Relatório Trabalho 1 - PIM

Vinícius Hansen

Outubro 2023

1 Questão A

Para aplicar a pseudocor fiz uma engenharia reversa da função jet da biblioteca matplotlib. A primeira função mapeia a nova cor a ser atribuida para um pixel em escala de cinza conforme seu valor numérico e a segunda aplica esse mapa à imagem em si. O resultado pode ser visualizado na figura 1.

```
def jet_color_map(value):
    """
    Implementação simplificada do colormap "jet" para um valor entre 0 e 1.
    """
    if value < 0.125:
        return (0, 0, 4 * value + 0.5)
    elif value < 0.375:
        return (0, 4 * value - 0.5, 1)
    elif value < 0.625:
        return (4 * value - 1.5, 1, -4 * value + 2.5)
    elif value < 0.875:
        return (1, -4 * value + 3.5, 0)
    else:
        return (-4 * value + 4.5, 0, 0)

def plot_jet_colored_image(image):
    """
    Aplica o colormap "jet" à imagem em escala de cinza e plota a imagem colorida.
    """
    image_np = np.array(image)

    # Normaliza a imagem para valores entre 0 e 1
    normalized_image = image_np / 255.0

    # Aplica o colormap "jet" à imagem
    jet_colored_image = np.zeros((image_np.shape[0], image_np.shape[1], 3))
    for i in range(image_np.shape[0]):
        for j in range(image_np.shape[1]):
            if image_np[i, j] != 255:  # Verifica se o pixel não é totalmente branco
                jet_colored_image[i, j] = jet_color_map(normalized_image[i, j])
            else:
                jet_colored_image[i, j] = [1, 1, 1]  # Mantém a cor branca original (para o fundo)
```

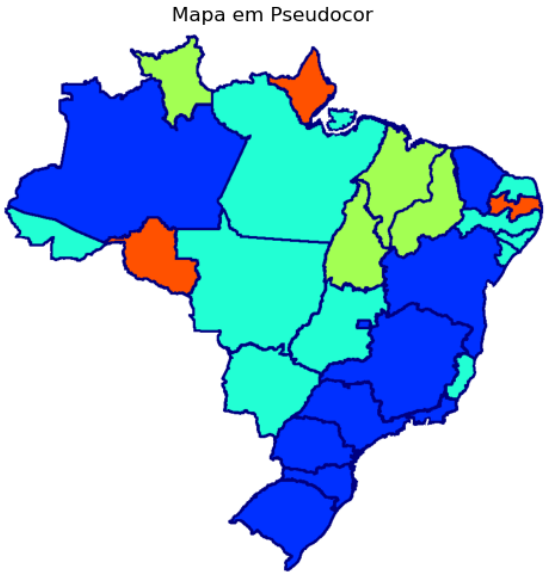

Figure 1: Mapa em Pseudo Cor

2 Questão B

Nessa questão, criei uma máscara que destaca a região desejada e a pintei de vermelho, inclui uma condição para ela não considerar o fundo, que seriam os pixels totalmente brancos. a figura 2 demonstra o resultado. Segue o código utilizado.

```
def highlight_regions_by_theft_rate_corrected(theft_rate, mapa_array, min_rate=0, max_rate=300
# Normalizar a taxa de roubo para o intervalo [0, 255]
normalized_theft_rate = 255 * (theft_rate - min_rate) / (max_rate - min_rate)

# Criar uma máscara para as regiões que correspondem à taxa de roubo
mask = (np.abs(mapa_array - normalized_theft_rate) < 5) & (mapa_array != 255) # não é branco(f

# Criar uma imagem RGB
highlighted_image = np.stack([mapa_array] * 3, axis=-1)

# Destacar as regiões em vermelho
highlighted_image[mask] = [255, 0, 0]

return highlighted_image
```

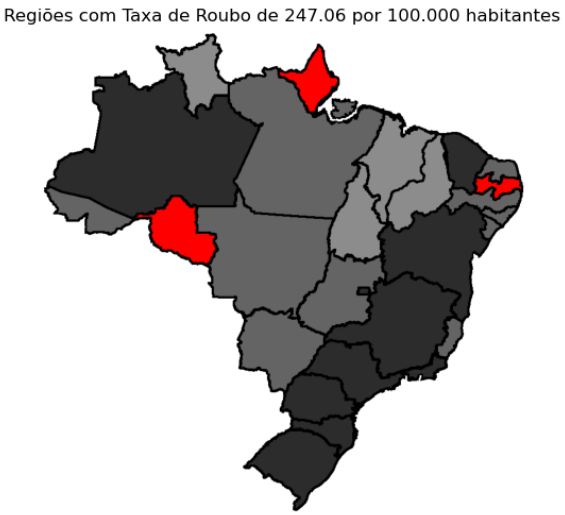

Figure 2: Mapa com regiões destacadas por valor

3 Questão C

Aqui implementei a equalização de histograma tanto local quanto global. Utilizei o conceito de arrays mascarados para evitar problemas com os valores iguais a zero. Pode-se analisar os resultados nas Figuras 3, 3, 5 e 6.

```
def equalize_histogram_global(image):
"""Equalização de histograma global."""
hist, bins = np.histogram(image.flatten(), 256, [0,256])
cdf = hist.cumsum() # soma cumulativa
cdf_m = np.ma.masked_equal(cdf, 0) #mascara para os valores 0
cdf_m = (cdf_m - cdf_m.min()) * 255 / (cdf_m.max() - cdf_m.min()) # equalização em si
cdf = np.ma.filled(cdf_m, 0).astype('uint8') # coloca os zeros de volta
return cdf[image]

def equalize_histogram_local(image, m=4):
"""Equalização de histograma local."""
h, w = image.shape
pad = m // 2 # borda
padded_image = np.pad(image, ((pad, pad), (pad, pad)), mode='constant',constant_values=(0,0))
output = np.zeros_like(image) # Cria imagem de saída
for i in range(h):
    for j in range(w):
        window = padded_image[i:i+m, j:j+m] # Janela deslizante
        hist, _ = np.histogram(window, 256, [0,256])
        cdf = hist.cumsum() # soma cumulativa
        cdf_m = np.ma.masked_equal(cdf, 0) # mascara de zeros
        cdf_m = (cdf_m - cdf_m.min()) * 255 / (cdf_m.max() - cdf_m.min()) #equalização em si
        cdf = np.ma.filled(cdf_m, 0).astype('uint8') # bota os zeros de volta
        output[i, j] = cdf[window[pad, pad]]
return output
```

Nota-se uma grande quantidade de artefatos na figura 4 que representa a equalização local da imagem Xadrez.png

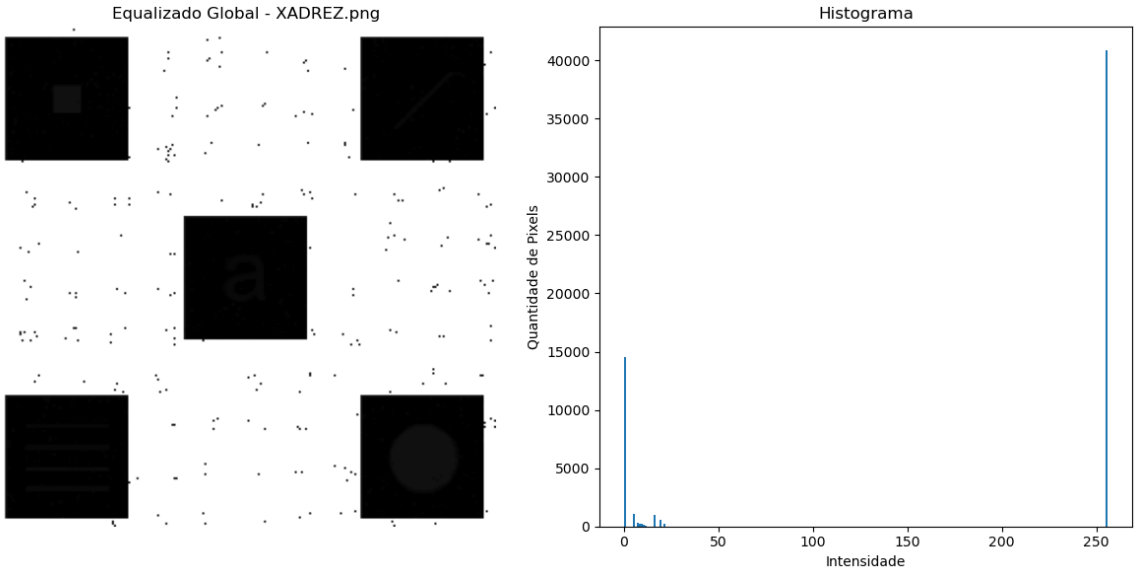

Figure 3: Equalização global da imagem Xadrez.png

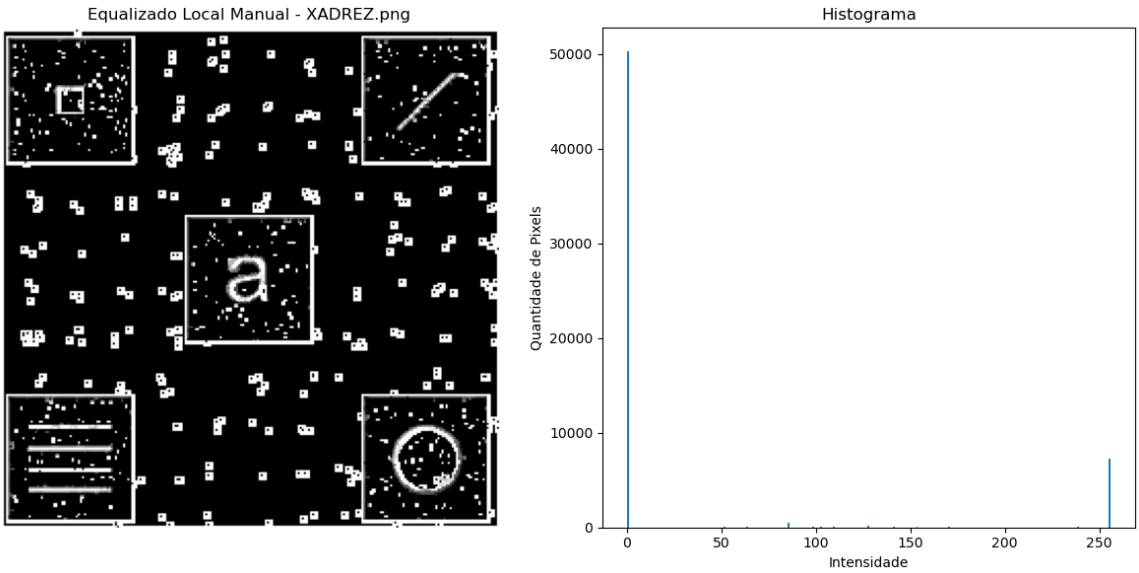

Figure 4: Equalização local da imagem Xadrez.png

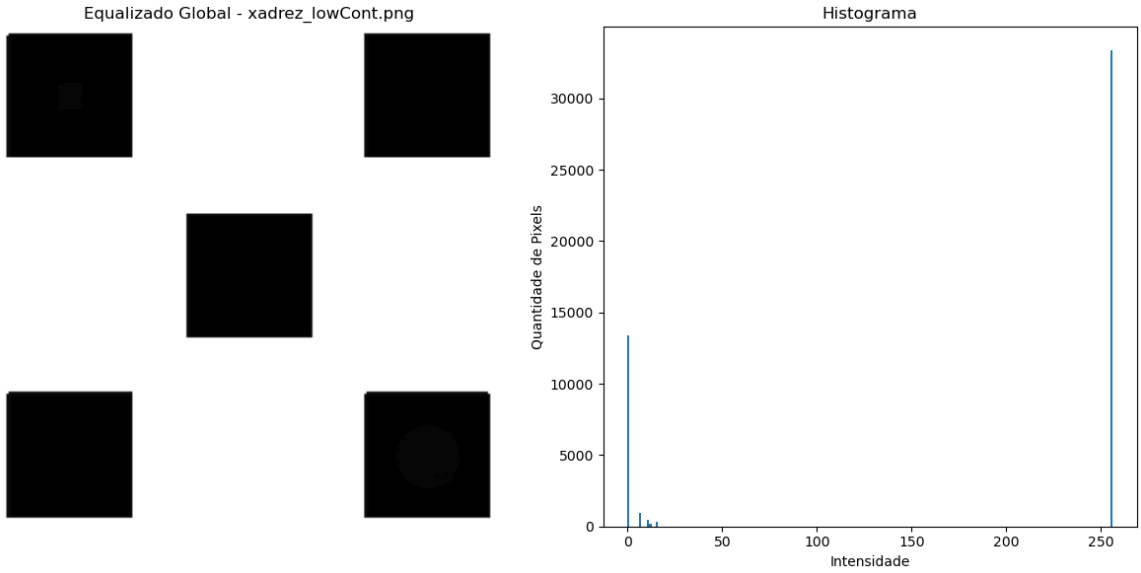

Figure 5: Equalização global da imagem xadrez lowCont

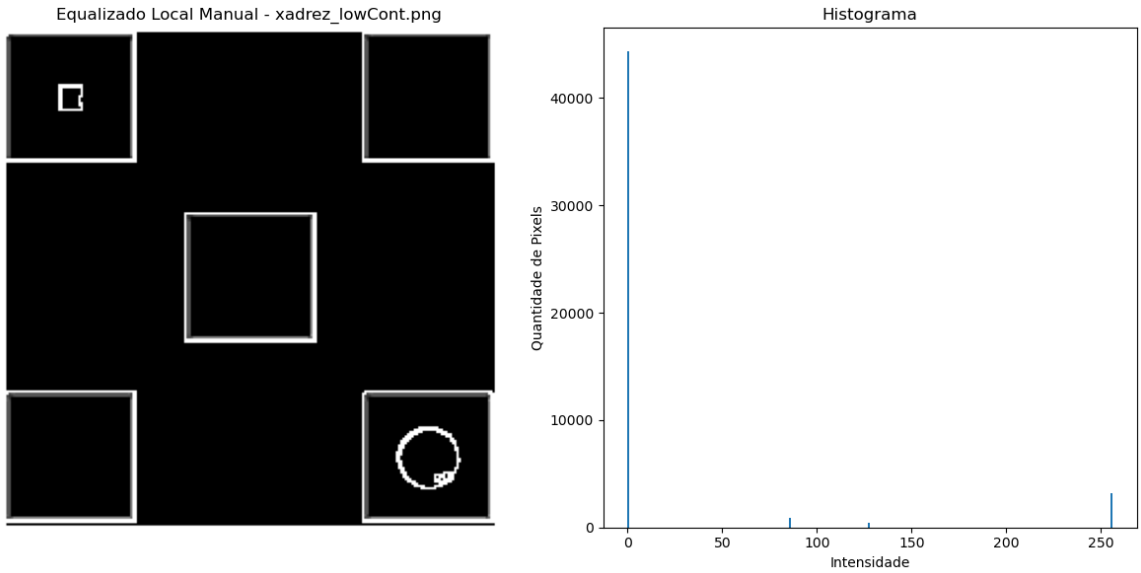

Figure 6: Equalização local da imagem xadrez lowCont

4 Questão D

Para fazer um "matching" do histograma da imagem com um histograma fornecido foi utilizado o método da soma cumulativa no histograma fornecido seugido por uma normalização do mesmo e um mapeamento no histograma original. Para a aplicação prática foi escolhido um histograma uniforme como histograma fornecido. Os resultados podem ser observado nas Figuras 7 e 8.

```
def histogram_matching(img, specified_hist):
    """Mathing do histograma."""
    hist, _ = np.histogram(img.flatten(), 256, [0,256])
    cdf_img = hist.cumsum()
    cdf_img_normalized = cdf_img / cdf_img[-1]

    # Calcule o histograma cumulativo do histograma especificado
    cdf_specified = specified_hist.cumsum()
    cdf_specified_normalized = cdf_specified / cdf_specified[-1] # cdf_specified[-1] é o último va

    mapping = np.zeros(256)
    for i in range(256):
        diff = np.abs(cdf_img_normalized[i] - cdf_specified_normalized)
        mapping[i] = np.argmin(diff)

    # Aplicar mapeamento à imagem
    output = mapping[img]
    return output

# Gerar um histograma uniforme
specified_hist = np.ones(256) * (xadrez_img_array.size / 256)
```

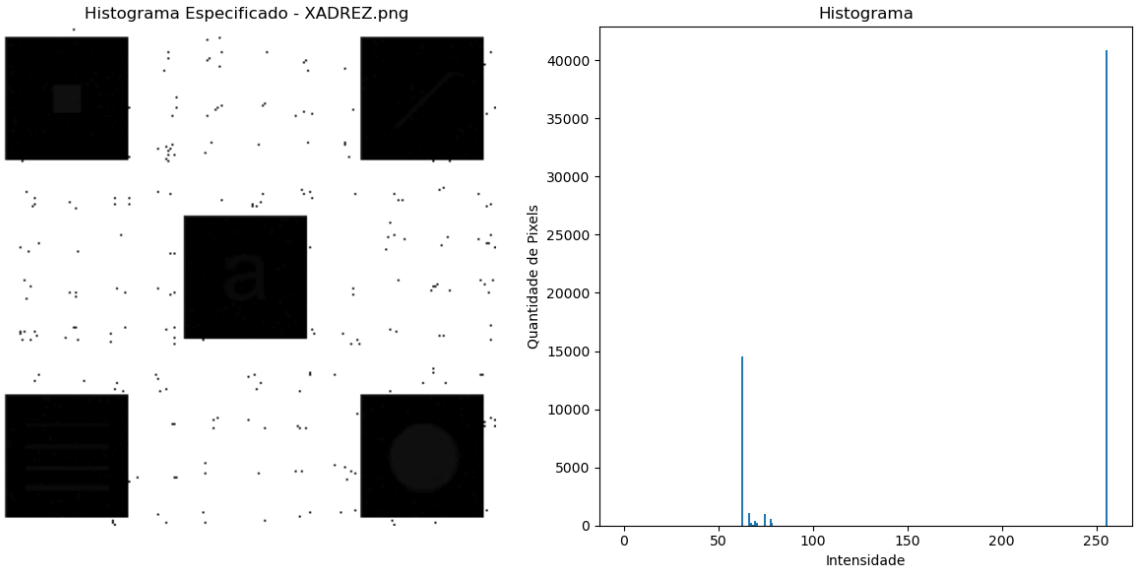

Figure 7: Matching da imagem Xadrez.png

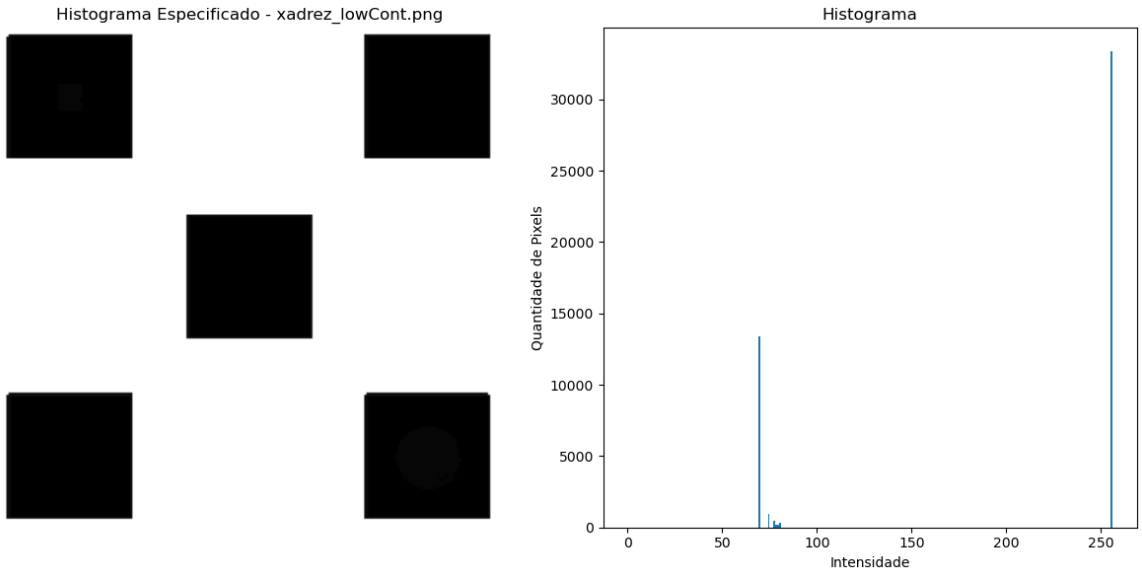

Figure 8: Matching da imagem xadrez lowCont.png

5 Questão E

Na primeira função foi utilizado o mesmo conceito de máscaras para a equalização de histograma, já a segunda função aplica o método aos canais RGB e ao canal Y do sistema de cores YIQ. Os resultados podem ser vistos nas Figuras 9 e 10.

```
def equalize_hist_channel(channel):
    """Equaliza um Canal."""
    hist, bins = np.histogram(channel.flatten(), 256, [0,1])
    cdf = hist.cumsum()
    if cdf[-1] == 0:
        return channel
    cdf_m = np.ma.masked_equal(cdf, 0) # Máscara de zeros
    cdf_m = (cdf_m - cdf_m.min()) * (1.0 / (cdf_m.max() - cdf_m.min())) # Equalização em si
    cdf = np.ma.filled(cdf_m, 0).astype(np.float64) # Devolve os zeros
    # Convertendo os valores de pixel em channel para inteiros antes de usá-los como índices
    channel_int = (channel * 255).astype(int)
    return cdf[channel_int].reshape(channel.shape)

def equalize_rgb_and_yiq_without_skimage(image_path):
    """Aplica a equalização por canal."""
    image = plt.imread(image_path)
    # RGB
    r_eq = equalize_hist_channel(image[:, :, 0])
    g_eq = equalize_hist_channel(image[:, :, 1])
    b_eq = equalize_hist_channel(image[:, :, 2])
    rgb_eq = np.stack([r_eq, g_eq, b_eq], axis=2)
    # YIQ
    yiq = rgb2yiq(image)
    y_eq = equalize_hist_channel(yiq[:, :, 0])
    yiq[:, :, 0] = y_eq
    yiq2rgb_eq = yiq2rgb(yiq)
    yiq2rgb_eq = np.clip(yiq2rgb_eq, 0, 1)
    return rgb_eq, yiq2rgb_eq
```

Figure 9: Equalizações da imagem outono LC.png

Figure 10: Equalizações da imagem predios.jpeg