Processos estocásticos - turma 1/2024
Universidade de Brasília

Trabalho computacional 1 - Requantização e análise de imagens

Gabriel Tambara Rabelo - 241106461

Referências:
https://docs.opencv.org/4.x/d1/db7/tutorial_py_histogram_begins.html
https://docs.opencv.org/4.9.0/d4/d1b/tutorial_histogram_equalization.html
https://docs.opencv.org/3.4/d1/d5c/tutorial_py_kmeans_opencv.html

Procedimento e imagens

O presente documento retrata o processo de quantização de 5 imagens sortidas obtidas para os fins de aprendizado. Para tal, utiliza-se das ferramentas dispostas pela linguagem de programação Python, versão 3.11, e das bibliotecas numpy, para tratamento facilitado de matrizes, matplotlitb, para melhor visualização de imagens e gráficos e opencv2 para tratamento e análise de imagens. Para melhor visualização e compreensão do programa, têm-se o código demonstrado seguido pelos resultados em imagens.

Para a inicialização do código, é feito o que se apresenta:

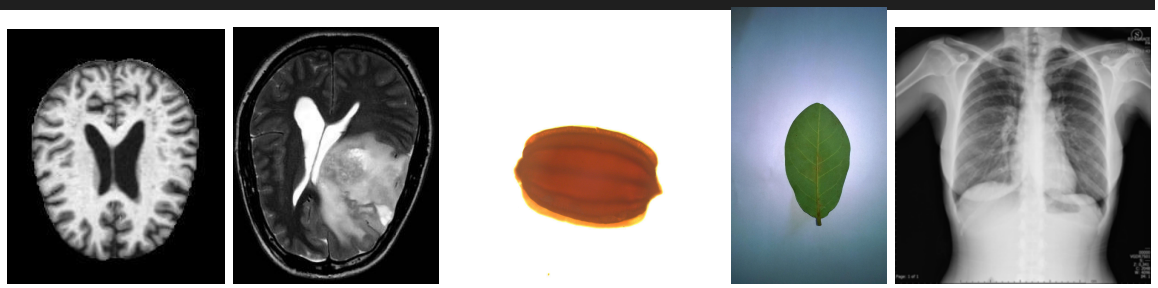
```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img_size = 300
images_addr = ["folha.jpg", "torso.png", "coisa.jpg", "cerebro.jpg",
"cerebro-tumor.jpg"]
images_addr.sort()
images = []
```

As cinco imagens definidas foram nomeadas como se apresenta e organizadas em uma pasta no mesmo diretório deste arquivo jupyter notebooks, uma extensão da linguagem Python para facilitação de leitura e visualização de resultados e documentação.

```
# control of when to stop showing images and progressing in the processing
def waitKey():
    cv.waitKey(0)
    cv.destroyAllWindows()

# show images
def printAll(list, subtitle):
    for i in range(len(list)):
        cv.imshow(images_addr[i] + subtitle, list[i])

# save images in the correct folder
def saveAll(list, subtitle):
    for i in range(len(list)):
        cv.imwrite("./images/" + images_addr[i][0:-4] + subtitle +
images_addr[i][-4:], list[i])
```

Foram desenvolvidas algumas funções para controle de fluxo de dados e para melhor visualização dos resultados quando executado fora de um ambiente jupyter, como se apresentou na imagem anterior. Também foram desenvolvidas funções para modularizar o processo de salvamento e coleta de dados.

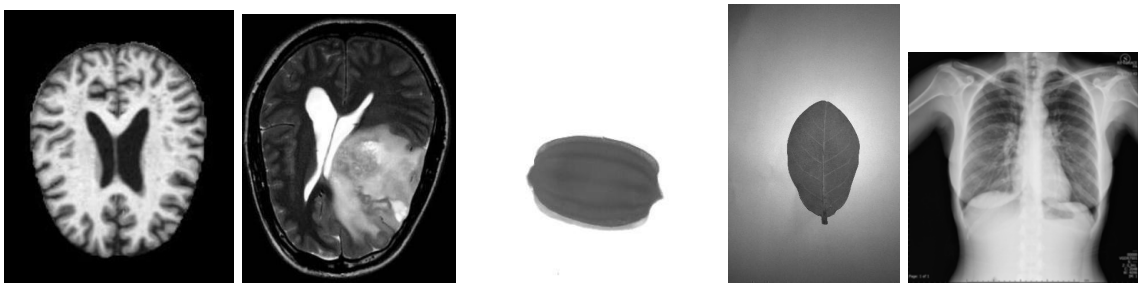
```
# reading and viewing images in the folder

for img in images_addr:
    images.append((cv.imread("./images/" + img)))
    ratio = img_size / images[-1].shape[1]
    images[-1] = cv.resize(images[-1], (img_size, int(images[-1].shape[0]
* ratio)), cv.INTER_AREA)

saveAll(images, '')
waitKey()
```



Aqui as imagens foram importadas e apresentadas pelas referidas bibliotecas e código apresentado. Aqui utiliza-se da variável ratio para garantir que as imagens mantenham-se na

mesma proporção da primeira imagem, já que umas eram bem maiores que outras, facilitando assim a visualização. Após isso, Deseja-se tratar apenas do canal de cores preto e branco das imagens, para que possamos analisar apenas uma variável e não as três usuais dos canais de cores RGB. Para isso, segue:

```
# converting images to gray-scale format

images_bw = []
for i in range(len(images)):
    images_bw.append(cv.cvtColor(images[i], cv.COLOR_BGR2GRAY))

saveAll(images_bw, '')
histList = makeHist(images_bw, 8)
showHist(images_bw, histList, 'grey', 16, True)
waitKey()
```



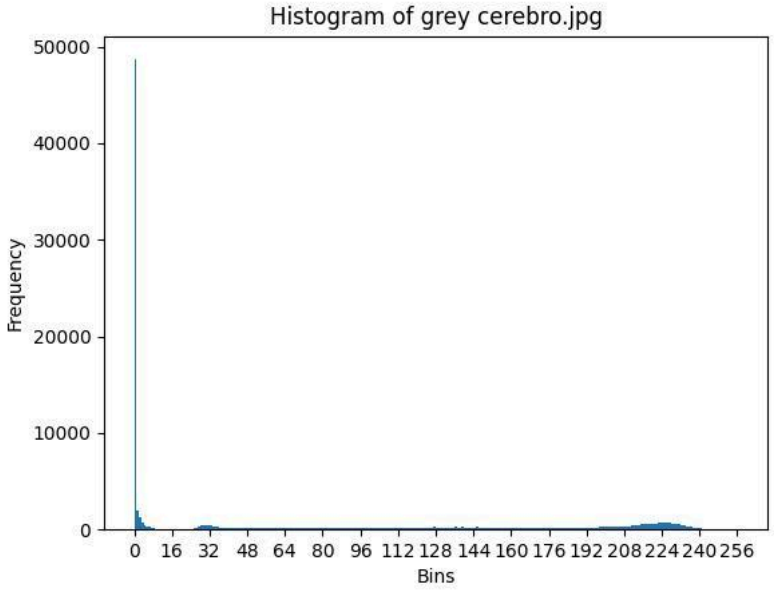
Também são definidas funções para criar histogramas a partir dos dados obtidos de modo que possamos visualizar as imagens dentro de um eixo apenas, como se segue:

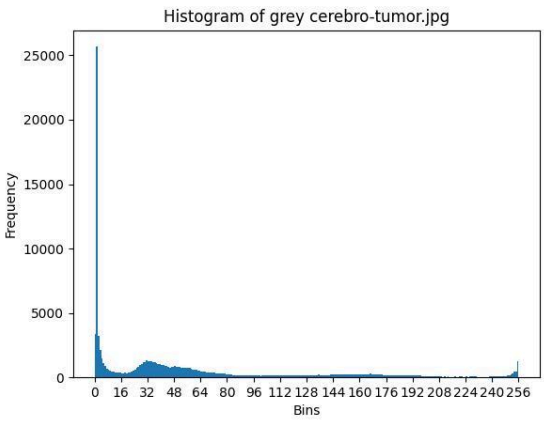
```
def makeHist(image_list, bits):
    histList = []
    for i in range(len(image_list)):
        hist = []
        bins = np.arange(0, 257, 2 ** (8 - bits)) #257 since the bins must
be 1 more than what it shows and we need even 1 more since arange excludes
the last one

        if(bits <= 0 or bits > 8):
            print("bitsize our of bounds")
            exit()

        image = image_list[i].flatten()
```

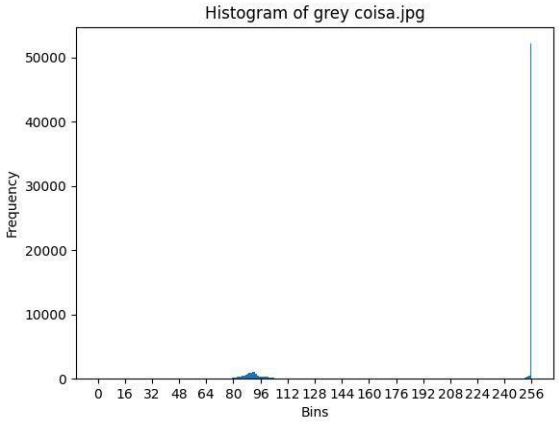
```
        #for pixel_intensity in bins:
        #    hist.append(np.sum(image == pixel_intensity))
        #    print(str(hist[-1]))
        hist, bins_new = np.histogram(image, bins=bins)

        histList.append((hist, bins))
    return histList

def normalizeHist(histList):
    normalizedHistList = []
    for hist, bins in histList:
        total_pixels = np.sum(hist)
        normalized_hist = hist / (total_pixels)
        normalizedHistList.append((normalized_hist, bins))
    return normalizedHistList

def showHist(image_list, histList, category, limit_bins=None,
limit_overtext=True):
    for i in range(len(image_list)):
        hist = histList[i][0]
        bins = histList[i][1]

        if limit_bins is not None:
            plt.xticks(bins[::len(bins)//limit_bins],
bins[::len(bins)//limit_bins])
        else:
            plt.xticks(bins)

        plt.bar(bins[:-1], hist, width=np.diff(bins), align='edge')

        if limit_overtext == False:
            for j in range(len(hist)):
                plt.text(bins[j], hist[j], str(hist[j]), ha='center',
va='bottom', rotation=30)

        plt.xlabel('Bins')
        plt.ylabel('Frequency')
        plt.title('Histogram of ' + category + ' ' + images_addr[i][0:-4]
+ images_addr[i][-4:])
```

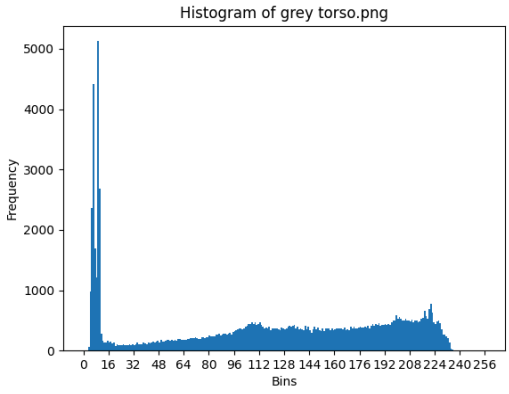
```
        plt.savefig("./images/hist_" + category + images_addr[i][0:-4] +
images_addr[i][-4:])
        plt.show()
```

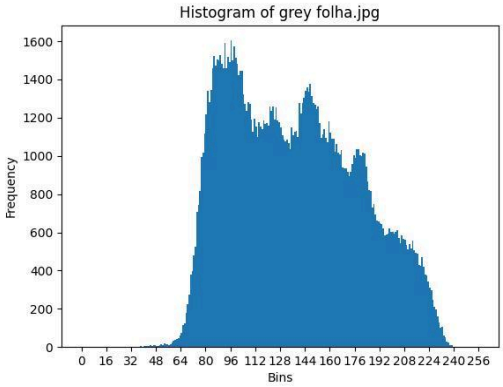
Para o caso das imagens em preto e branco temos os seguintes histogramas gerados, seguidos pela sua versão normalizada:





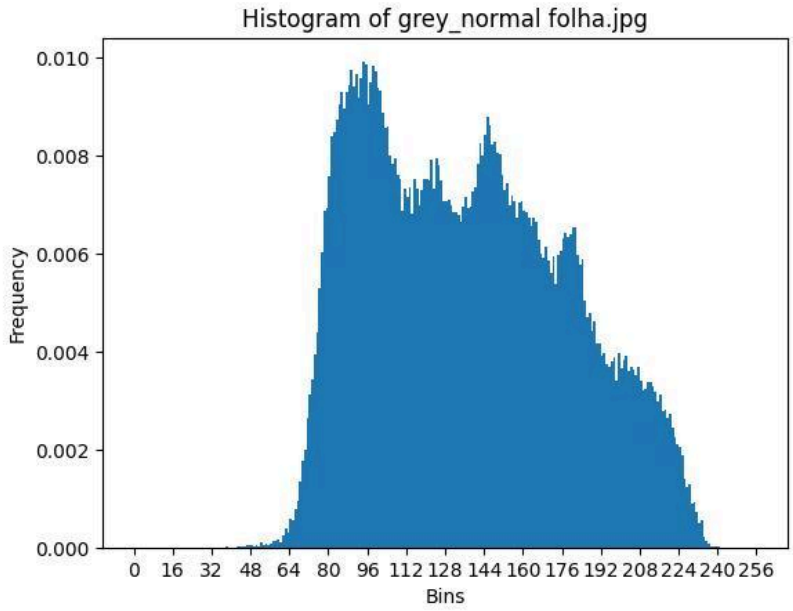






Percebe-se a distribuição de cores diferentes em cada imagem bem diferentes, e condizentes com o perfil apresentado nas imagens. Imagens mais escuras vão ter concentração de dados pendendo para um lado, ou um skewness mais acentuado para a esquerda, e imagens mais neutras terão uma distribuição mais próxima da normal.

Para suas versões normalizadas, segue um exemplo em que pode-se perceber que a soma de seus elementos resulta em 1 pela razoabilidade de suas amplitudes.

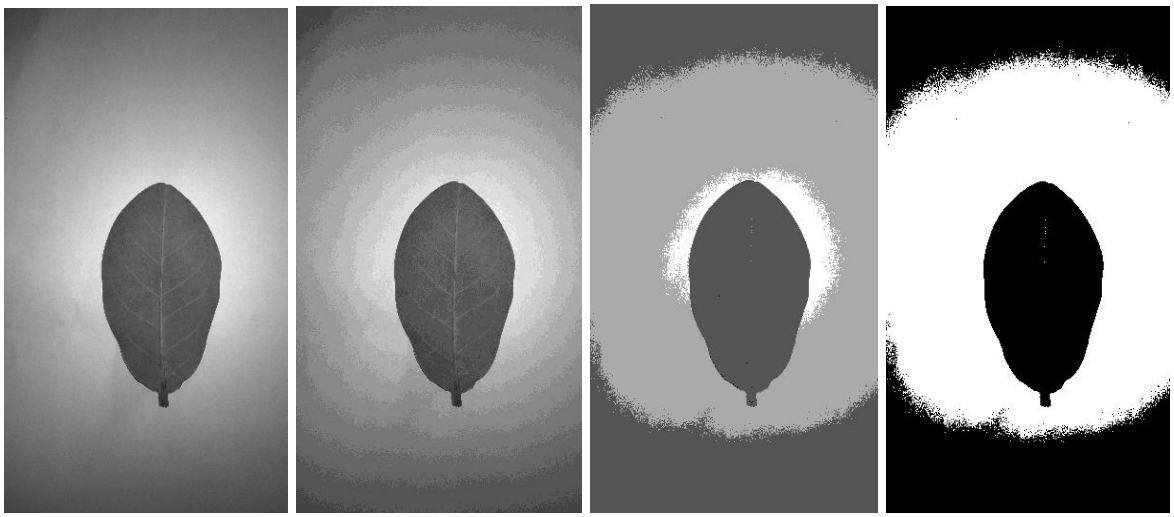


Agora, para aprofundar a análise, deseja-se realizar uma requantização das imagens para observar como seus formatos e contornos permanecem quando reduzimos a resolução em pixels disponíveis para quantizar as informações nelas contidas. A seguinte função tem por objetivo realizar este processo sem alterar o formato e o tamanho das imagens.

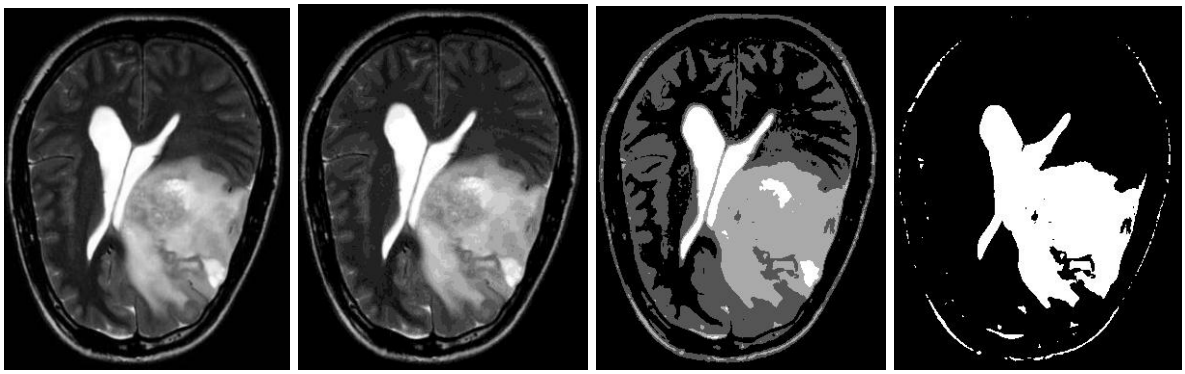
```
def requantization(input, bits):
    images_formatted = []

    def normalize(x):
        return x/255

    def denormalize(x):
        return (int)(255*x)

    def quantize(x):
        return denormalize((np.round((2**bits -1) *
normalize(x)))/(2**bits -1))

    for image in input:
        images_formatted.append(np.vectorize(quantize)(image))
    return images_formatted
```

Agora, para avançar na análise, apresenta-se a versão para quantizar as imagens e gerar suas respectivas versões de menor resolução. De início foi realizada a quantização de 4 bits de resolução, portanto, resultando em 8 tons/cores diferentes. Depois com 2 e depois com 1 bit. Seguem dois exemplos de imagens que passaram pelo processo.

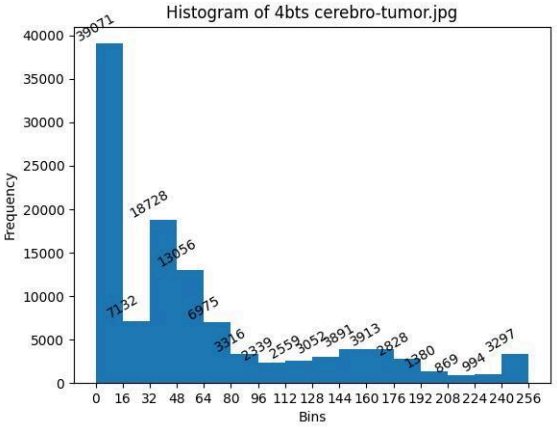
Para a primeira, tem-se a folha sob superfície plana. Neste caso, quanto mais reduzimos a resolução, mais fácil se torna observar seu contorno, contudo, observa-se o fenômeno de falsos contornos sendo criados pela transição aproximada de cores resultante da diferença de iluminação dos contornos da imagem com o centro.

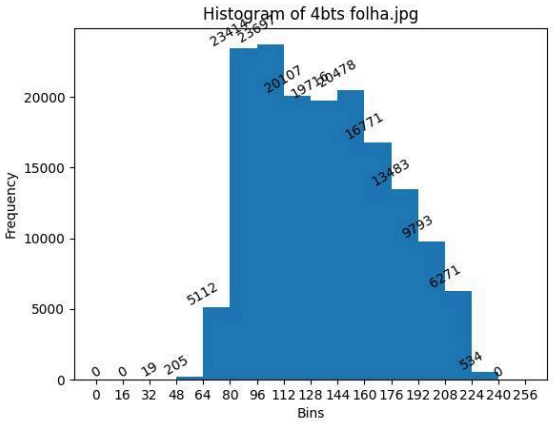


Para o exemplo seguinte, observa-se uma fatia de radiografia de um crânio com tumor. Neste cenário, fica evidente a facilitação da visualização dos contornos do elemento tumoral.

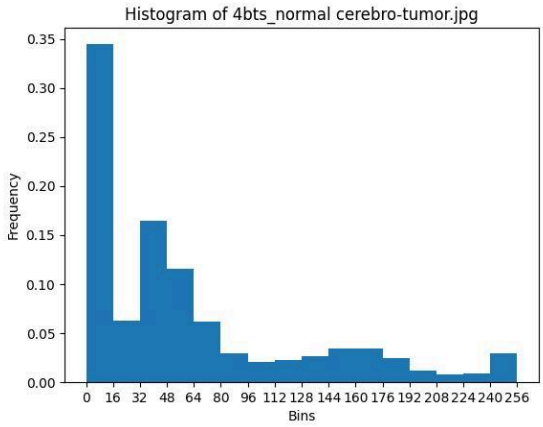


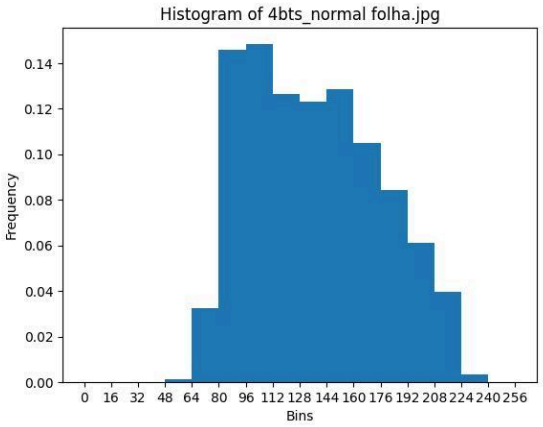
A seguir, temos a apresentação de dois histogramas quantizados em 4 bits, de ambas as imagens. Percebe-se a distribuição semelhante à apresentada previamente, contudo, com menor resolução e um número limitado de ‘bins’, neste caso, de 2 elevado a 4, ou seja, 16 ‘bins’.



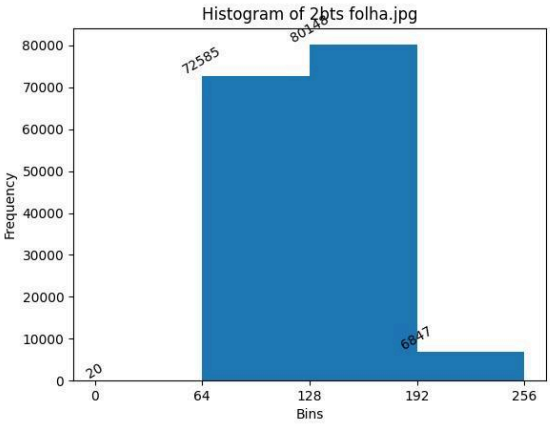


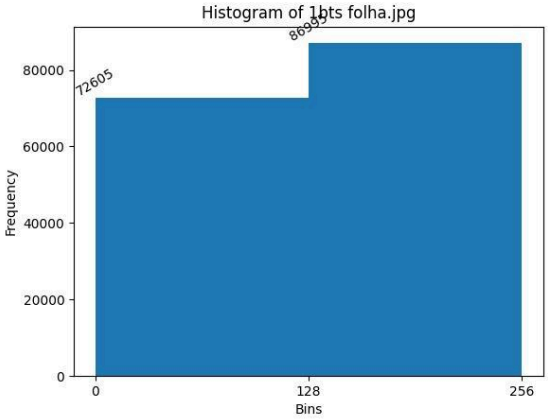
Os mesmos histogramas são apresentados em suas versões normalizadas:



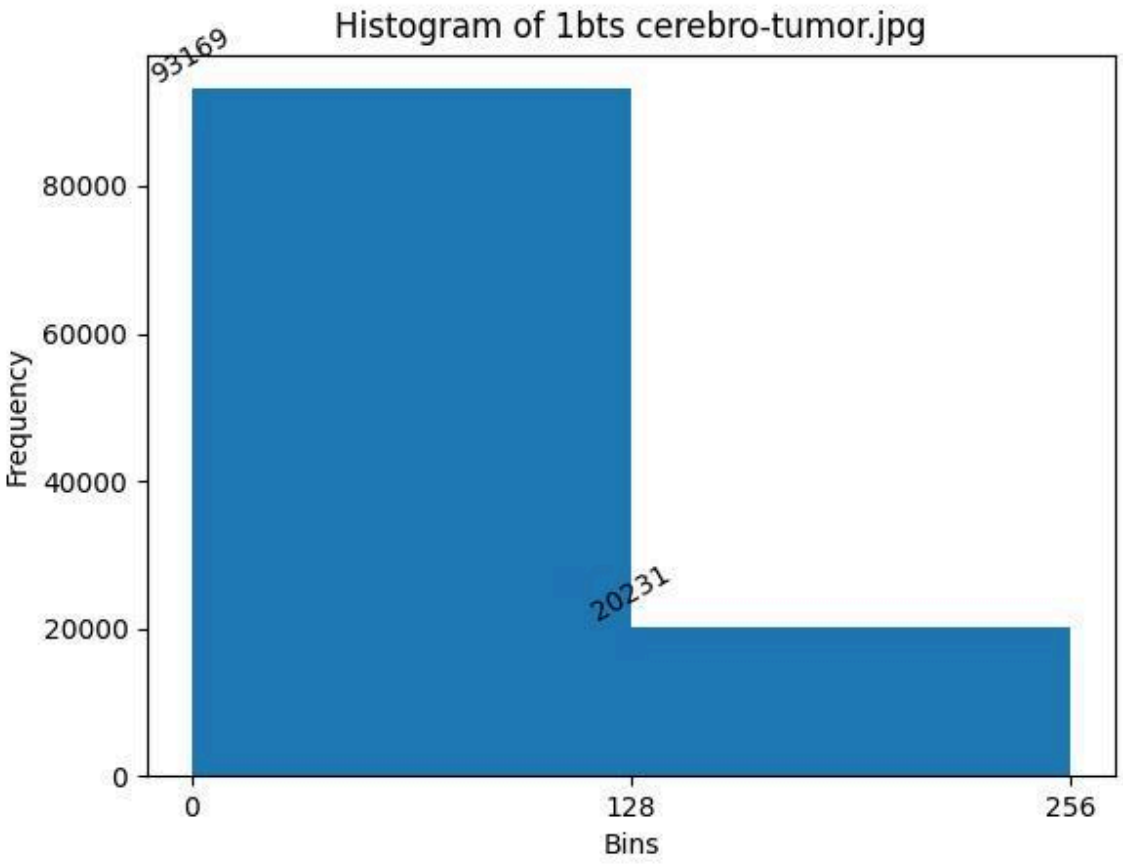


As versões em 2 bits e 1 bit da imagem da folha são apresentadas:





Aqui percebe-se que há uma convergência dos valores para os grupos maiores, o que está de acordo com o processo de quantização utilizado, que aproxima os pixels para os bins de cor mais próxima, ultimamente levando a um cenário com pixels em preto e branco. No caso da folha, percebe-se uma paridade entre as cores tendo em vista sua distribuição mais suave. Podemos comparar com a seguinte imagem que retrata o caso de 1 bit para o cérebro com o tumor em que a maioria se localiza com ton 0 e a minoria com 255, o valor máximo.



Conclusão e resumo

Foram realizados processos de coleta de imagens, quantização e tratamento das mesmas para observar os efeitos de quantização nas mesmas. É evidente a perda de detalhes, contudo, a facilitação da observação de certos padrões de tendência da imagem como um todo, ou seja, dos contornos. Também observa-se a criação de falsos contornos, podendo levar a interpretações equivocadas caso não se considere a imagem original para análise. Ademais, também observou-se através da criação de histogramas dos dados gerados, que as distribuições se assemelham de normais quando normalizadas, tendo em vista o caráter de que sua integral retorna o valor 1 e seus valores de frequência para bins maiores que 256 e menores são sempre zero.

Código fonte

```
# %% [markdown]
# #
# #   Stochastic Processes - class 1/2024 - University of Brasília
# #   Computational work 1 - Requantization and image analysis
# #
# #   Gabriel Tambara Rabelo - 241106461
# #
# #   References:
# #   https://docs.opencv.org/4.x/d1/db7/tutorial_py_histogram_begins.html
# #
https://docs.opencv.org/4.9.0/d4/d1b/tutorial_histogram_equalization.html
# #   https://docs.opencv.org/3.4/d1/d5c/tutorial_py_kmeans_opencv.html
# #

# %%

import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img_size = 300
images_addr = ["folha.jpg", "torso.png", "coisa.jpg", "cerebro.jpg",
"cerebro-tumor.jpg"]
images_addr.sort()
```

```
images = []

# control of when to stop showing images and progressing in the processing
def waitKey():
    cv.waitKey(0)
    cv.destroyAllWindows()

# show images
def printAll(list, subtitle):
    for i in range(len(list)):
        cv.imshow(images_addr[i] + subtitle, list[i])

# save images in the correct folder
def saveAll(list, subtitle):
    for i in range(len(list)):
        cv.imwrite("./images/" + images_addr[i][0:-4] + subtitle +
images_addr[i][-4:], list[i])

def makeHist(image_list, bits):
    histList = []
    for i in range(len(image_list)):
        hist = []
        bins = np.arange(0, 257, 2 ** (8 - bits)) #257 since the bins must
be 1 more than what it shows and we need even 1 more since arange excludes
the last one

        if(bits <= 0 or bits > 8):
            print("bitsize our of bounds")
            exit()

        image = image_list[i].flatten()
        #for pixel_intensity in bins:
        #    hist.append(np.sum(image == pixel_intensity))
        #    print(str(hist[-1]))
        hist, bins_new = np.histogram(image, bins=bins)

        histList.append((hist, bins))
    return histList

def normalizeHist(histList):
```

```
    normalizedHistList = []
    for hist, bins in histList:
        total_pixels = np.sum(hist)
        normalized_hist = hist / (total_pixels)
        normalizedHistList.append((normalized_hist, bins))
    return normalizedHistList

def showHist(image_list, histList, category, limit_bins=None,
limit_overtext=True):
    for i in range(len(image_list)):
        hist = histList[i][0]
        bins = histList[i][1]

        if limit_bins is not None:
            plt.xticks(bins[::len(bins)//limit_bins],
bins[::len(bins)//limit_bins])
        else:
            plt.xticks(bins)

        plt.bar(bins[:-1], hist, width=np.diff(bins), align='edge')

        if limit_overtext == False:
            for j in range(len(hist)):
                plt.text(bins[j], hist[j], str(hist[j]), ha='center',
va='bottom', rotation=30)

        plt.xlabel('Bins')
        plt.ylabel('Frequency')
        plt.title('Histogram of ' + category + ' ' + images_addr[i][0:-4]
+ images_addr[i][-4:])
        plt.savefig("./images/hist_" + category + ' ' +
images_addr[i][0:-4] + images_addr[i][-4:])
        plt.show()

# gerenate bits colored resolution image data from input with kmeans
def kmeans(input, bits):
    images_formatted = []
    criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 10, 1.0)

    for i in range(len(input)):
```

```
        images_formatted.append(np.float32(input[i]).reshape(-1, 3))
        compactness, labels, center = cv.kmeans(images_formatted[i],
2**bits, None, criteria, 10, cv.KMEANS_RANDOM_CENTERS)
        center = np.uint8(center)
        final_img = center[labels.flatten()]
        images_formatted[i] = final_img.reshape(input[i].shape)

    return images_formatted

def requantization(input, bits):
    images_formatted = []

    def normalize(x):
        return x/255

    def denormalize(x):
        return (int)(255*x)

    def quantize(x):
        return denormalize((np.round((2**bits -1) *
normalize(x)))/(2**bits -1))

    for image in input:
        images_formatted.append(np.vectorize(quantize)(image))
    return images_formatted

# %%
# reading and viewing images in the folder

for img in images_addr:
    images.append((cv.imread("./images/" + img)))
    ratio = img_size / images[-1].shape[1]
    images[-1] = cv.resize(images[-1], (img_size, int(images[-1].shape[0]
* ratio)), cv.INTER_AREA)

saveAll(images, '')
waitKey()

# %%
# converting images to gray-scale format
```

```
images_bw = []
for i in range(len(images)):
    images_bw.append(cv.cvtColor(images[i], cv.COLOR_BGR2GRAY))

saveAll(images_bw, '')
histList = makeHist(images_bw, 8)
showHist(images_bw, histList, 'grey', 16, True)
waitKey()

# %%
histList = normalizeHist(histList)
showHist(images_bw, histList, 'grey_normal', 16, True)
waitKey()

# %%
# using quantization with K-MEANS to reduce colors in images to 4 bits of
resolution

images_formatted = requantization(images_bw, 4)
saveAll(images_formatted, " in 4 bits")
histList = makeHist(images_formatted, 4)
showHist(images_bw, histList, '4bts', 16, False)
waitKey()

# %%
histList = normalizeHist(histList)
showHist(images_bw, histList, '4bts_normal', 16, True)
waitKey()

# %%
# repeating for 2 bits of resolution

images_formatted = requantization(images_bw, 2)
saveAll(images_formatted, " in 2 bits")
histList = makeHist(images_formatted, 2)
showHist(images_bw, histList, '2bts', 4, False)
waitKey()

# %%
```

```
histList = normalizeHist(histList)
showHist(images_bw, histList, '2bts_normal', 4, True)
waitKey()

# %%
# repeating for 1 bit of resolution

images_formatted = requantization(images_bw, 1)
saveAll(images_formatted, " in 1 bit")
histList = makeHist(images_formatted, 1)
showHist(images_bw, histList, '1bts', 2, False)
waitKey()

# %%
histList = normalizeHist(histList)
showHist(images_bw, histList, '1nts_normal', 2, True)
waitKey()
```