Pontifícia Universidade Católica de Goiás
Escola de Ciências Exatas e da Computação
Curso de Engenharia de Computação

LUCAS MACEDO DA SILVA

LISTA 2 – TRANSFORMAÇÕES PONTO A PONTO

Goiânia
2020

Considerações iniciais

Para a resolução do presente trabalho foi utilizada a seguinte imagem.

Figura 1 – Imagem utilizada

Fonte: Google imagens (2020)

Ela é uma imagem de 1920 x 1080 pixels e foi utilizada em todas as questões.

Questão 1

Converta a sua imagem em uma imagem cinza e normalize a imagem para valores entre [0 e 1]. Posteriormente binarize a imagem com limite k = 0,68 e qualquer número desejado e, mostre as imagens resultantes.

O seguinte trecho de código foi utilizado para resolução da questão, as funções normaliza, escala_cinza e binariza serão explicadas posteriormente.

Código 1 – Código para resolução da questão 1

```
img = imread('D:\Estudos\Eng. de Computação\Processamento Digital de Imagens\Listas\Lista 2\Imagens\img.jpg');

L = 0.1;

img_cinza = escala_cinza(img);
imshow(img_cinza);
title('Imagem em escala de cinza');

img_norm = normaliza(img_cinza);
figure();
imshow(img_norm);
```

```
title('Imagem normalizada');

img_bin = binariza(img_norm, L);
figure();
imshow(img_bin);
title(strcat('Imagem Binarizada com L = ', num2str(L)));
```

Fonte: Autoria Própria (2020)

Parte 1 – Conversão da imagem em escala de cinza

Para realizar a conversão da imagem em escala de cinza foi utilizada a seguinte equação:

$$s = 0.29 * R + 0.59 * G + 0.11 * B$$

Onde R corresponde ao componente vermelho (*red*), G corresponde ao componente verde (*green*) e B corresponde ao componente azul (*blue*), do pixel antigo da imagem, s corresponde ao novo valor do pixel. Segundo Nogueira (2016) esse é o método de conversão clássico para escala de cinza e é o mesmo método usado pela função rgb2gray do Matlab.

Para converter a imagem em escala de cinza foi desenvolvida a seguinte função em Matlab. Ela recebe como entrada a imagem e retorna a imagem em escala de cinza como saída. Para realizar esse procedimento, em cada pixel da imagem é aplicada a equação retromencionada gerando o novo pixel em escala de cinza.

Código 2 – Função escala_cinza

```
function imagem = escala_cinza(img)
   [i j k] = size(img);

   imagem = zeros(i, j, 'uint8');

   for x = 1 : i
      for y = 1 : j
         imagem(x, y) = 0.29 * img(x, y, 1) +  0.59 * img(x, y, 2) + 0.11 * img(x, y, 3);
      end
   end
end
```

Fonte: Autoria Própria (2020)

Ao executar a função obtém-se a seguinte imagem em escala de cinza.

Figura 2 – Imagem em escala de cinza

Fonte: Autoria Própria (2020)

Parte 2 – Normalização da imagem para a escala entre 0 e 1

Para normalizar a imagem para a escala entre 0 e 1, foi utilizada a seguinte função desenvolvida pelo autor.

Código 3 – Função normaliza

```
function imagem = normaliza(img)

   L = max(max(img)); % Define o maior valor da escala, por ser uma matriz foi necessário usar a função max duas veze

   [i j k] = size(img);

   imagem = zeros(i, j, k, 'double');

   for x = 1: i
      for y = 1: j
         imagem(x, y, :) = double(img(x, y, :)) ./ double(L);
      end
   end
end
```

Fonte: Autoria Própria (2020)

A normalização pode ser definida como uma operação matemática na qual um conjunto de valores numéricos de diversas magnitudes fiquem em uma faixa definida de valores. Uma das formas de normalizar o conjunto é dividir todos os elementos pelo maior valor presente no conjunto. Por exemplo, a imagem de entrada da função normaliza está em escala de cinza, com isso possui valores de pixels de 0 a 255, dessa forma para normalizá-la é necessário dividir todos os pixels pelo valor máximo que no caso é 255.

Sendo assim, a normalização aqui utilizada tem a seguinte forma:

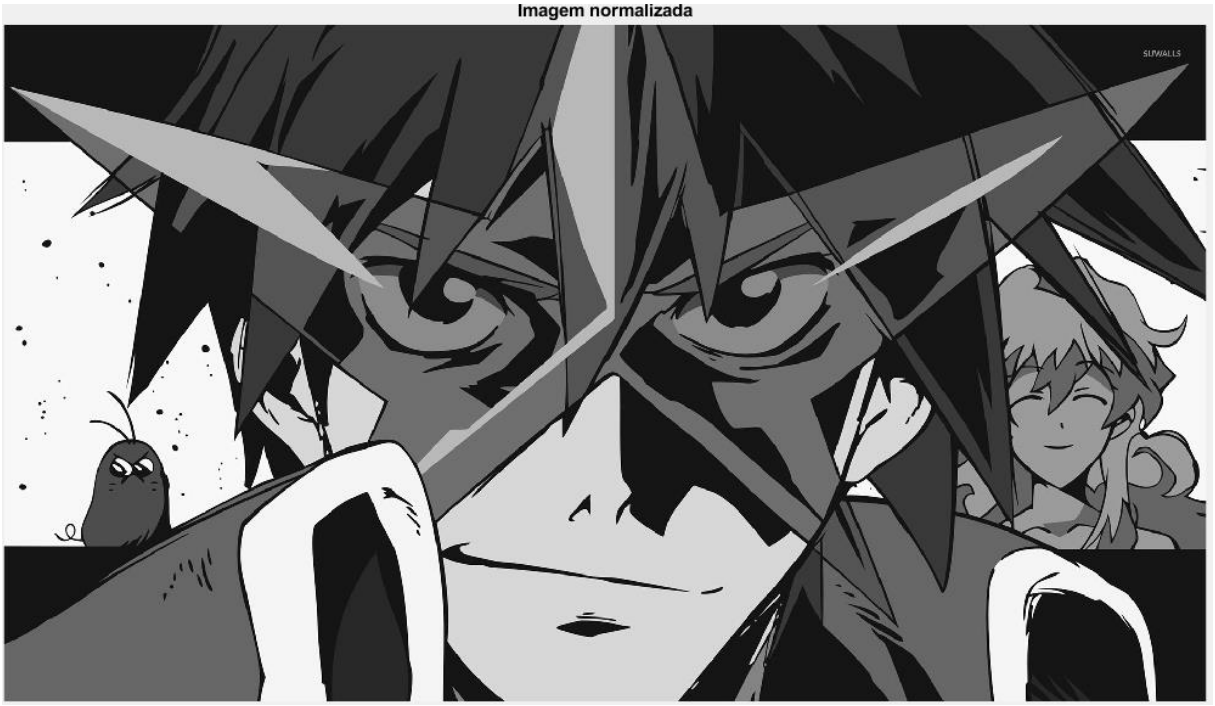
$$pixel_{normalizado} = \frac{pixel(x, y)}{L}$$

Onde $pixel_{normalizado}$ corresponde ao novo valor do pixel após aplicar a operação, pixel(x, y) é o valor do pixel e L é o valor máximo do intervalo, que nesse caso é 255.

Dessa forma, obtém-se uma imagem normalizada com os pixels entre os valores 0 e 1. Uma nova imagem foi criada como *double* para comportar os novos valores dos pixels.

Após a aplicação da normalização da imagem em escala de cinza, obtém-se a seguinte imagem.

Figura 3 – Imagem normalizada



Fonte: Autoria Própria (2020)

Parte 3 – Binarização da imagem

A binarização de uma imagem consiste em determinar um valor de pixel, e valores menores ou iguais a esse pixel recebem um valor especifico (como o valor correspondente ao branco) e o os valores maiores que esse valor recebem um outro valor especifico (como o valor correspondente ao preto). Assim, é possível separar a imagem em duas cores diferentes. Esse é o tipo mais básico de binarização (MARQUES FILHO; VIEIRA NETO, 1999).

Em termos matemáticos a binarização pode ser definida como:

$$pixel_{novo} = \begin{cases} 1, se\ pixel_{antigo} \leq L \\ 0, caso\ contrário \end{cases}$$

Onde $pixel_{novo}$ refere-se ao novo valor de pixel após a aplicação da operação, $pixel_{antigo}$ refere-se ao antigo valor do pixel, L é o valor que define a binarização, isto é, é o limiar.

Esse tipo de binarização foi aplicado no presente trabalho.

A função escrita em Matlab que binariza a imagem é a seguinte.

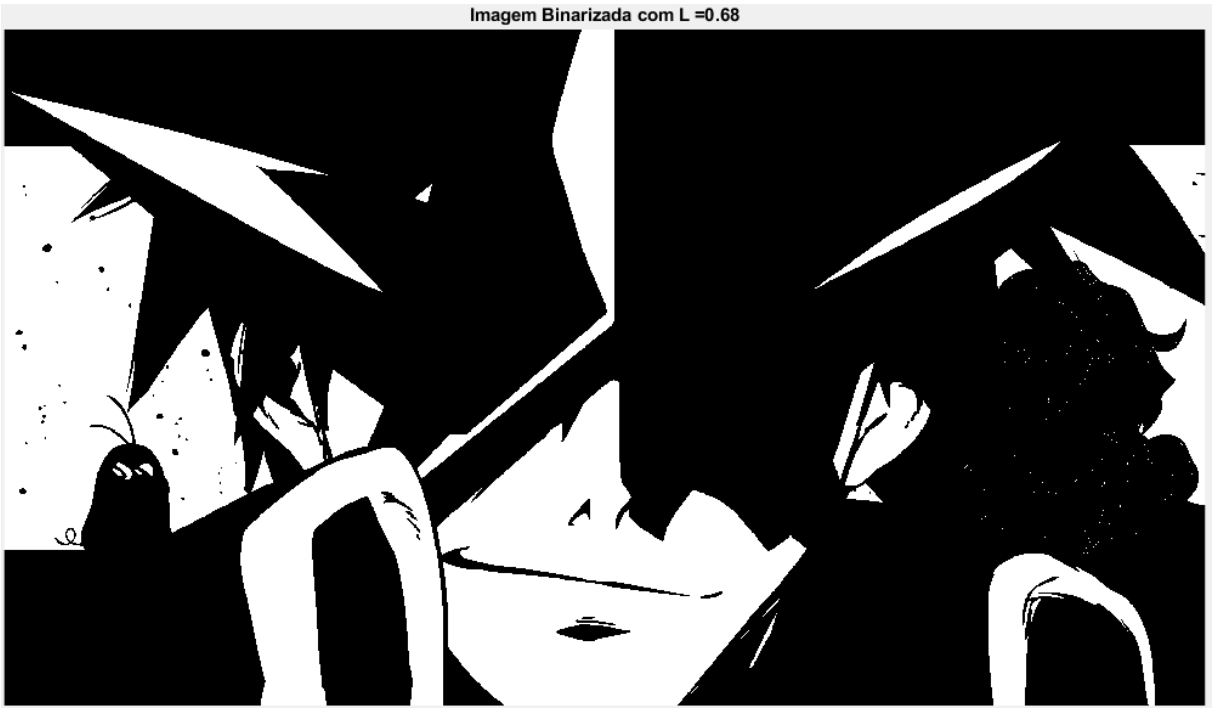
Código 4 – Função binariza

```
function imagem = binariza(img, lim)

   [i j k] = size(img);

   imagem = zeros(i, j, k, 'double');

   for x = 1: i
      for y = 1: j
         if img(x, y, :) <= lim
            imagem(x, y, :) = 0;
         else
            imagem(x, y, :) = 1;
         end
      end
   end
end
```

Fonte: Autoria Própria (2020)

A primeira execução foi realizada com um limite (L), isto é, o valor que divide as faixas, igual a 0.68 como recomendado no exercício. Foi então obtida a seguinte imagem.

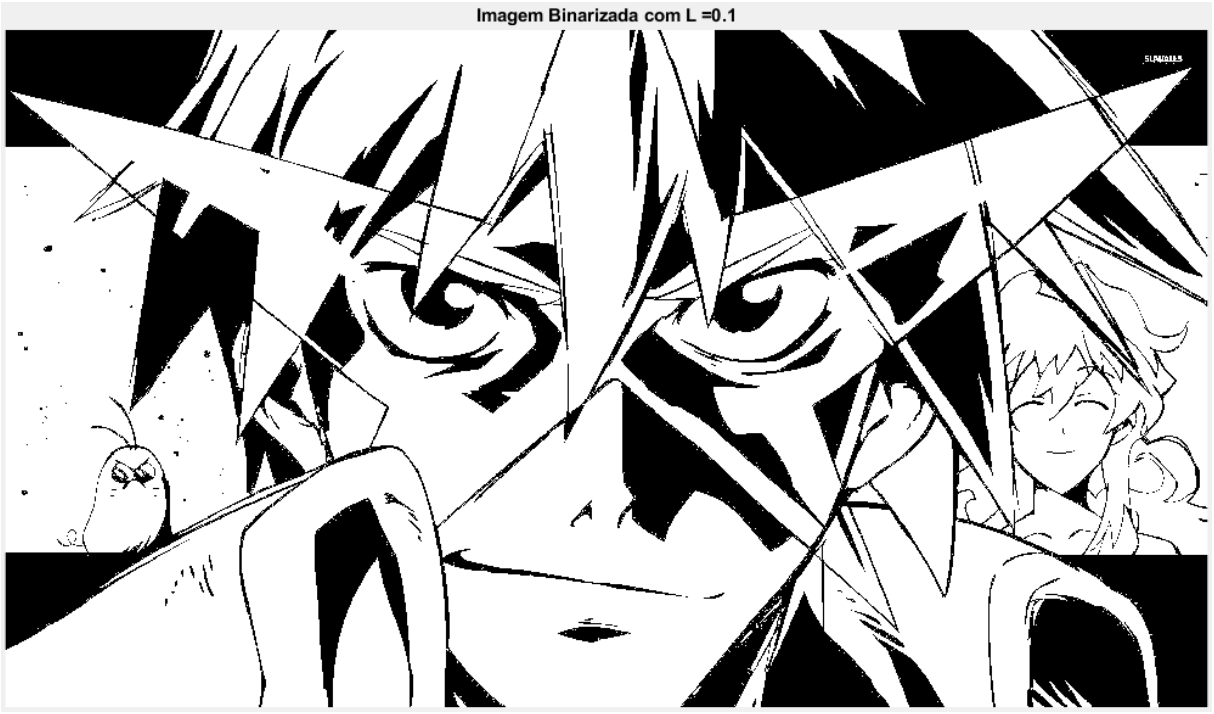
Figura 4 – Imagem binarizada com limite = 0.68



Fonte: Autoria Própria (2020)

Posteriormente o limite foi definido para 0.1 obtendo a seguinte imagem.

Figura 4 – Imagem binarizada com limite = 0.1



Fonte: Autoria Própria (2020)

Questão 2

Execute a correção gama para a imagem inicial em escala de cinza e normalize a imagem para valores entre [0 e 1]. A equação de transformação é $s=r*cy$, onde r é a intensidade de entrada, s é a intensidade de saída, c é uma constante e geralmente é definida como 1, y é o parâmetro que você pode alterar. Defina y = 0.2, y = 0.8 e y = 2.5, exiba e comente os diferentes resultados.

Correção Gama

Gonzalez e Woods (2010) afirmam que a correção gama é importante para a exibição com exatidão da imagem na tela de um computador. Além disso ela altera a intensidade dos pixels, bem como as proporções RGB da imagem.

Gonzalez e Woods (2010) trazem a equação para realizar a correção gama na imagem:

$$s = c * r^{\gamma}$$

Onde,

s é o novo valor de intensidade do pixel,

c é uma constante, costuma-se usar c = 1, essa convenção foi usada no presente trabalho,

r é o valor de intensidade do pixel antigo.

Partindo dessa equação a seguinte função foi escrita no *software* Matlab para realizar a correção gama na imagem.

Código 5 – Função correcao_gama

```
function imagem = correcao_gama(img, gama, c)
   [i j k] = size(img);
   imagem = zeros(i, j, k, 'double');
   for x = 1: i
      for y = 1: j
         imagem(x, y, :) = c*(double(img(x, y, :)).^double(gama));
      end
   end
end
```

Fonte: Autoria Própria (2020)

A ideia por trás é percorrer cada pixel da imagem e calcular o novo valor baseado na equação da correção gama.

Código completo

Para realizar as operações necessárias citadas no enunciado, foi utilizado o seguinte código.

Código 6 – Função para resolução da Questão 2

```
function imagem = Questao2(img, gama, c)

   img_cinza = escala_cinza(img);
   imshow(img_cinza);
   title('Imagem em escala de cinza');

   img_corr = correcao_gama(img_cinza, gama, c);
   figure();
   imshow(img_corr);
   title(strcat('Imagem com correcao gama =  ', num2str(gama)));

   imagem = normaliza(img_corr);
   figure();
   imshow(imagem);
   title(strcat('Imagem normalizada com correcao gama =  ', num2str(gama)));

end
```

Fonte: Autoria Própria (2020)

Para converter a imagem para a escala de cinza foi utilizada a função escala_cinza (Código 2 – Função escala_cinza), a imagem resultante foi a mesma que a apresentada na Figura 2.

Foi aplicada a correção gama na imagem em escala de cinza. Para isso, foi utilizado o Código 5 – função correcao_gama retromencionado. A normalização da imagem foi realizada com a função apresentada no código Código 3 – Função normaliza.

Por fim os parâmetros foram variados conforme pedido na questão, obtendo as imagens descritas a seguir.

Vale ressaltar que as imagens aqui apresentadas são aquelas resultantes do processo. Ou seja, a imagem em escala de cinza com correção gama aplicada e já normalizada.

Gama = 0.2

Figura 5 – Imagem com correção gama, gama = 0.2

Fonte: Autoria Própria (2020)

Gama = 0.8

Figura 6 – Imagem com correção gama, gama = 0.8

Fonte: Autoria Própria (2020)

Gama = 2.5

Figura 7 – Imagem com correção gama, gama = 2.5

Fonte: Autoria Própria (2020)

Comentários

A partir das três imagens obtidas ao variar o coeficiente gama, percebe-se que quanto maior o valor de gama, mais escura a imagem se torna. Assim, conforme aumenta-se o valor de gama a intensidade dos pixels de saída se aproxima da intensidade máxima da faixa de valores.

O que está de acordo com Gonzalez e Woods (2010) que afirma que para gama = 2.5 os sistemas tendem a produzir uma imagem mais escura que o pretendido. Esse fato fica evidente na Figura 7 que em comparação com a Figura 2 é visualmente mais escura.

Questão 3

Refaça a questão 2, mas utilize sua imagem colorida.

Para a resolução desta questão a imagem não foi transformada em escala de cinza. Com isso, a entrada para o algoritmo é a imagem colorida.

A seguinte função foi utilizada para resolução da questão 3.

Código 7 – Função para resolução da Questão 3

```
function imagem = Questao3(img, gama, c)

    img_corr = correcao_gama(img, gama, c);
    figure();
    imshow(img_corr);
```

```
    title('Imagem com correcao gama = 2.5');

    imagem = normaliza(img_corr)
    figure();
    imshow(imagem);
    title(strcat('Imagem colorida normalizada com correcao gama =  ', num2str(gama)));

end
```

Fonte: Autoria Própria (2020)

A função correção_gama está descrita no Código 5 – Função correcao_gama e a função normaliza está descrita no Código 3 – Função normaliza.

Ao executar o código citado variando-se o parâmetro gama, obtém-se as seguintes imagens.

Gama = 0.2

Figura 5 – Imagem com correção gama, gama = 0.2

Fonte: Autoria Própria (2020)

Gama = 0.8

Figura 6 – Imagem com correção gama, gama = 0.8

Fonte: Autoria Própria (2020)

Gama = 2.5

Figura 7 – Imagem com correção gama, gama = 2.5

Fonte: Autoria Própria (2020)

Comentários

Os comentários são os mesmos daqueles apresentados na questão anterior, exceto pelo fato de que a imagem agora é colorida, porém em relação a teoria não muda em nada. Com isso, quanto maior o valor de gama percebe-se que a imagem fica mais escura.

Já em relação a cada valor de gama, em ambas as questões (2 e 3), percebe-se que o valor de gama = 0.2 a imagem está mais clara que a original; o valor de gama = 0.8 implica em uma imagem quase idêntica a original porém mais clara; e em relação a gama = 2.5 a imagem é mais escura que a original.

Questão 4

Crie seu algoritmo para calcular o histograma de uma imagem e utilize-a na imagem resultante da questão 2 e 3 para y=2.5.

O histograma de uma imagem é um conjunto de números que representam quantas vezes determinada intensidade de cor apareceu na imagem. Esse conjunto costuma ser representado em um gráfico de barras que fornece para cada nível o valor correspondente de pixels naquela imagem (MARQUES FILHO; VIEIRA NETO, 1999).

Gonzalez e Woods (2010) definem um histograma como sendo uma função discreta

$$h(r_k) = n_k$$

Onde r_k é o k-ésimo valor de intensidade e n_k é o número de pixels com intensidade r_k.

Partindo da equação foi então desenvolvida a seguinte função para calcular o histograma de uma imagem.

Código 8 – Função histograma

```
function hist = histograma(img)
   [i j k] = size(img);

   hist = zeros(256, 1);

   index = 1;

   img = im2uint8(img);

   for x = 1 : i
      for y = 1 : j
         index = (img(x, y, :)) + 1;
         hist(index) = hist(index) + 1;
      end
   end

   bar( 0:255, hist, 'histc');
end
```

Fonte: Autoria Própria (2020)

Como os valores resultantes das questões 2 e 3 são imagens em que os pixels estão em formato *double*, foi empregada a função im2uint8 para converter a imagem de double para uint8 facilitando assim no momento de contagem de intensidade dos pixels.

A função bar do Matlab plota o gráfico de barras.

Assim, o seguinte código chama as funções de resolução da questão 2 (Código 6 – Função para resolução da Questão 2) e questão 3 (Código 7 – Função para resolução da Questão 3) e plota os histogramas de cada uma das imagens resultantes.

Código 9 – Código para resolução da Questão 4

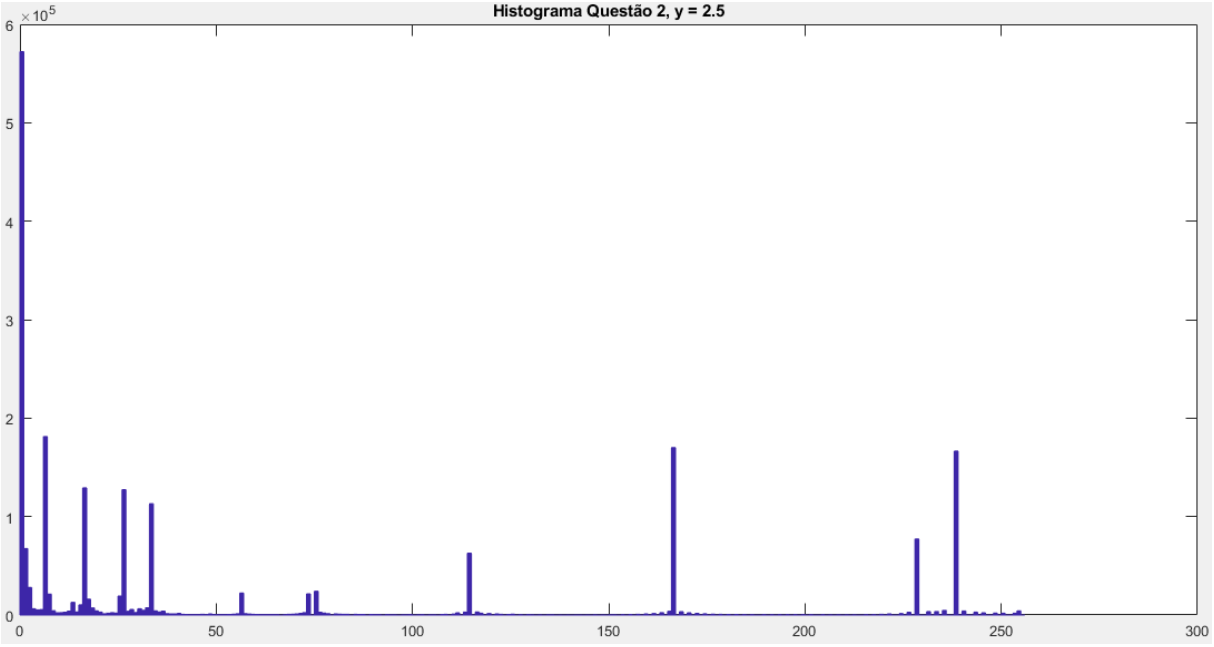
```
img = imread('D:\Estudos\Eng. de Computação\Processamento Digital de Imagens\Listas\Lista 2\Imagens\img.jpg');
imagem = (Questao2(img, 2.5, 1));
figure();
histograma(imagem);
title('Histograma Questão 2, y = 2.5');

imagem = (Questao3(img, 2.5, 1));
figure();
histograma(imagem);
title('Histograma Questão 3, y = 2.5');
```

Fonte: Autoria Própria (2020)

Os histogramas resultantes são apresentados a seguir.

Histograma 1 – Histograma para a imagem resultante da questão 2 com gama = 2.5

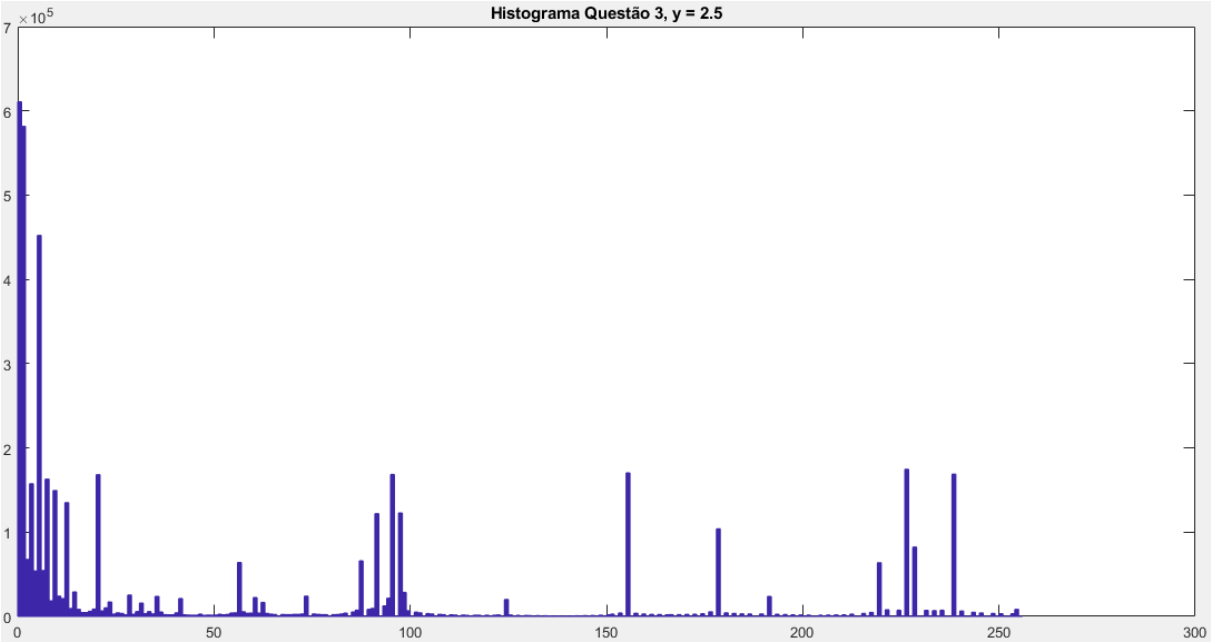
Figura 8 – Histograma para a imagem resultante da questão 2 com gama = 2.5



Fonte: Autoria Própria (2020)

Histograma 2 – Histograma para a imagem resultante da questão 3 com gama = 2.5

Figura 9 – Histograma para a imagem resultante da questão 3 com gama = 2.5



Fonte: Autoria Própria (2020)

Questão 5

Torne a imagem de entrada mais clara ou mais escura.

Segundo Marques Filho e Vieira Neto (1999) a operação de adição, somar uma constante *c* a todos os pixels de uma imagem, torna-a mais clara. Sendo assim, esse pressuposto foi utilizado na resolução do problema.

A seguinte função realiza a adição da constante *c* em todos os pixels da imagem.

Código 10 – Função para tonar a imagem mais clara

```
function imagem = img_mais_clara(img, c)
   [i j k] = size(img);

   imagem = zeros(i, j, k, 'uint8');

   for x = 1 : i
     for y = 1 : j
       imagem(x, y, :) = img(x, y, :) + c;
     end
   end
end
```

Fonte: Autoria Própria (2020)

Primeiro percorre-se a imagem, e em cada valor de pixel é somada a constante *c* naquele valor. Vale ressaltar que para realizar a operação de escurecimento, tornar a imagem mais escura, basta passar um valor de c < 0, isto é, realizar a operação de subtração ((MARQUES FILHO; VIEIRA NETO, 1999).

O código a seguir lê a imagem do disco, chama a função img_mais_clara (Código 10 – Função para tonar a imagem mais clara) e então plota a imagem resultante. Para a resolução desse problema foi usado um c = 100.

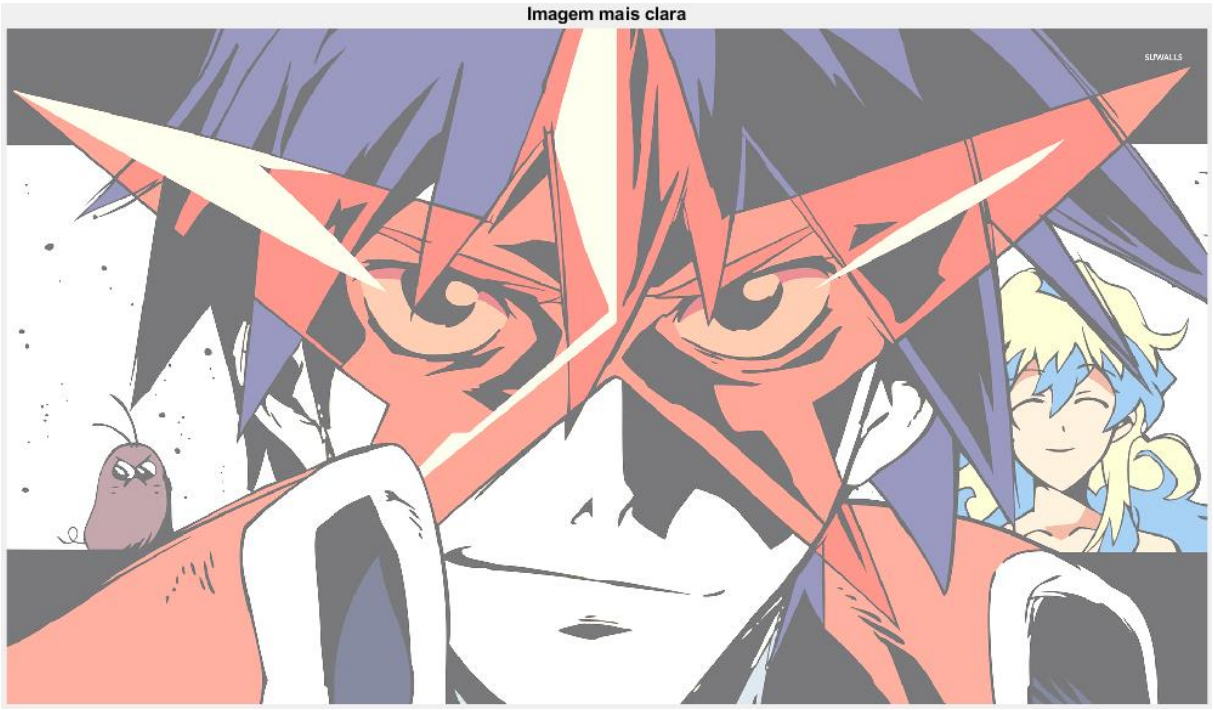
Código 11 – Código para a resolução da Questão 5

```
img = imread('D:\Estudos\Eng. de Computação\Processamento Digital de Imagens\Listas\Lista 2\Imagens\img.jpg');

c = 100;
img_clara = img_mais_clara(img, c);

imshow(img_clara);
title('Imagem mais clara');
```

Fonte: Autoria Própria (2020)

Como resultado foi obtida a seguinte imagem.

Figura 10 – Imagem mais clara



Fonte: Autoria Própria (2020)

Questão 6

Escolha um limiar e aplique em todos os componentes da sua imagem.

Para a resolução deste problema considerou-se que limiar se refere ao valor que separa a determinação de contastes de um pixel na transformação de limiarização. Com isso pode-se utilizar a seguinte função para realizar a limiarização da imagem (GONZALEZ; WOODS 2010), (MARQUES FILHO; VIEIRA NETO, 1999), (KULKARNI, 2012), (GUIMARÃES; CANDEIAS, 2009):

$$pixel_{novo} = \begin{cases} = 255, se\ pixel_{antigo} \geq L \\ = 0, \quad caso\ contrário \end{cases}$$

Onde $pixel_{novo}$ refere-se ao novo valor do pixel ao aplicar a operação, L se refere ao limite de limiarização e $pixel_{antigo}$ é o valor anterior do pixel. Esse processo foi aplicado em cada canal de cor da imagem, isto é, nos canais R, G e B. O valor escolhido foi L = 128.

Assim, esse valor foi utilizado para determinar se o pixel receberia a cor branca ou a cor preta. A seguinte função foi utilizada para realizar a limiarização da imagem.

Código 12 – Função para limiarização

```
function imagem = limiarizacao(img, L)
   [i j k] = size(img);
   imagem = zeros(i, j, k, 'uint8');

   for x = 1 : i
      for y = 1 : j
         for z = 1: k
              imagem(x, y, z) = verifica(img(x, y, z), L);
         end
      end
   end
end

function pixel = verifica(pixel_ant, L)
   if pixel_ant >= L
       pixel = 255;
   else
       pixel = 0;
   end
end
```

Fonte: Autoria Própria (2020)

Como limiar foi utilizado 128 a metade da instensidade máxima que o pixel pode receber. O seguinte código lê a imagem do disco e aplica a função de limiarização e então mostra a imagem resultante.

Código 13 – Código para resolução da questão 6

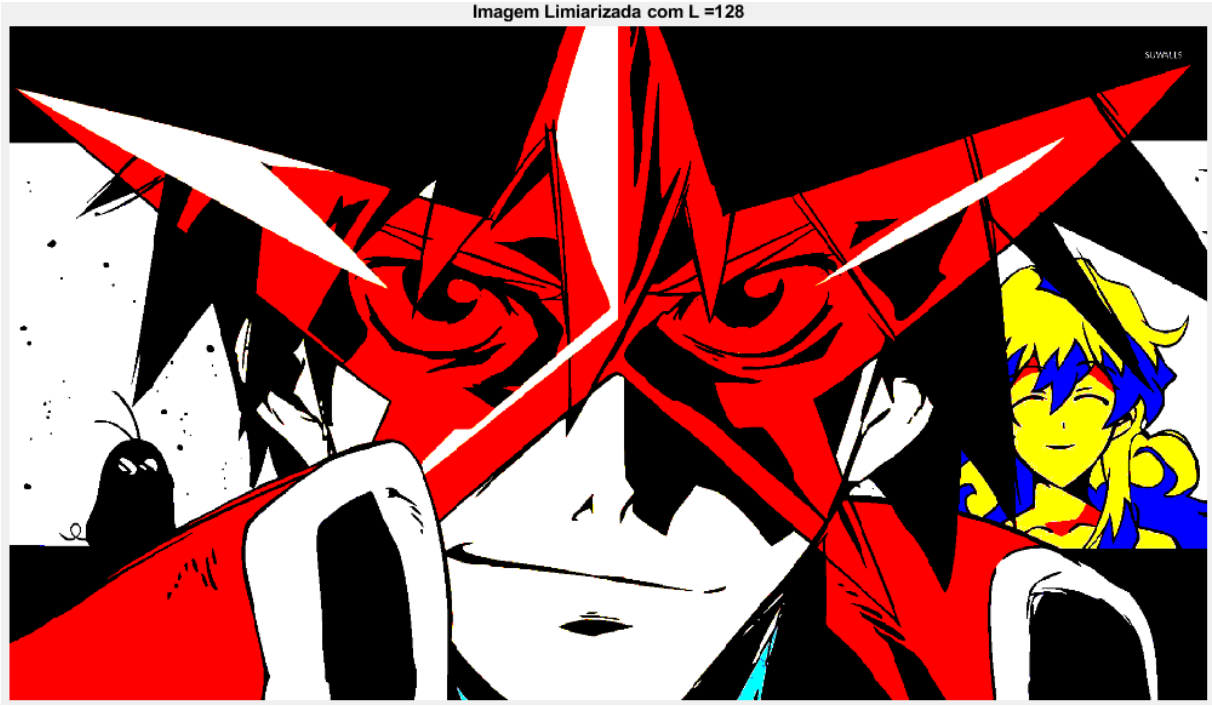
```
img = imread('D:\Estudos\Eng. de Computação\Processamento Digital de Imagens\Listas\Lista 2\Imagens\img.jpg');
```

```
L = round(255/2);
img_lim = limiarizacao(img, L);

imshow(img);
title('Imagem original');

figure();
imshow(img_lim);
title(strcat('Imagem Limiarizada com L = ', num2str(L)));
```

Fonte: Autoria Própria (2020)

Para a resolução desse problema foi utilizado um limiar igual a 128. A imagem resultante é apresentada a seguir.

Figura 11 – Imagem limiarizada com limiar igual a 128



Fonte: Autoria Própria (2020)

Questão 7

Quantifique os planos de cores usando 2 bits e 4 bits visualize o efeito das operações.

Para a resolução desta questão, foi considerado que os bits indicam quantos níveis de cor que a imagem possuirá, após realizada a operação, a seguinte equação expressa isso:

$$qtd_{cores} = 2^n$$

Onde qtd_{cores} refere-se a nova quantidade de cores que a imagem terá e n é o número de bits. Considere, por exemplo, n = 2 a nova imagem será formada por 4 cores diferentes.

Marques Filho e Vieira Neto (1999) definem esse processo como digitalização, já Gonzalez e Woods (2010) chamam esse processo de quantização. Desta forma, a ideia por trás do processo aqui desenvolvido é reduzir a quantidade de cores que determinada imagem possui para uma quantidade menor.

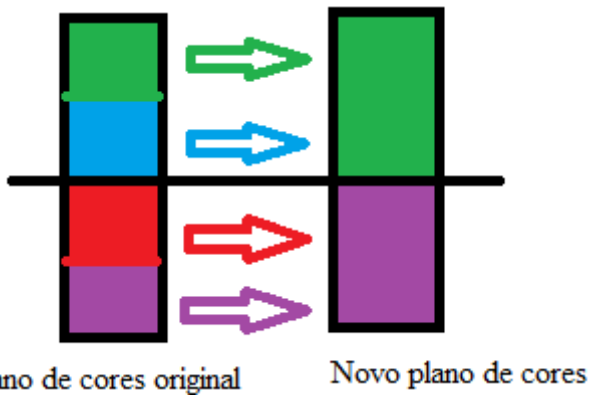
A imagem original (Figura 1) utilizada para a resolução da questão foi lida do disco como uint8 que indica que ela possui 8 bits para representar suas cores. Ela possui, portanto, 2^8 = 256 cores distintas. Após aplicar o processo desenvolvido ela apresentará 4 e 16 cores, para n = 2 e n = 4, respectivamente. Reduzindo assim, os planos de cores.

Explicação do algoritmo

A imagem original possui 256 possíveis cores. Dado um número n de bits existiram 2^n possíveis cores para o novo plano de cores. Torna-se necessário definir os limites que definem as cores entre os dois sistemas, isto é, as cores de 0 a i, i + 1 a i + n, e assim sucessivamente da imagem original devem ser mapeadas para uma nova escala do novo plano de cores.

Considere, por exemplo, os dois planos de cores a seguir (Figura 12). O plano a esquerda é o plano origina e possui 4 possíveis cores, o plano a direita é o novo plano e possui duas cores. Sendo assim, para reduzir o número de bits as cores do plano original foram mapeadas para cores do novo plano. As cores verde e azul do plano original se tornaram verde no novo plano e as cores vermelho e roxo do plano original se tornaram roxo no novo plano.

Figura 12 – Exemplo de mapeamento do plano de cores antigo para um novo plano de cores com menos bits



Fonte: Autoria Própria (2020)

Para definir as cores (os valores de pixels) que constaram no novo plano de imagens, no presente trabalho foi definido um vetor que contém os números de 0 a 256 separados entre si pela razão:

$$passo = \frac{256}{2^n}$$

Onde n representa a quantidade de bits do novo plano.

No Matlab isso foi feito com as seguintes linhas de comando:

Código 14 – Definição dos valores dos novos pixels

```
passo = 256/2^n;
cores = 0:passo:256;
```

Fonte: Autoria Própria (2020)

Tomando como exemplo, n = 2, o vetor será composto por:

$$cores = [0 \quad 64 \quad 128 \quad 192 \quad 256\ ]$$

Posteriormente à definição do vetor de cores, a imagem foi percorrida e em cada pixel foi aplicada a seguinte equação:

$$pixel_{novo} = \frac{pixel_{antigo}}{passo} + 1$$

Onde $pixel_{novo}$ refere-se ao novo valor do pixel, $pixel_{antigo}$ refere-se ao antigo valor do pixel, e passo refere-se razão retromencionada, a equação é responsável por mapear os valores de cores antigos para novos valores de cores (Figura 12).

A ideia por trás dessa equação é relacionar o índice do vetor cores com o resto da divisão entre o valor do pixel e o passo, de forma que o valor do resto se refira ao índice que corresponde valor armazenado no vetor que representará a cor antiga no novo sistema. Soma-se o valor 1 para evitar indexação do zero, já que o Matlab começa a indexação de vetores em 1. Esse procedimento foi aplicado a cada um dos canais de cor da imagem.

A função a seguir realizado o procedimento discutido.

Código 15 – Função para quantificação dos planos de cores

```
function imagem = quantifica_plano(img, n)
   [i j k] = size(img);
   imagem = zeros(i, j, k, 'uint8');
   % Cria o vetor de cores
   passo = 256/2^n;
   cores = 0:passo:256;
   for x = 1 : i
      for y = 1 : j
         % Aplica a equação a cada pixel da imagem
          imagem(x, y, 1) = cores((img(x, y, 1) / passo) + 1);
          imagem(x, y, 2) = cores((img(x, y, 2) / passo) + 1);
          imagem(x, y, 3) = cores((img(x, y, 3) / passo) + 1);
      end
   end
end
```

Fonte: Autoria Própria (2020)

O seguinte trecho de código foi utilizado para ler a imagem do disco, aplicar a operação e plotar a imagem.

Código 16 – Código para a resolução da Questão 7

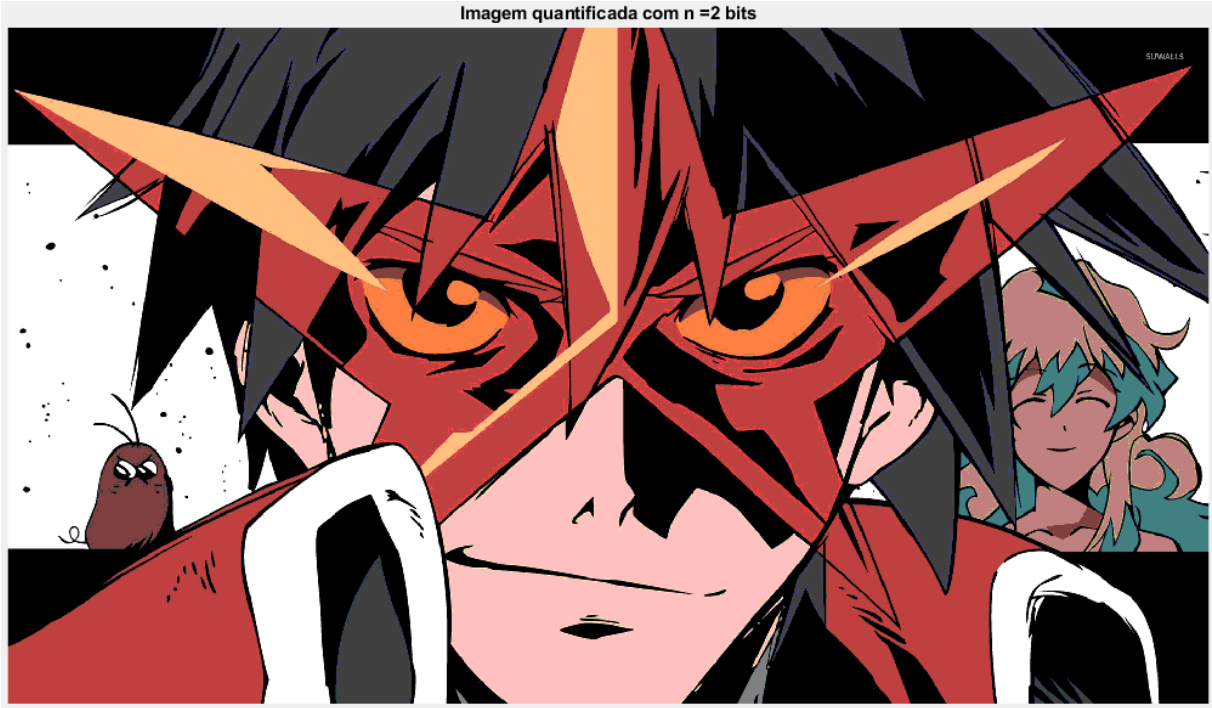
```
img = imread('D:\Estudos\Eng. de Computação\Processamento Digital de Imagens\Listas\Lista 2\Imagens\img.jpg');
bit = 4;
img_pb = quantifica_plano(img, bit);

figure();
imshow(img_pb);
```

```
title(strcat('Imagem quantificada com n = ', num2str(bit), ' bits'));
```

Fonte: Autoria Própria (2020)

Por fim, foram executados testes com a imagem para n = 2 e n = 4, os seguintes resultados foram obtidos.

Figura 13 – Imagem quantificada para n = 2 bits



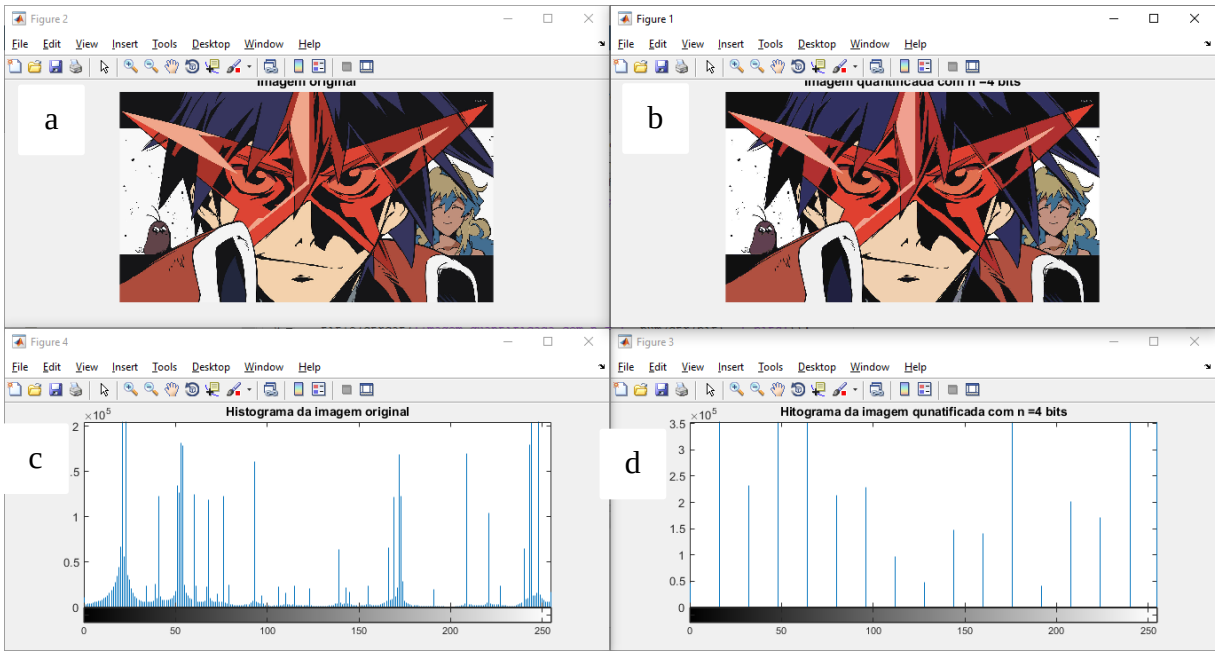
Fonte: Autoria Própria (2020)

Figura 14 – Imagem quantificada para n = 4 bits

Fonte: Autoria Própria (2020)

A Figura 14 em comparação com a imagem original (Figura 1) parecem visualmente semelhantes, porém a Figura 14 é formada por menos cores que a primeira. A figura 15a mostra a imagem original enquanto a figura 15c mostra o histograma daquela imagem, já a figura 15b traz a imagem com plano reduzido para 4 bits e a figura 15d traz o histograma daquela imagem. A partir dos histogramas, pode-se verificar que enquanto o histograma da imagem original apresenta todas as cores entre 0 e 256, o histograma da figura com plano reduzido apresenta valores apenas nas cores definidas para o mapeamento. Sendo assim, mesmo que as imagens sejam visualmente muito semelhantes os seus histogramas são diferentes. Com isso, pode-se verificar que a aplicação dessa redução pode representar muito bem uma imagem, reduzindo assim a quantidade de cores a serem utilizados e o posterior processamento para aquela imagem.

Figura 15 – Imagens e seus respectivos histogramas



Fonte: Autoria Própria (2020)

Referências Bibliográficas

GONZALEZ, Rafael C.; WOOD, Richard E.. **Processamento Digital de Imagens**. 3. ed. São Paulo: Pearson Pratice Hall, 2010.

GUIMARÃES, Luciano Macedo; CANDEIAS, Ana Lúcia Bezerra. Utilização de técnicas de processamento de imagens digitais para definição de contorno de reservatórios. **Simpósio Brasileiro de Sensoriamento Remoto**, Recife, p.6927-6934, abr. 2009.

KULKARNI, Nilima. Color Thresholding Method for Image Segmentation of Natural Images. **International Journal Of Image, Graphics And Signal Processing**, [s.l.], v. 4, n. 1, p.28-34, 3 fev. 2012. MECS Publisher. http://dx.doi.org/10.5815/ijigsp.2012.01.04.

MARQUES FILHO, Ogê; VIEIRA NETO, Hugo. **Processamento Digital de Imagens**, Rio de Janeiro: Brasport, 1999. ISBN 8574520098.

NOGUEIRA, Ricardo César de Almeida. **Análise de Conversão de Imagem Colorida para Tons de Cinza Via Contraste Percebido**. 2016. 46 f. TCC (Graduação) - Curso de Engenharia da ComputaÇÃo, Centro de Informática, Universidade Federal de Pernambuco, Recife, 2016.