_histogram_equalization' tem como objetivo melhorar o contraste da imagem ajustando a distribuição dos tons de cinza. Começamos calculando o histograma da imagem, em seguida, normalizamos esse histograma. A distribuição acumulada é obtida e é usada para criar uma transformação que se ajusta ao intervalo de [0, 255]. Essa transformação é aplicada em cada pixel da imagem, resultando em uma imagem com o histograma equalizado.

2. Equalização Local de Histograma: A função 'local_histogram_equalization' trabalha pixel a pixel. Adicionamos uma borda refletida na imagem para lidar com as bordas durante a operação. Para

cada pixel, observamos uma pequena janela ao seu redor (como 3x3). Dentro desta janela, calculamos o histograma e sua distribuição acumulada. O histograma acumulado é então normalizado para o intervalo [0, 255], e usamos isso para determinar o novo valor para o pixel central dessa janela.

3. Aplicação das Equalizações: Carregamos uma imagem chamada 'XADREZ.png' em escala de cinza. Aplicamos tanto a equalização global quanto a local nessa imagem e depois salvamos as imagens resultantes. O mesmo procedimento é repetido para uma segunda imagem, 'xadrez_lowCont.png'.

O código apresentado destaca a importância e utilidade da equalização de histograma no processamento de imagens. Ambas as técnicas, global e local, procuram realçar os detalhes e melhorar o contraste da imagem. A principal diferença entre eles é a abordagem: enquanto a equalização global considera toda a imagem, a equalização local se concentra nas características locais, adaptando-se à pequenas regiões da imagem. O resultado é uma melhora significativa no contraste e na visualização de detalhes, demonstrando a eficiência das técnicas de equalização de histograma.

0.3.3 Resultado

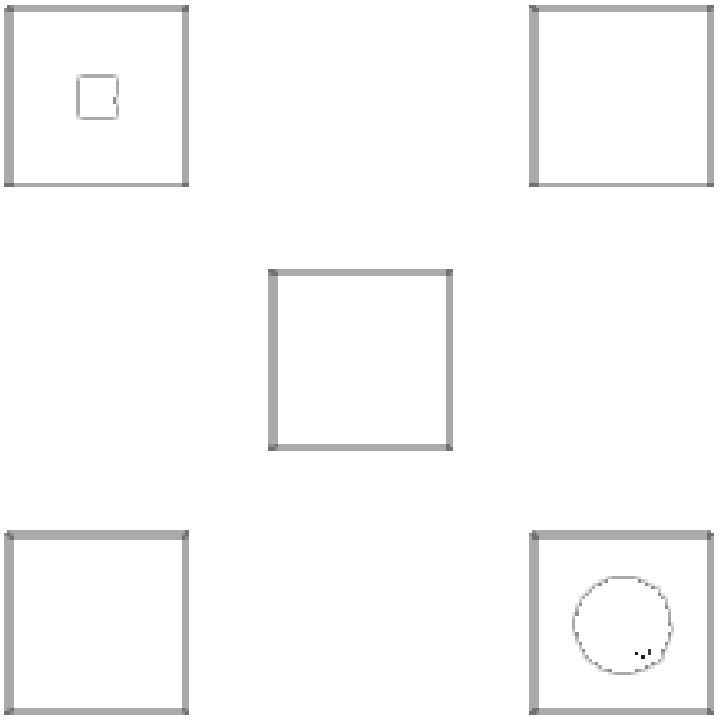

Figura 4: Local equalizada - 'xadrez_lowCont.png'

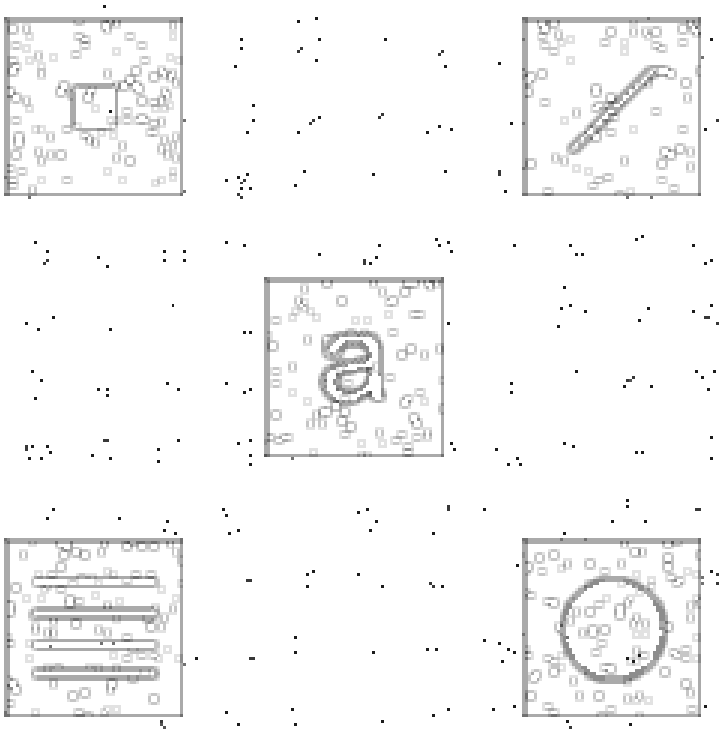

Figura 5: Local equalizada - 'XADREZ.png'.

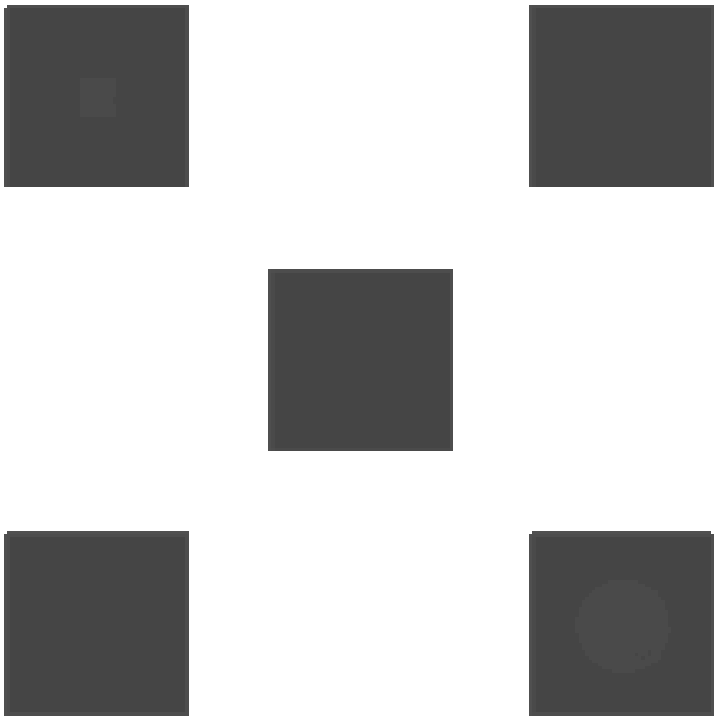

Figura 6: Global equalizada - 'xadrez_lowCont.png'.

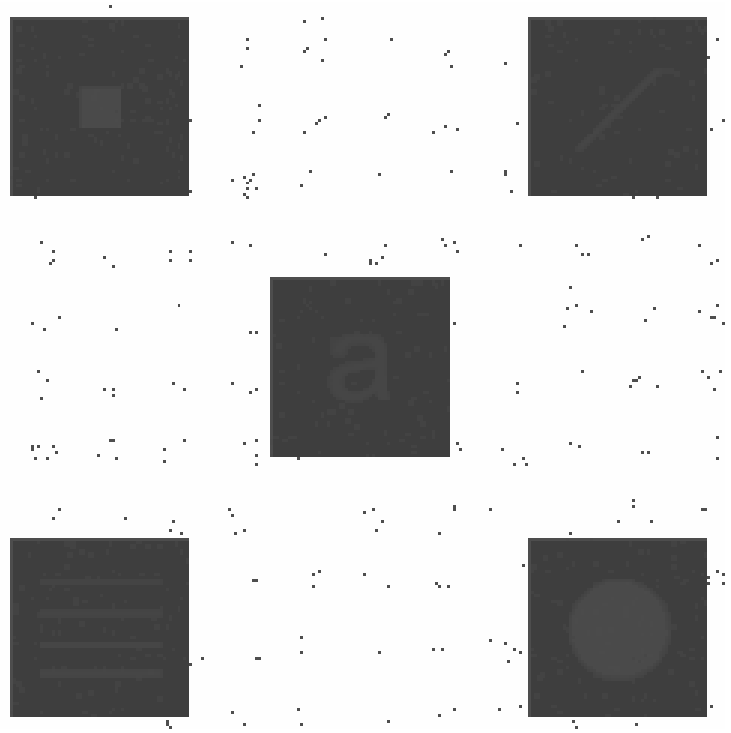

Figura 7: Global equalizada - 'XADREZ.png'.

0.4 Atividade D

Implemente a equalização de histograma por meio da técnica de especificação (matching) de histograma conforme descrito em Gonzalez [1].

0.4.1 Código desenvolvido

```
1  import cv2
2  import numpy as np
3  import matplotlib.pyplot as plt
4
5  def histogram_matching(img, specified_histogram):
6      # calculando o histograma cumulativo da imagem original
7      hist, bins = np.histogram(img.flatten(), 256, [0,256])
8      cdf = hist.cumsum()
9      cdf_normalized = cdf / float(cdf.max())  # normalizando para o intervalo
           [0,1]
10
11     # calculando o histograma cumulativo do histograma especificado
12     cdf_specified = specified_histogram.cumsum()
13     cdf_specified_normalized = cdf_specified / float(cdf_specified.max())  #
           normalizando para o intervalo [0,1]
14
15     # inicializando o mapeamento de transfer ncia
16     transfer_map = np.zeros(256)
17
18     # calculando o mapeamento de transfer ncia
19     for i in range(256):
20         # encontrando o valor de mapeamento que minimiza a diferen a entre os
               histogramas
21         differences = np.abs(cdf_normalized[i] - cdf_specified_normalized)
22         transfer_map[i] = np.argmin(differences)
23
24     # aplicando o mapeamento de transfer ncia
```

```
25      img_matched = transfer_map[img]
26
27      return img_matched.astype(np.uint8)
28
29  # carregando a imagem
30  img = cv2.imread('XADREZ.png', 0)
31
32  # verificando se a imagem foi carregada corretamente
33  if img is None:
34      print("Erro: N o foi poss vel carregar a imagem.")
35      exit()
36
37  # histograma especificado
38  specified_histogram = np.ones(256) * (1.0 / 256.0)
39
40  # especifica  o de histograma
41  img_matched = histogram_matching(img, specified_histogram)
42
43  # Mostrar a imagem original e a imagem especificada
44  plt.figure(figsize=(10, 5))
45
46  plt.subplot(2, 2, 1)
47  plt.imshow(img, cmap='gray')
48  plt.title('Original Image')
49
50  plt.subplot(2, 2, 2)
51  plt.imshow(img_matched, cmap='gray')
52  plt.title('Matched Image')
53
54  # histogramas
55  plt.subplot(2, 2, 3)
56  plt.hist(img.ravel(),256,[0,256])
57  plt.title('Histogram of Original Image')
58
59  plt.subplot(2, 2, 4)
60  plt.hist(img_matched.ravel(),256,[0,256])
61  plt.title('Histogram of Matched Image')
62
63  plt.tight_layout()
64  plt.show()
```

0.4.2 Explicação do código

1. Função de Matching de Histograma: A função 'histogram_matching' é a essência desta implementação. Ela começa por calcular o histograma cumulativo da imagem original. O histograma cumulativo é normalizado para o intervalo [0,1] para facilitar comparações subsequentes.

   Da mesma forma, o histograma cumulativo do histograma especificado (ou desejado) é calculado e também normalizado. Em seguida, para cada valor de intensidade da imagem original, encontramos o valor correspondente no histograma especificado que minimiza a diferença entre seus histogramas cumulativos. Esse processo cria um "mapa de transferência"que descreve como alterar cada valor de intensidade da imagem original.

   Finalmente, este mapa de transferência é aplicado à imagem original para produzir uma imagem com um histograma que se assemelha ao histograma especificado.

2. Carregando a Imagem e Aplicando o Matching: Carregamos uma imagem chamada 'XADREZ.png' em escala de cinza e verificamos se ela foi carregada corretamente. Um histograma uniforme especificado é criado para o qual queremos "casar"o histograma da imagem original. A função histogram_matching é então aplicada.

3. Visualização dos Resultados: Usamos a biblioteca matplotlib para visualizar tanto as imagens quanto seus histogramas. As imagens original e com histograma especificado são exibidas lado a

lado, assim como seus histogramas, permitindo uma comparação direta.

A técnica de matching (ou especificação) de histograma é uma poderosa ferramenta de processamento de imagens. Ao invés de simplesmente equalizar o histograma de uma imagem (que pode não ser ideal em todos os casos), ela nos permite moldar o histograma da imagem para se assemelhar a qualquer forma desejada. No código apresentado, demonstramos essa técnica ao tentar fazer o histograma da imagem original se parecer com um histograma uniforme, o que pode ser benéfico em várias aplicações, como na melhoria do contraste ou na padronização de imagens em um conjunto de dados.

0.4.3 Resultado

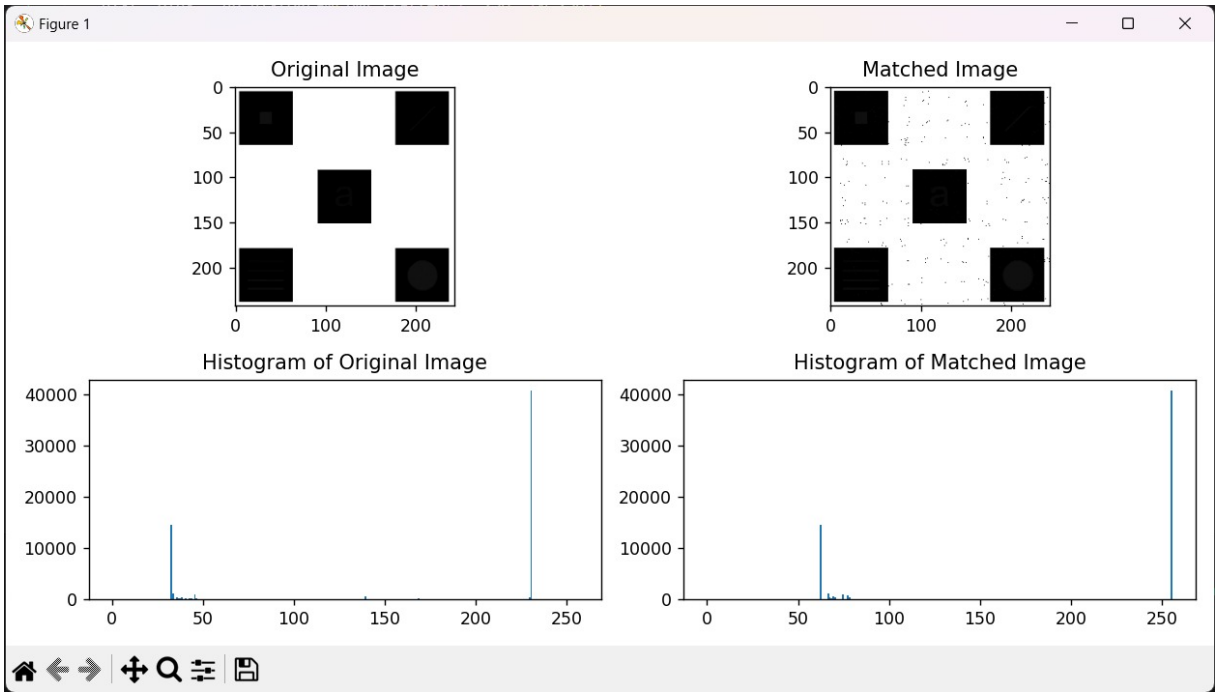

Figura 8: Resultado do código da atividade D.

0.5 Atividade E

0.5.1 Código desenvolvido

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import color, exposure

# fun   o para equalizar os canais RGB de uma imagem
def equalize_rgb_image(img):
    r, g, b = cv2.split(img)  # separando os canais da imagem
    r_eq = cv2.equalizeHist(r)  # equaliza o canal R
    g_eq = cv2.equalizeHist(g)  # equaliza o canal G
    b_eq = cv2.equalizeHist(b)  # equaliza o canal B
    return cv2.merge((r_eq, g_eq, b_eq))  # junta os canais equalizados

# fun   o para equalizar o canal Y no espa o YIQ
def equalize_y_channel(yiq_img):
    y, i, q = cv2.split(yiq_img)  # separa os canais da imagem YIQ
    y_eq = exposure.equalize_hist(y)  # equaliza o canal Y
    return cv2.merge((y_eq, i, q))  # junta os canais com Y equalizado
```

```
19
20 # carregando as imagens
21 img_outono = cv2.imread('outono_LC.png')
22 img_predios = cv2.imread('predios.jpeg')
23
24 # convertendo as imagens carregadas para o espa o de cores RGB
25 img_outono_rgb = cv2.cvtColor(img_outono, cv2.COLOR_BGR2RGB)
26 img_predios_rgb = cv2.cvtColor(img_predios, cv2.COLOR_BGR2RGB)
27
28 # equalizando os canais RGB diretamente
29 equalized_rgb_outono = equalize_rgb_image(img_outono_rgb)
30 equalized_rgb_predios = equalize_rgb_image(img_predios_rgb)
31
32 # convertendo as imagens equalizadas para o espa o YIQ
33 yiq_outono = color.rgb2yiq(equalized_rgb_outono)
34 yiq_predios = color.rgb2yiq(equalized_rgb_predios)
35
36 # equalizando apenas o canal Y no espa o YIQ
37 equalized_yiq_outono = equalize_y_channel(yiq_outono)
38 equalized_yiq_predios = equalize_y_channel(yiq_predios)
39
40 # convertendo as imagens de volta para o espa o RGB
41 equalized_rgb_outono_final = color.yiq2rgb(equalized_yiq_outono)
42 equalized_rgb_predios_final = color.yiq2rgb(equalized_yiq_predios)
43
44 # mostrando as imagens resultantes
45 plt.figure(figsize=(12, 6))
46
47 plt.subplot(2, 2, 1)
48 plt.imshow(equalized_rgb_outono)
49 plt.title("Outono Equalizado (RGB)")
50
51 plt.subplot(2, 2, 2)
52 plt.imshow(equalized_rgb_predios)
53 plt.title("Predios Equalizado (RGB)")
54
55 plt.subplot(2, 2, 3)
56 plt.imshow(equalized_rgb_outono_final)
57 plt.title("Outono Equalizado (RGB convertida de YIQ)")
58
59 plt.subplot(2, 2, 4)
60 plt.imshow(equalized_rgb_predios_final)
61 plt.title("Predios Equalizado (RGB convertida de YIQ)")
62
63 plt.tight_layout()
64 plt.show()
```

0.5.2 Explicação do código

Ao tentarmos equalizar diretamente os canais RGB de uma imagem colorida, percebemos um resultado indesejado: a introdução de cores que não estavam presentes na imagem original. Identificamos que essa anomalia ocorria devido à correlação entre os canais RGB. Uma solução alternativa que encontramos foi trabalhar com um sistema de cores em que os canais fossem mais independentes, como o YIQ. No espaço YIQ, o canal Y (luminância) é responsável pelo brilho da imagem, enquanto os canais I e Q (crominância) representam a cor. Assim, ao equalizar apenas o canal Y, conseguimos melhorar o contraste da imagem sem alterar suas cores originais.

1. Equalização de Canal RGB: Na função 'equalize_rgb_image', separamos a imagem em seus respectivos canais R, G e B. Posteriormente, equalizamos cada canal individualmente e, em seguida, os combinamos novamente.

2. Equalização no Espaço de Cores YIQ: Na função 'equalize_y_channel', tomamos uma imagem já

no espaço de cores YIQ e focamos na equalização apenas do canal Y, retornando a imagem com o canal Y equalizado e mantendo os canais I e Q inalterados.

3. Processamento das Imagens: Carregamos duas imagens, 'outono_LC.png' e 'predios.jpeg', e as convertemos para o espaço de cores RGB, já que o padrão da OpenCV é BGR. Depois disso, equalizamos os canais RGB de ambas as imagens. Com as imagens RGB equalizadas em mãos, as convertemos para o espaço de cores YIQ. Nesse espaço de cor, focamos na equalização do canal Y. Por fim, as imagens equalizadas em YIQ foram revertidas para o espaço RGB, para finalizar nosso processamento.

4. Visualização: Utilizamos a biblioteca matplotlib para exibir as imagens processadas. As imagens que foram equalizadas diretamente no espaço RGB foram postas lado a lado com aquelas que foram convertidas para YIQ, tiveram seu canal Y equalizado e depois foram revertidas para RGB.

A técnica de equalização de histograma provou ser uma ferramenta valiosa para a melhoria do contraste em imagens. Entretanto, ao lidar com imagens coloridas, foi crucial escolher cuidadosamente o espaço de cores em que aplicamos a equalização. Com base em nossa experimentação, concluímos que, ao trabalhar com o espaço de cores YIQ e equalizando apenas o canal de luminância (Y), conseguimos preservar as cores originais da imagem. Ao comparar os resultados de equalização direta nos canais RGB com a abordagem YIQ, a superioridade da abordagem YIQ em manter as cores originais da imagem ficou evidente.

0.5.3 Resultado

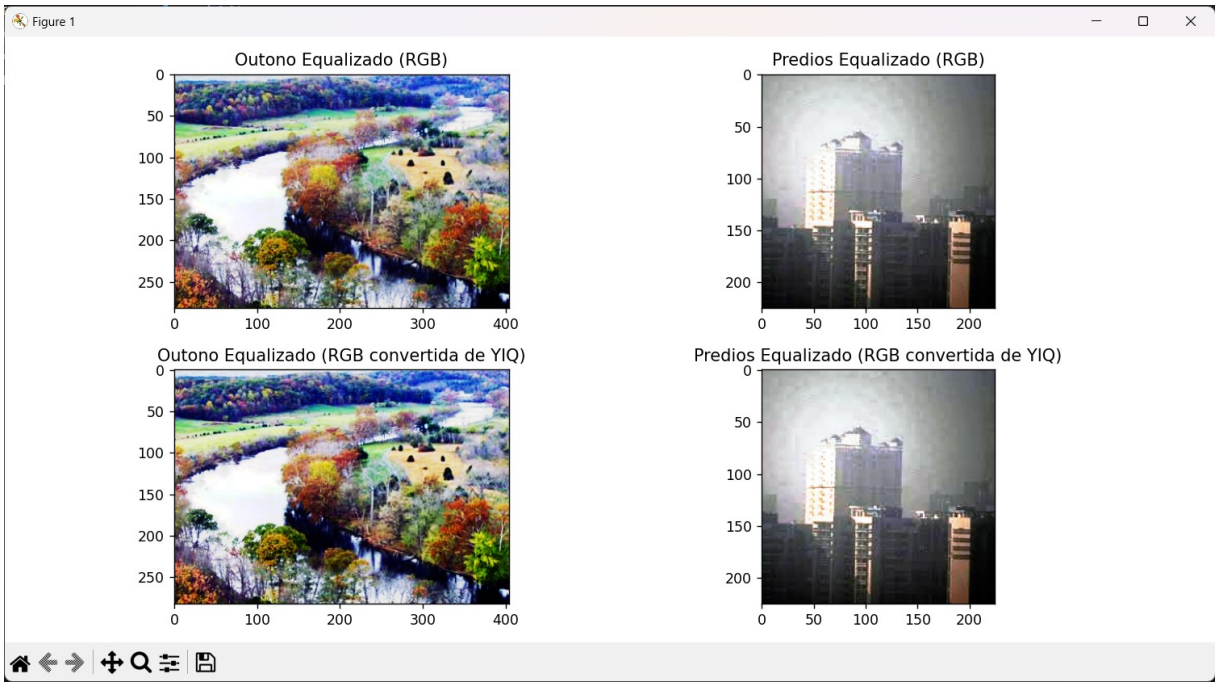

Figura 9: Resultado do código da atividade E.

Conclusões

No decorrer deste projeto prático, aprofundamos significativamente nosso entendimento e competência em diversas técnicas pertinentes à análise e processamento de imagens.

Inicialmente, abordamos a técnica de pseudocor. Esta técnica provou ser essencial ao realçar informações em imagens originalmente monocromáticas. Ao implementar a transformação, conseguimos convertê-la de escala de cinza para uma versão colorida, oferecendo um destaque visual mais evidente. Através desta, não apenas diferenciamos as regiões em variados tons de cinza, mas também destacamos áreas do mapa que representam taxas específicas de roubo de carros, demonstrando a relevância do pseudocor na visualização de informações.

Na sequência, nos dedicamos à equalização de contraste em tons de cinza. Por meio das técnicas de equalização de histograma, tanto global quanto local, observamos uma acentuada melhoria no contraste das imagens XADREZ.png e xadrez_lowCont.png. A análise dos histogramas pré e pós equalização permitiu uma clara visualização das alterações e benefícios proporcionados por tais técnicas.

Ao explorar o sistema RGB, identificamos desafios decorrentes da correlação entre seus canais. A solução adotada foi o emprego do sistema YIQ, caracterizado por canais menos correlacionados. Utilizando as funções providas pelo pacote skimage, executamos a conversão e equalização de imagens com eficácia, focando no canal de luminância (Y) e mantendo intacta a crominância. A subsequente comparação entre os histogramas RGB e YIQ reiterou a eficácia desta abordagem.

Concluindo, este trabalho prático reforçou nossa compreensão sobre os conceitos teóricos discutidos nas aulas e aprofundou nossas habilidades práticas. A experiência adquirida nesse projeto será útil enquanto avançamos no estudo de Processamento de Imagens e em outras disciplinas relacionadas ao longo do curso.