PROCESSAMENTO DIGITAL DE IMAGENS - TRABALHO 1

Bruna Medeiros da Silva - 16/0048711

UnB - FGA

1. QUESTÃO A

Importando a imagem e utilizando o atributo *shape*, foi verificado que a imagem é uma imagem de tamanho 512 x 512.

Utilizando os métodos .max() e .min(), notou-se que os valores da imagem estãao sim dentro do limite de representação de 8 pixels ($2^8 = 255$), possuindo valores máximo e mínimo, respectivamente, de 241 e 36.

Esse desenvolvimento pode ser visualizado no código 1.

2. QUESTÃO B

Figura original.

Esse resultado foi obtido implementando o código 2.

3. OUESTÃO C

Fig. 1. Versões da imagem com 512x512, 256x256, 128x128, 64x64 e 32x32 pixels

Imagem obtida pela execução do código 3.

4. QUESTÃO D

Nearest Neighbours

Método usado para interpolação, regreção e classificação que se utiliza da teoria de que amostras espacialmente próximas tendem a conter informações parecidas.

Com isso, o método de interpolação dos k vizinhos mais próximos em imagens 2D utiliza o valor dos k pixels mais próximos (com uma menor distância espacial) para definir o valor do pixel em questão.

Essa distância utilizada no cálculo é a distância euclidiana.

Recomenda-se utilizar valores de k **ímpares**, para evitar ”empates” no momento de definir o valor adequado. No geral, são utilizados valores de $k = \sqrt{N}$ ou $k = \log N$, onde N é o número de amostras que você possui.

5. QUESTÃO E

Realizando o procedimento descrito na questão anterior, obteve-se os seguintes resultados:

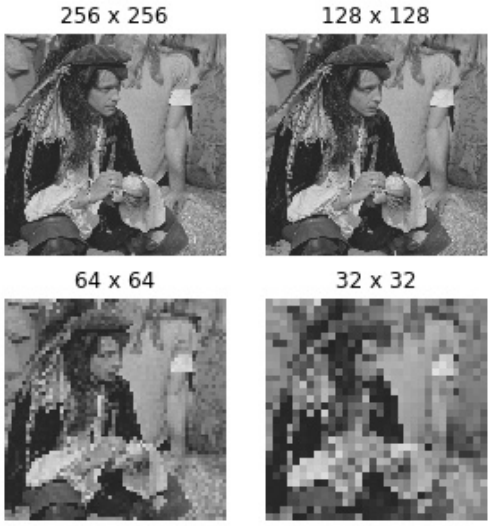

Fig. 2. Imagem reconstruídas utilizando a técnica do KNN.

Figuras reconstruídas utilizando o código 4

6. QUESTÃO F

Interpolação bilinear

Essa forma de interpolação pode ser aplicada em imagens (ou array) de duas dimensões (como x e y, por exemplo)

Para aplicar essa técnica são feitas duas interpolações lineares em sequência: aplica-se primeiro em uma das dimensões e, no resultado obtido, aplica-se na outra dimensão.

Interpolação linear

A interpolação linear utiliza, se certa forma, uma média ponderada dos valores dos pixels mais próximos. Os pesos dessa média serão definidos pela distância total entre esses pixels.

Voltando à interpolação bilinear

Considerando que o processo visto acima poderá ser feito tanto em uma coluna quanto em uma linha, a interpolação bilinear é feita do seguinte modo:

1. Primeiro, considerando apenas a linha de cima da imagem e ignorando todo o resto, podemos fazer uma interpolação linear no eixo x de forma que:

$$X = A \cdot \frac{x - x_1}{x_2 - x_1} + B \cdot \frac{x_2 - x}{x_2 - x_1} \tag{1}$$

2. Fazendo o mesmo na linha de baixo:

$$Y = C \cdot \frac{x - x_1}{x_2 - x_1} + D \cdot \frac{x_2 - x}{x_2 - x_1} \tag{2}$$

3. Para encontrar Z, agora que temos os valores de X e Y, fazemos o mesmo procedimento no intervalo vertical entre eles:

$$Z = X \cdot \frac{y - y_1}{y_2 - y_1} + Y \cdot \frac{y_2 - y}{y_2 - y_1} \tag{3}$$

6.1. Observações

- Esse processo também pode ser feito de forma ”direta” se substituirmos os valores de X e Y no cálculo de Z;
- Nesse caso, considera-se que os eixo x e y crescem da esquerda pra direita e de cima para baixo, respectivamente.

7. QUESTÃO G

Fig. 3. Imagem reconstruídas utilizando a técnica da interpolação bilinear.

Imagens produzidas utilizando o código 5.

8. QUESTÃO H

PSNR

PSNR é a sigla em inglês do termo *Peak Signal-to-Noise Ratio* (relação sinal-ruído de pico). Esse é uma medida que se utiliza do erro médio quadrático (MSE - *Mean squared error*) para calcular ”a relação entre a máxima energia de um sinal e o ruído que afeta sua representação fidedigna”.

O PSNR geralmente é dado utilizando escala logarítima, em decibéis.

A fórmula para obtenção desse valor é:

$$
\begin{aligned}
PSNR &= 10 \cdot \log_{10}\left(\frac{MAX_I^2}{MSE}\right) \\
&= 20 \cdot \log_{10}\left(\frac{MAX_I}{\sqrt{MSE}}\right) \\
&= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE)
\end{aligned}
$$

Fig. 4. PSNR.

Onde:

$$
MSE = \frac{1}{mn}\sum_{0}^{m-1}\sum_{0}^{n-1}\|f(i,j) - g(i,j)\|^2
$$

Fig. 5. MSE

9. QUESTÃO I

Utilizando a psnr como métrica base, pode-se dizer que, para a imagem utilizada, o método de amplificação da imagem por meio interpolação bilinear se mostrou muito mais fiel às imagens originais (em todos os casos aqui experimentados) do que o método do vizinho mais próximo (knn).

	size	psnr_knn_db	psnr_bilinear_db
0	256 x 256	26.52	30.38
1	128 x 128	21.76	25.75
2	64 x 64	18.71	22.66
3	32 x 32	16.37	20.04

Fig. 6. PSNR com knn e interpolação bilinear

10. QUESTÃO J

Se fôssemos fazer uma conversão de uma linha que vai de 0 a 255 para outra linha que vai de 0 a L, poderíamos seguir o seguinte raciocínio:

$y = ax + b$, onde

- y é o valor equivalente na linha L
- x é o valor original
- a é o coeficiente angular
- b é o coeficiente linear

Sabendo que o 0 será 0 em qualquer uma das retas:
$0 = a \cdot 0 + b$. Logo, $b = 0$
Convertendo o ponto 255 para o ponto L:
$L = a \cdot 255$. Logo, $a = \frac{L}{255}$ e $\frac{1}{a} = \frac{255}{L}$.
Esse valor $\frac{1}{a}$ será o valor pelo qual dividiremos os valores dos pixels originais para obter o pixel dentro dos demais ranges propostos.

- 16 = 4 bits,
- 4 = 2 bits e
- 2 = 1 bit.

11. QUESTÃO J

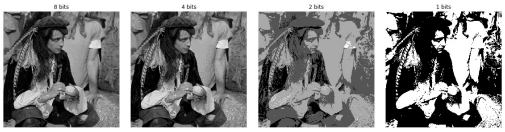

Fig. 7. Visualização da imagem com 8, 4, 2 e 1 bits.

Essa imagem foi gerada utilizando o código 7

12. CÓDIGOS

Obs.: linhas finalizadas com **...** no final são linhas que continuarão na linhas abaixo e que tiveram que ser quebradas em duas para não ultrapassar o limite da página.

Listing 1. Importando e verificando shape da imagem

```
1
2 img_path = os.path.abspath('images/standard_test_images')
3 img_name = 'pirate.tif'
4 file = os.path.join(img_path, img_name)
5 img = cv.imread(file, 0)
6
7 print(img.shape)
8 print('Maximum value: %d\nMinimum value: %d' % (img.max(), img.min()))
```

Listing 2. Mostrando e exportando a imagem

```
1 fig = plt.imshow(img, cmap = 'gray', vmin = 0, vmax = 255)
2 plt.colorbar(fig)
3 plt.savefig("b.jpg")
```

Listing 3. Reduzindo as dimensões da imagem

```
1
2  def downsample_512(img):
3          original_size = 512
4          size = original_size
5          images = {}
6          idx = 0
7
8          img = cv.imread(file, 0)
9          fig, axs = plt.subplots(1, 5, figsize=[15, 15])
10         fig.subplots_adjust(wspace=0.1)
11
12         axs[0].axis('off')
13         axs[0].imshow(img, cmap = 'gray', vmin = 0, vmax = 255, interpolation='none')
14         axs[0].set_title('%d x %d' % (size, size))
15
16
17         for ax in axs[1:]:
18                 img = img[0 : size : 2, 0 : size : 2]
19                 images[idx] = img
20                 idx += 1
21                 size = int(size / 2)
22
23                 ax.axis('off')
24                 ax.imshow(img, cmap = 'gray', vmin = 0, vmax = 255, interpolation='none')
25                 ax.set_title('%d x %d' % (size, size))
26         plt.savefig("c.jpg")
27 return images
28
29 images = {}
30 images = downsample_512(img)
```

Listing 4. Aumentando a imagem pelo método do KNN

```
def knn_512(images):
        n_images = len(images)
        new_images = {}
        final_size = 512

        plot_row = 0
        plot_column = 0

        fig, axs = plt.subplots(2, round(n_images/2), figsize=[5, 5])
        fig.subplots_adjust(wspace=0.1)

        for idx in range(n_images):
                initial_size = len(images[idx])
                space = final_size / initial_size
                new_images_row = np.zeros([initial_size, final_size])
                new_images[idx] = np.zeros([final_size, final_size])

                new_images_idx = np.zeros([final_size, final_size])
                for row in range(initial_size):
                        for column in range(final_size):
                                new_images_idx[row][column] = int(column / space)
                                pixel = images[idx][row][int(new_images_idx[row][column])]
                                new_images_row[row][column] = pixel


                new_images_idx = np.zeros([final_size, final_size])
                for column in range(final_size):
                        for row in range(final_size):
                             new_images_idx[row][column] = int(row / space)
                             pixel = new_images_row[int(new_images_idx[row][column])][column]
                             new_images[idx][row][column] = pixel


                axs[plot_row][plot_column].axis('off')
                axs[plot_row][plot_column].imshow(new_images[idx], cmap = 'gray',
                                                  vmin = 0, vmax = 255,
                                                  interpolation='none')
                axs[plot_row][plot_column].set_title('%d_x_%d' % (initial_size, initial_size))

                plot_column += 1
                if(plot_column == round(n_images/2)):
                        plot_row += 1
                        plot_column = 0

        plt.savefig('e.jpg')
return new_images;

new_images = knn_512(images)
```

Listing 5. Aumentando a imagem pela interpolação bilinear

```
def bilinear_512(images):
        n_images = len(images)
        new_images = {}
        final_size = 512

        plot_row = 0
        plot_column = 0

        fig, axs = plt.subplots(2, round(n_images/2), figsize=[5, 5])
        fig.subplots_adjust(wspace=0.1)

        for img in range(n_images):
                initial_size = len(images[img])
                space = final_size / initial_size
                new_images[img] = np.zeros([final_size, final_size])

                for row in range(final_size):
                        percent_row = row/space - math.floor(row/space)
                        idx_row = math.ceil(row/space) if row/space < (initial_size - 1) ...
                                                       else math.floor(row/space)

                        for col in range(final_size):
                                percent_col = col/space - math.floor(col/space)
                                idx_col = math.ceil(col/space) if col/space < ...
                                (initial_size - 1) else math.floor(col/space)

                                x_floor = images[img][math.floor(row/space)] ...
                                [math.floor(col/space)]
                                x_ceil  = images[img][math.floor(row/space)][idx_col]
                                x = ((1 - percent_col) * x_floor) + (percent_col * x_ceil)

                                y_floor = images[img][idx_row][math.floor(col/space)]
                                y_ceil  = images[img][idx_row][idx_col]
                                y = ((1 - percent_col) * y_floor) + (percent_col * y_ceil)

                                new_images[img][row][col] = ((1 - percent_row) * x) + ...
                                ((percent_row) * y)


                        axs[plot_row][plot_column].axis('off')
                        axs[plot_row][plot_column].imshow(new_images[img], cmap = 'gray',
                                                          vmin = 0, vmax = 255,
                                                          interpolation='none')
                        axs[plot_row][plot_column].set_title('%d x %d' % ...
                                                             (initial_size, initial_size))

        plot_column += 1
        if(plot_column == round(n_images/2)):
        plot_row += 1
        plot_column = 0
        plt.savefig('g.jpg')
        return new_images;
new_images = bilinear_512(images)
```

Listing 6. Mostrando os valores de PSNR da imagem em dataFrame

```
def img_psnr(image, original_image):
        M = original_image.shape[0]
        N = original_image.shape[1]
        mse = 0

        for i in range(M):
                for j in range(N):
                        mse = mse + abs(original_image[i][j] - image[i][j])**2

        mse = mse/(M * N)
        PSNR = 20 * math.log10(255/math.sqrt(mse))
return PSNR

images_bilinear = bilinear_512(images)
images_knn = knn_512(images)

dim = [256, 128, 64, 32]
psnr_df = pd.DataFrame(columns = ['size', 'psnr_knn_db', 'psnr_bilinear_db'])
for i in range(len(images_knn)):
        psnr_knn = img_psnr(images_knn[i], img)
        psnr_bilinear = img_psnr(images_bilinear[i], img)
        psnr_df = psnr_df.append({'size': '%d x %d'% (dim[i], dim[i]),
                'psnr_knn_db': '%.2f'% psnr_knn,
                'psnr_bilinear_db': '%.2f'% psnr_bilinear},
                ignore_index=True)
                psnr_df
```

Listing 7. Reduzindo o número de bit/pixel da imagem

```
def img_scale(image):
        n_images = 4
        images = {}
        fig, axs = plt.subplots(1, n_images, figsize=[20, 20])
        fig.subplots_adjust(wspace=0.1)

        for idx in range(n_images):
                ratio = 255/(2**(2**idx) - 1)
                images[idx] = np.round(np.divide(image, ratio))

                axs[(n_images - 1) - idx].axis('off')
                axs[(n_images - 1) - idx].imshow(images[idx], cmap = 'gray',
                                                 interpolation='none')
                axs[(n_images - 1) - idx].set_title('%d_bits' % (2**idx))

        plt.savefig('j.jpg')

img_scale(img)
```