UNIVERSIDADE DO ESTADO DO RIO GRANDE DO NORTE – UERN
Pró-Reitoria de Ensino e Graduação – PROEG
Departamento de Ciência da Computação
Campus de Natal – CaN

Daniel Teodolino Barbosa Torres
William Morais

Processamento de Imagens - Implementações

NATAL
2019

Sumário

1 MONOCROMATICA OU TONS DE CINZA

Consiste na manipulação cada pixels da imagem colorida de de tal forma que o valor de cada pixel se torno uma única amostra de um espaço de cores, variando entre o preto como a menor intensidade e o branco como maior intensidade, passando pelo cinza.

```python
def escalaCinza(img):

    rows = img.shape[0]
    cols = img.shape[1]

    cv2.imshow('Imagem Original', img)

    for i in range(rows):
        for j in range(cols):
            (b,g,r) = img[i,j]
            img[i,j] = ((b*0.114)+(g*0.587)+(r*0.299))

    return cv2.imshow('Tom de Cinza', img)
```

FIGURA 1

A figura 1 mostra como foi feita a implementação da imagem colorida em monocromática. Para isso, foi capturado o tamanho da matriz que representa a imagem (rows e cols), acessamos a matriz e percorremos cada pixel para capturar a intensidade das bandas red, green, blue (RGB ou BGR, para o Python).Efetuando o produto da soma (b*0,114)+(g*0,587)+(r*0,299) e atribuindo esse novo valor ao pixel, teremos a imagem em tons de cinza. A figura 2, mostra o resultado desse processamento.

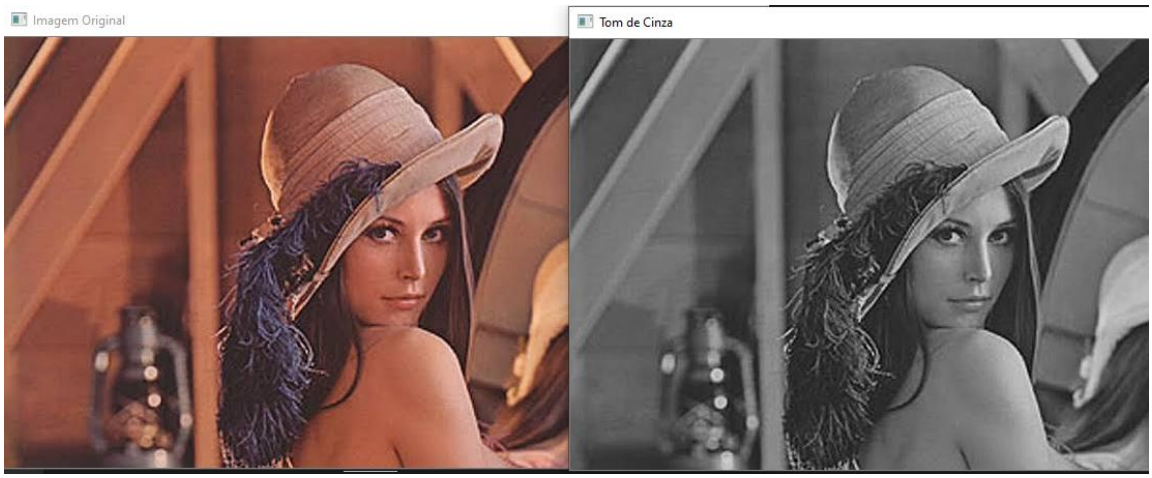

FIGURA 2

2 BINARIZAÇÃO

Consiste na manipulação da imagem em tons de cinza de tal forma que o valor de cada pixel, se menor que um valor de limiar, receberá um valor 0, caso contrario receberá 255.

```
def binarizacao(img2):

    rows = img2.shape[0]
    cols = img2.shape[1]
    limiar = 128 # 256/2

    cv2.imshow('Imagem Original', img2)

    for i in range(rows):
        for j in range(cols):
            px = img2[i,j]
            if (px < limiar):
                img2[i,j] = 0
            else:
                img2[i,j] = 255
    return cv2.imshow('Binarizacao', img2)
```

FIGURA 3

A figura 3 mostra como foi feita a implementação da imagem em tons de cinza em binária. Para isso, foi capturado o tamanho da matriz que representa a imagem (rows e cols), acessamos a matriz e percorremos cada pixel para capturar a intensidade desse pixel. Efetuamos a comparação desse pixel com o valor de limiar. Caso seja menor, o pixel recebe o valor 0, caso contrário recebe o valor 255. Isso tornará a imagem preto e branco. A figura 4 mostra o resultado desse processamento.

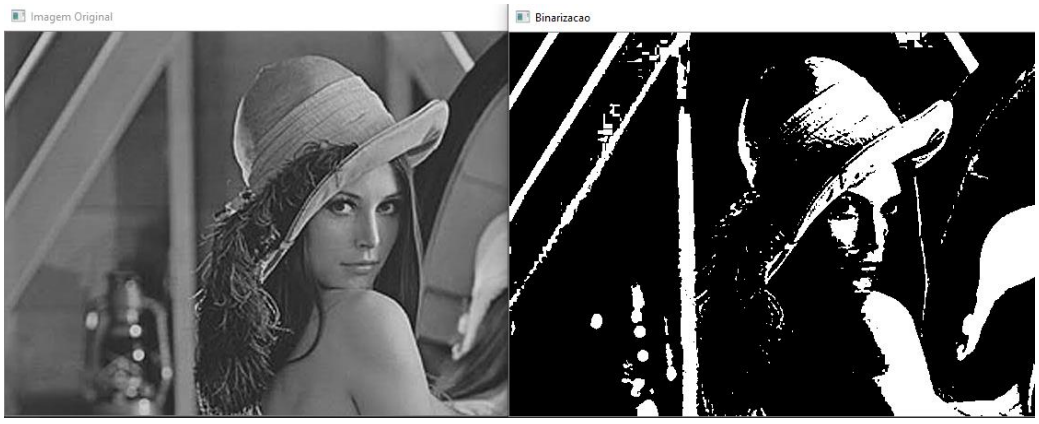

FIGURA 4

3 NEGATIVO

Consiste na manipulação da imagem em tons de cinza ou colorida de tal forma que o valor de cada pixel, receba o resultado da subtração 255-pixel.

```
def negativo(img2):

    rows = img2.shape[0]
    cols = img2.shape[1]

    cv2.imshow('Imagem Original', img2)

    for i in range(rows):
        for j in range(cols):
            img2[i,j] = ((255 - img2[i,j]))

    return cv2.imshow('Nagativo', img2)
```

FIGURA 5

A figura 5 mostra como foi feita a implementação da imagem em negativo. Para isso, foi capturado o tamanho da matriz que representa a imagem (rows e cols), acessamos a matriz e percorremos cada pixel para capturar a intensidade desse pixel. Efetuamos a subtração 255-pixel e atribuimos o novo valor ao pixel. A figura 6 mostra o resultado desse processamento.

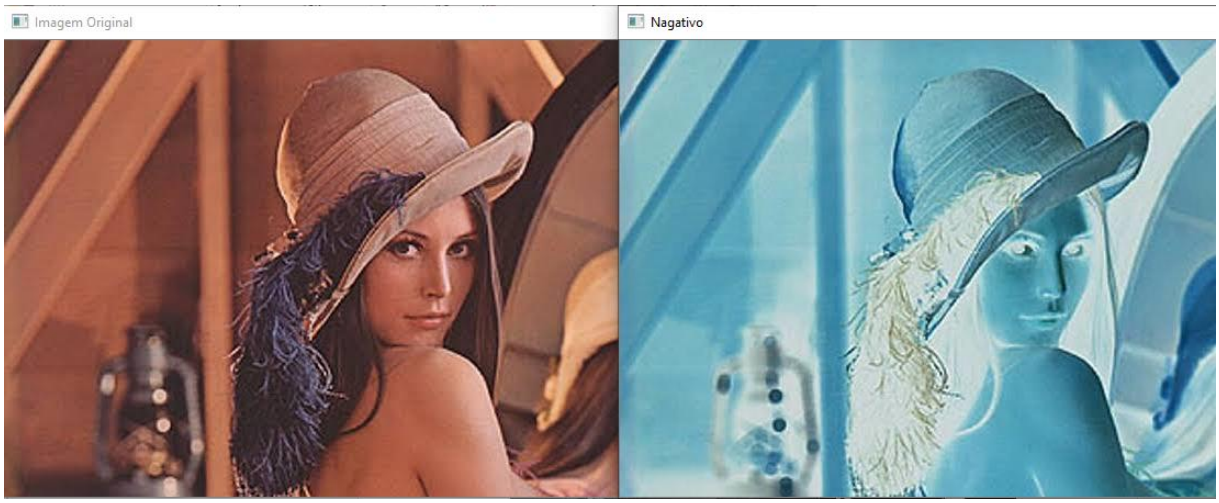

FIGURA 6

4 OPERAÇÕES ARITMÉTICA (SOMA, SUBTRAÇÃO, MULTIPLICAÇÃO E DIVISÃO)

Consiste na manipulação da imagem em tons de cinza ou colorida de tal forma que o valor de cada pixel, receba o resultado de uma operação aritmética (soma, subtração, multiplicação ou divisão). A operação pode ser feita entre a matriz que representa a imagem e um fator (valor definido) ou por uma outra matriz que representa uma outra imagem.

```
def operacaoMult(img2):

    rows = img2.shape[0]
    cols = img2.shape[1]

    cv2.imshow('Imagem Original', img2)

    fatorMult = 4

    for i in range(rows):
        for j in range(cols):
            px = img2[i,j]
            #(b,g,r) = img[i,j]
            if ((fatorMult * px) < 0):
                img2[i,j] = 0
            elif ((fatorMult * px) > 255):
                img2[i,j] = 255
            else:
                img2[i,j] = fatorMult * px

    return cv2.imshow('Multiplicacao Escalar', img2)
```

FIGURA 7

A figura 7 mostra como foi feita a implementação da operação aritmética multiplicação. Para isso, foi capturado o tamanho da matriz que representa a imagem (rows e cols), acessamos a matriz e percorremos cada pixel para capturar a intensidade desse pixel. Efetuamos a comparação: se o resultado da operação for menor que ZERO, o novo valor do pixel será ZERO, senao se o resultado da operação for maior que 255, o novo valor do pixel será 255, caso contrário, o valor do pixel será o resultado da operação aritmética. Abaixo exibimos o resultado das quatro operações fundamentais (figura 8 – multiplicação, . O código implementado segue a mesma lógica, sendo necessário modificar somente a operação aritmética.

FIGURA 8 - Multiplicação

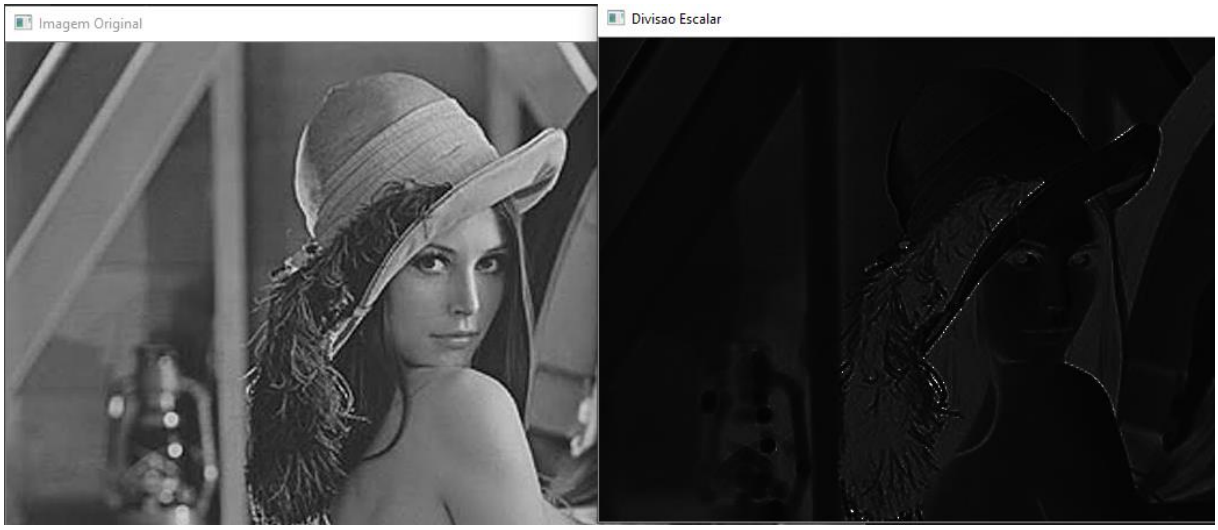

FIGURA 9 - Divisão

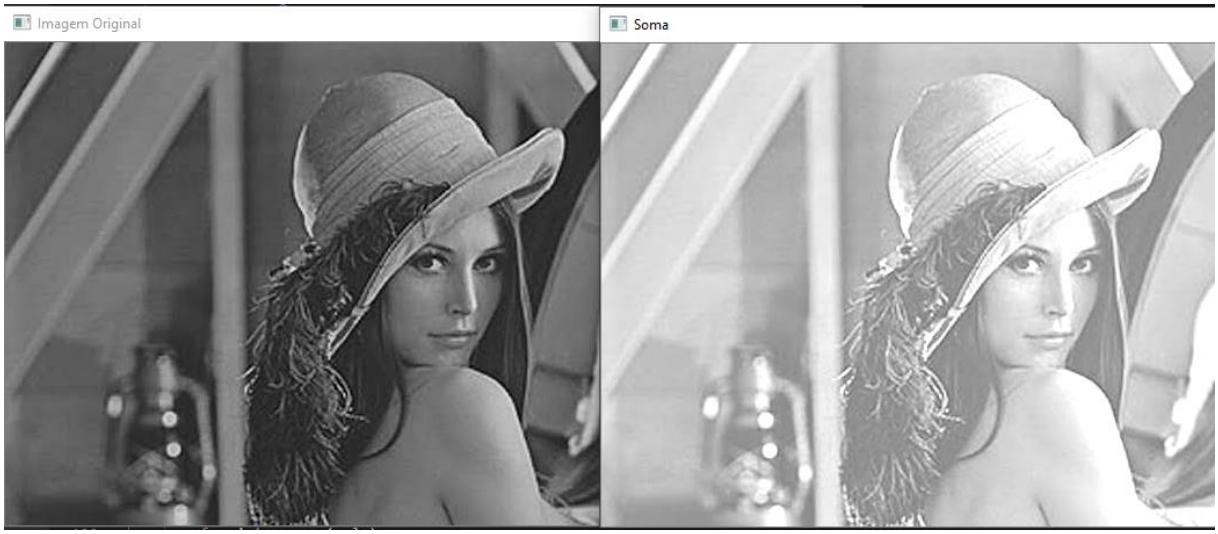

FIGURA 10 - Soma

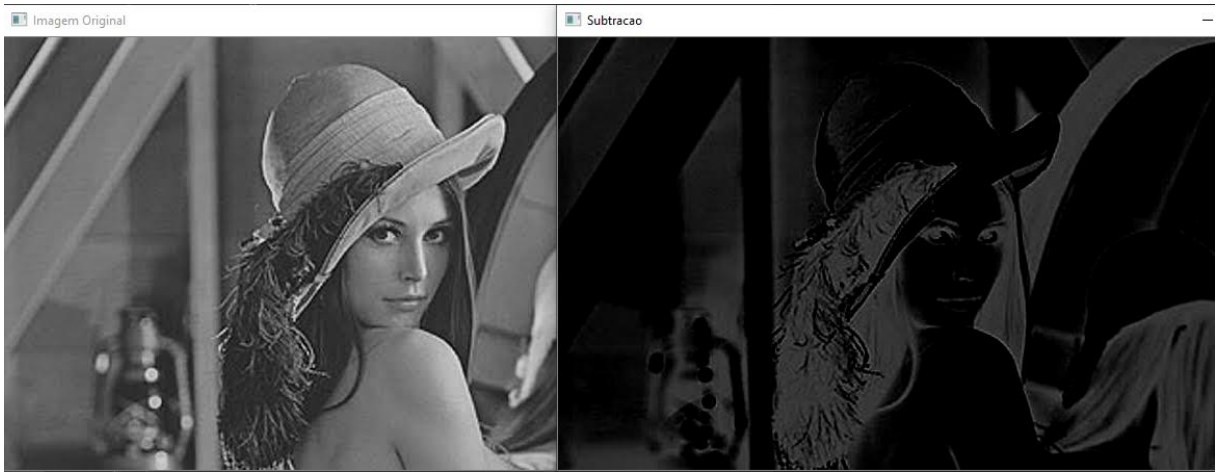

FIGURA 11 - Subtração

5 ALARGAMENTO DE CONTRASTE

Consiste em modificar a intensidade de um grupo de pixels que estão fora de uma escala conhecida.

```
def alargamentoContraste(img2):

    rows = img2.shape[0]
    cols = img2.shape[1]
    cv2.imshow('Imagem Original', img2)

    maior = img2[0,0]
    menor = img2[0,0]

    idmaior = 100 #depois solicitar dados ao usuario
    idmenor = 20

    for i in range(rows):
        for j in range(cols):
            px = img2[i,j]
            if (px > maior):
                maior = px
            elif (menor > px):
                menor = px

    for i in range(rows):
        for j in range(cols):
            img2[i,j] = (img2[i,j]-menor)*(idmaior-idmenor)/(maior-menor)+menor

    return cv2.imshow('Alargamento', img2)
```

FIGURA 12

A figura 12 mostra como foi feita a implementação do alargamento de contraste da imagem. Para isso, foi capturado o tamanho da matriz que representa a imagem (rows e cols). Foram criadas dias variáveis auxiliares, maior e menor, inicialmente inicializadas com ZERO. Essas variáveis, auxiliarão no processo de identificação do maior e menor pixel dentro da matriz que representa a imagem. As variáveis idmaior e idmenor, representam a escala que queremos alargar o contraste. A figura 13 mostra o resultado desse processamento.

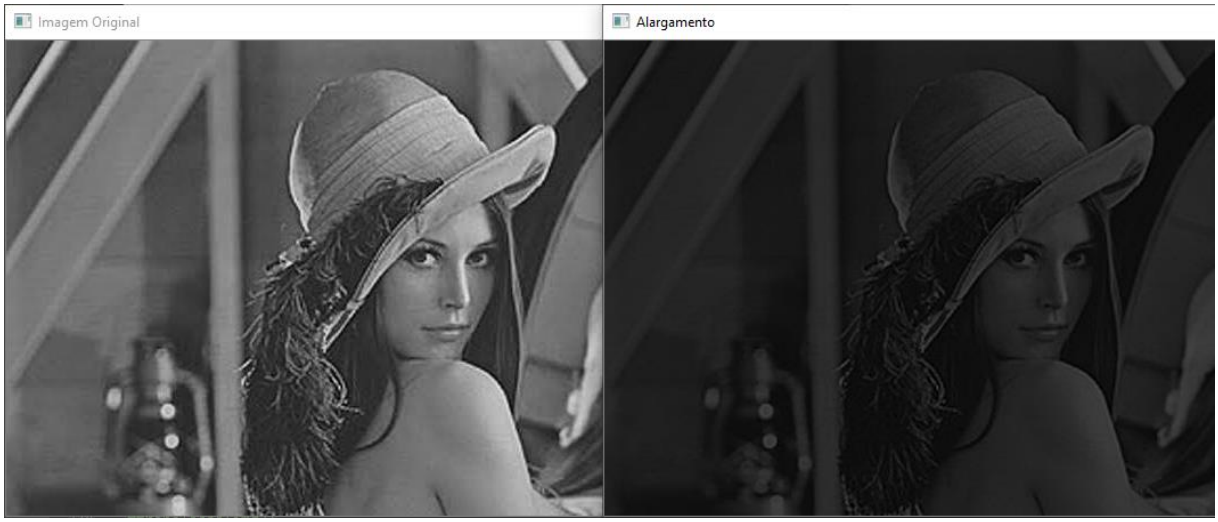

FIGURA 13

6 HISTOGRAMA

Consiste na contagem do número de intensidades dos pixels da matriz que representa a imagem.

```
def histograma(img2):

    rows = img2.shape[0]
    cols = img2.shape[1]

    cv2.imshow('Imagem Original', img2)

    escala = 255
    contador = []

    for k in range(escala):
        contador.append(0)

    for i in range(rows):
        for j in range(cols):
            for k in range(escala):
                px = img2[i,j]
                if (px == k):
                    contador[k] = contador[k] + 1

    #print(contador)
    cv2.imshow('Histograma', img2)
    x = np.arange(0, 255)
    y = contador
    plt.plot(x, y)
    plt.title("Histograma")
    plt.show()
```

FIGURA 14

A figura 14 mostra como foi feita a implementação do histograma da imagem. Para isso, foi capturado o tamanho da matriz que representa a imagem (rows e cols). Foi criado um vetor de 256 posições e inicalizado com ZERO, esse vetor servirá para armazenar a contagem das intensidades. Acessamos a matriz e percorremos cada pixel para contar a intensidade de cada pixel. Feito isso plotamos as informações em um gráfico onde o eixo x será a escala de 0 à 255 e o y a quantidade de cada uma das intensidades. A figura 15 mostra o resultado desse processamento.

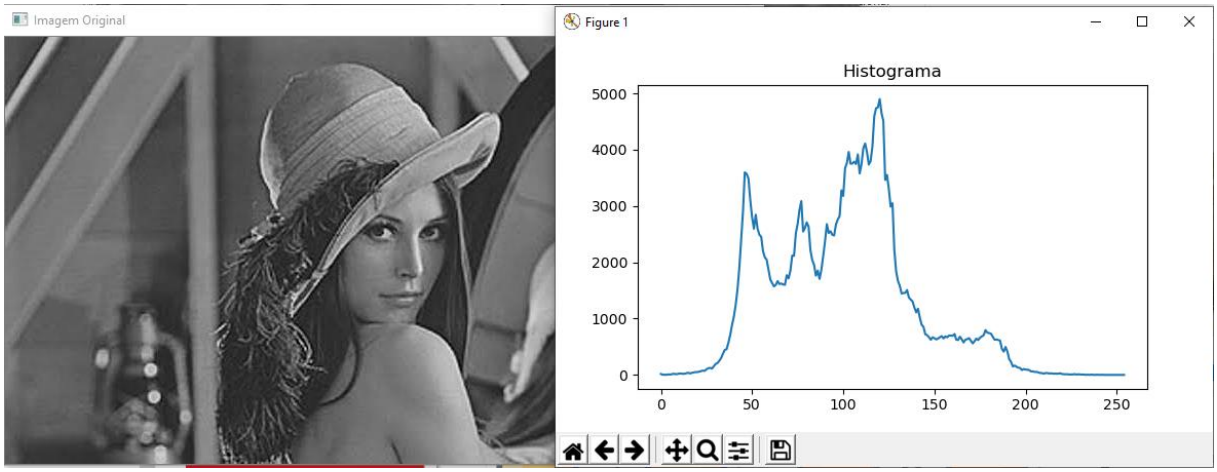
Imagem Original
Figure 1
Histograma
5000
4000
3000
2000
1000
0
0
50
100
150
200
250

FIGURA 15

7 ESPELHAMENTO

Consiste na modificação dos valores que representam as colunas na matriz da imagem.

```
def espelhamento(img2):

    rows = img2.shape[0]
    cols = img2.shape[1]

    cv2.imshow('Original', img2)

    for i in range (0, rows-1):
        for j in range(0,cols-1):
            esp[i, j] = img2[i ,(img2.shape[1]-1) - j]

    return cv2.imshow('Espelhamento', esp)
```

FIGURA 16

A figura 16 mostra como foi feita a implementação do espelhamento da imagem. Para isso, foi capturado o tamanho da matriz que representa a imagem (rows e cols). Percorremos a matriz, até o o tamanho -1, tanto na coluna, quanto na linha, e para cada pixel manipulamos o valor que representa a coluna. A figura 17 mostra o resultado desse processamento.

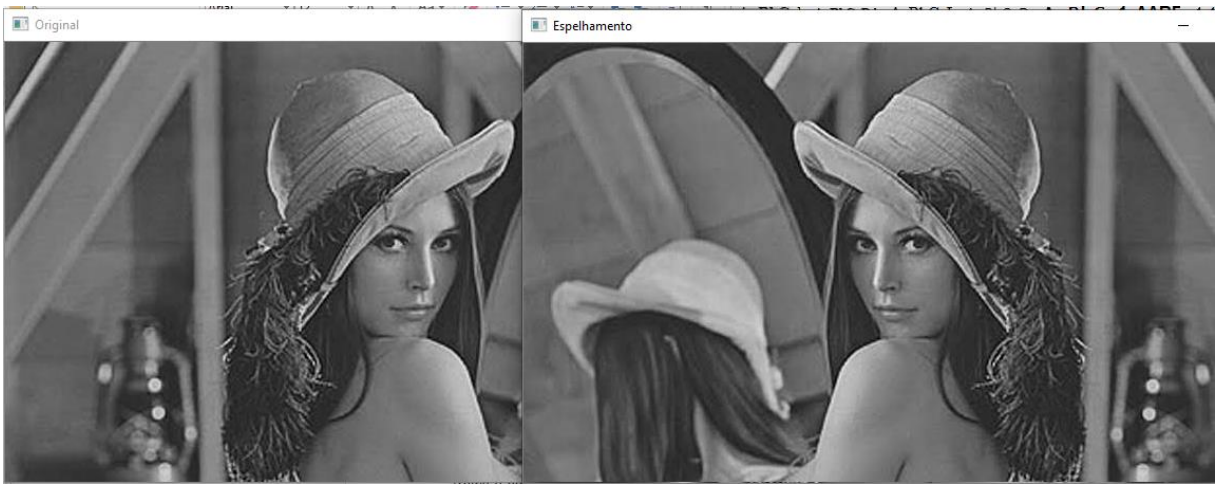

FIGURA 17

8 TRANSLAÇÃO

Consiste em deslocar a imagem em relação aos eixos x e y..

```
def translacao(img2):

    rows = img2.shape[0]
    cols = img2.shape[1]
    cv2.imshow('Imagem Original', img2)

    translacaoX = 250 #depois solicitar dados ao usuario
    translacaoY = 100 #sao valores que definem o fator de translacao
                      #nos eixos X e Y

    for i in range(rows):
        for j in range(cols):
            if (((i+translacaoY) < rows) and
            ((j+translacaoX) < cols) and
            ((i+translacaoY) >= 0) and
            ((j+translacaoX) >= 0)):
                trans[i,j] = img2[(i+translacaoY),(j+translacaoX)]
            else:
                trans[i,j] = 0

    return cv2.imshow('Translacao', trans)
```

FIGURA 18

A figura 18 mostra como foi feita a implementação da translação da imagem. Para isso, foi capturado o tamanho da matriz que representa a imagem (rows e cols). Atribuimos um valor de translação tanto pata o eixo X quanto para o eixo Y. Percorremos a matriz e se o valor for maior que ZERO e menor que a posição atual, atualizamos o valor do pixel, caso contraio atribuimos o valor zero (para imprimir a cor preto no espaço deslocado). A figura 19 mostra o resultado desse processamento.

FIGURA 19

9 ESCALA

Consiste em aumentar ou diminuir o valor das linhas e colunas que representam o pixel da imagem.

```python
def escala(img2): #NAO ERA PARA FAZER! MAS EU FIZ

    rows = img2.shape[0]
    cols = img2.shape[1]
    #cv2.imshow('Imagem Original', img2)

    escalaX = 2 #depois solicitar dados ao usuario
    escalaY = 2 #sao valores que definem a faixa que nao sera preto.
                #tudo fora dessa faixa sera preto.

    for i in range(rows):
        for j in range(cols):
            if (((i*escalaY) < rows) and ((j*escalaX) < cols) and ((i*escalaY) >= 0) and ((j*escalaX) >= 0)):
                img2[i,j] = img2[(i*escalaY),(j*escalaX)]

    return cv2.imshow('Escala', img2)
```

FIGURA 20

A figura 20 mostra como foi feita a implementação da escala da imagem. Para isso, foi capturado o tamanho da matriz que representa a imagem (rows e cols). E os fatores de escala para X e Y. Percorremos a matriz e se o valor for maior que ZERO e menor que a posição atual, atualizamos o valor do pixel, caso contraio o pixel permanece inalterado. A figura 21 mostra o resultado desse processamento.

FIGURA 21

10 FATIAMENTO

Consiste em modificar o valor de um pixel que estão foram de uma faixa definida.

```
def fatiamento(img2):

    rows = img2.shape[0]
    cols = img2.shape[1]
    cv2.imshow('Imagem Original', img2)

    valorMinimo = 150 #depois solicitar dados ao usuario
    valorMaximo = 150  #sao valores que definem a faixa que nao sera preto.
                       #tudo fora dessa faixa sera preto.

    for i in range(rows):
        for j in range(cols):
            px = img2[i,j]
            if ((valorMinimo >= px) and (img2[i,j] <= valorMaximo)):
                img2[i,j] = px
            else:
                img2[i,j] = 255

    return cv2.imshow('Fatiamento', img2)
```

FIGURA 22

A figura 22 mostra como foi feita a implementação da escala da imagem. Para isso, foi capturado o tamanho da matriz que representa a imagem (rows e cols). E os valores min. e max., os valores fora dessa escala serão atualizados para a cor preto e os demais permanecerão inalterados. A figura 23 mostra o resultado desse processamento.

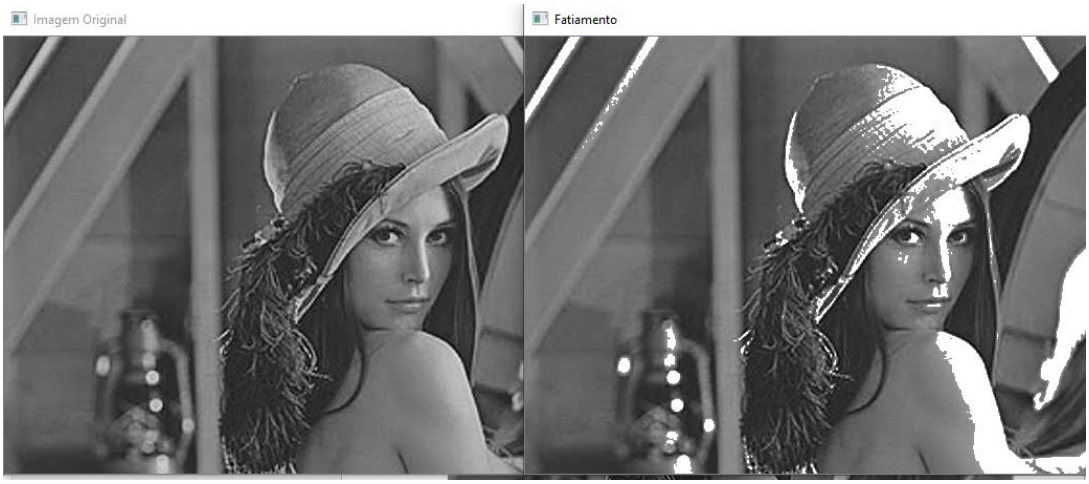

FIGURA 21

11 RUÍDO SAL E PIMENTA

Consiste em identificar pixels igual a ZERO e UM e realçar esses pixels.

```
def salEPimenta(img2):

    rows = img2.shape[0]
    cols = img2.shape[1]

    cv2.imshow('Imagem Original', img2)

    for i in range(rows):
        for j in range(cols):
            px = img2[i,j]
            aleatorio = randint(0, 30)
            #print ("Aleatorio >> ", aleatorio)
            if (aleatorio == 0):
                img2[i,j] = 0
            elif (aleatorio == 1):
                img2[i,j] = 255
            elif (aleatorio >= 2):
                img2[i,j] = px

    return cv2.imshow('Sal e Pimenta', img2)
```

FIGURA 22

A figura 22 mostra como foi feita a implementação do ruído sal e pimenta na imagem. Para isso, foi capturado o tamanho da matriz que representa a imagem (rows e cols) e gerados valores aleatórios entre 0 e 30, esses valores serão comparados com os pixels da matriz e caso seja igual a 0, o pixel será realçado com ZERO, caso seja igual a 1, o pixel será realçado com 255, e caso seja maior que 1, permanecerá com o valor que posssui. A figura 23 mostra o resultado desse processamento.

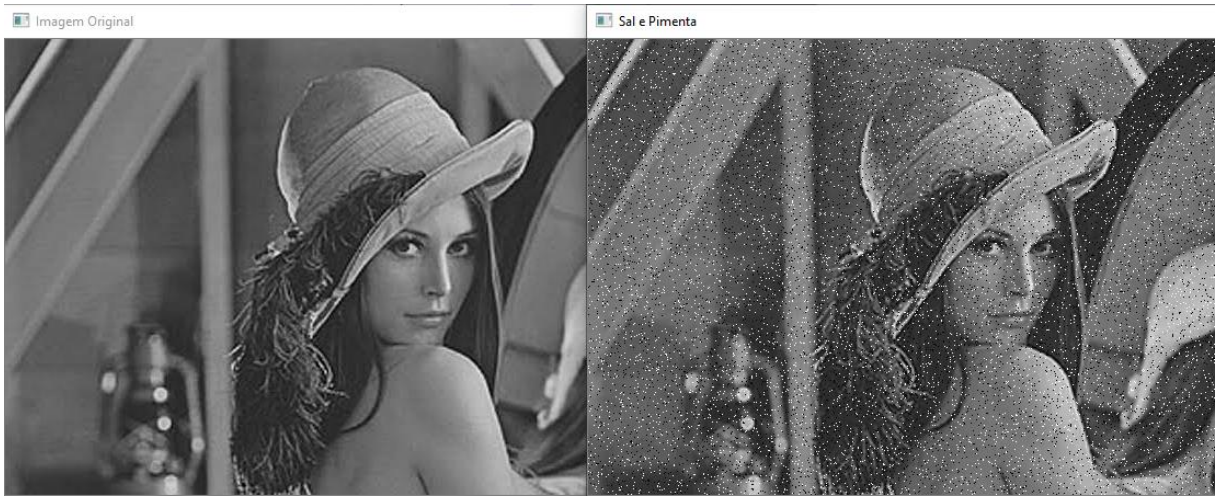

FIGURA 21

12 PASSA-BAIXA (3, 5, 7)

O filtro passa-baixa é mais utilizado em processamentos específicos, principalmente para suavização de imagens. Podem ser com máscara de 3x3, 5x5 ou 7x7.

```
def passaBaixo5(img2):

    soma = 0
    cont = 0

    #img2 = cv2.imread('lena.jfif', 0)
    cv2.imshow('Original', img2)

    for x in range (0, img2.shape[0]-4):
        for y in range(0, img2.shape[1]-4):
            while(cont < 2):
                for i in range(-1, 3):
                    for j in range(-1, 3):
            #            img[(x+i),(y+j)] = 255
                        if cont == 0:
                            soma = soma + img2[(x+i),(y+j)]
                        else:
                            img2[(x+i),(y+j)] = soma/16

                cont = cont + 1

            cont = 0
            soma = 0

    return cv2.imshow('Passa Baixo 5', img2)
```

FIGURA 24

A figura 24 mostra como foi feita a implementação da passa-baixa da imagem. Estamos demostrando o método com a máscara 5x5. Para a máscara 3x3 ou 7x7, basta modificar os limites dos laços for e a soma para cálculo da média. A figura 25 mostra o resultado desse processamento.

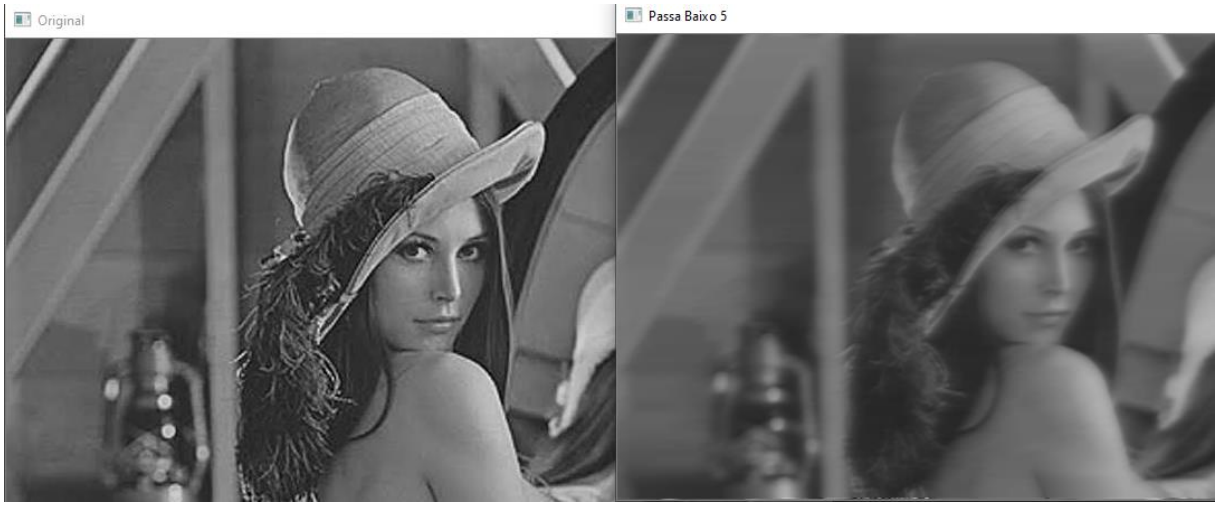

FIGURA 25

13 PASSA-ALTA (3, 5, 7)

O filtro passa-alta é mais utilizado em processamentos específicos, principalmente para realçar detalhes. Podem ser com máscara de 3x3, 5x5 ou 7x7.

```
import cv2

aux = 0
soma = 0

img = cv2.imread('lena.jfif', 0)
cv2.imshow('janela original', img)
img_passaAlta = img.copy()

for x in range (0, img.shape[0]-5):
    for y in range(0, img.shape[1]-5):
        for i in range(0, 5):
            for j in range(0, 5):
                #img[(x+i),(y+j)] = 255
                if i == 2 and j == 2:
                    soma = soma + (24*img[(i+x),(j+y)])
                else:
                    soma = soma + (-1*img[(i+x),(j+y)])


        if soma < 0:
            img_passaAlta[2+x,2+y] = 0
        elif soma > 255:
            img_passaAlta[2+x,2+y] = 255
        else:
            img_passaAlta[2+x,2+y] = soma

        soma = 0


cv2.imshow('janela resultado', img_passaAlta)
cv2.waitKey(0)
```

FIGURA 25

A figura 25 mostra como foi feita a implementação da passa-alta da imagem. Estamos demostrando o método com a máscara 5x5. A figura 26 mostra o resultado desse processamento.

FIGURA 26

14 EQUALIZAÇÃO DO HISTOGRAMA – CORRIGIR

Para equalizar o histograma é necessário identificar o pixel de maior intensidade e calcular a probabilidade de cada pixel. Para isso multiplicamos o pixel de maior intensidade pela quantidade de ocorrência de cada intensidade e dividimos pelo tamanho da imagem. Feito isso, calculamos a probabilidade acumulada, que consiste em somar a probabilidade atual com a probabilidade anterior.

```
#laco p calcular o valor acumulado hist[k]
maiorIntensidade = len(probabilidade)-1
for k in range(escala):
    probabilidade.append((maiorIntensidade*contPixel[k])/pxTotal)

print("\n\nVetor de probabilidade >> ", probabilidade)

for k in range(escala):
    if (k == 0):
        probabilidadeAcumulada.append(probabilidade[k])
    else:
        probabilidadeAcumulada.append(math.ceil(probabilidadeAcumulada[k-1]+probabilidade[k]))

    #contInteracao = contInteracao + probabilidade[k]

print("\n\nVetor de probabilidade acumulada >> ", probabilidadeAcumulada)

x = np.arange(0, 255)
y = contPixel
plt.plot(x, y)
plt.title("Equalizacao do Histograma")
plt.show()

for i in range(rows):
    for j in range(cols):
        for k in range(escala):
            img2[i,j] = probabilidadeAcumulada[k]


return cv2.imshow('Equalizacao do Histograma', img2)
```

FIGURA 27

A figura 27 mostra como foi feita a implementação da da equalização do histograma.

15 LIMIARIZAÇÃO - CORRIGIR

Consiste em calcular a quantidade de tons com base em uma quantidade definida de limiares.

```
def limiarizacao(img2): #possui varios limiares definidos pelo usuario
    rows = img2.shape[0]
    cols = img2.shape[1]
    cv2.imshow('Imagem Original', img2)
    limiares = int(input("Digite um valor para o Limiar >> "))
    cores = limiares+1
    tons = 255/cores
    cor = tons/2

    vetorLimiares = []
    vetorCores = []

    for k in range(cores):
        if (k==0):
            vetorLimiares.append(tons)
        else:
            vetorLimiares.append(vetorLimiares[k-1]+tons)

    for w in range(cores):
        if (w==0):
            vetorCores.append(int(cor))
        else:
            vetorCores.append(int(vetorCores[w-1]+cor))

    for i in range(rows):
        for j in range(cols):
            for x in range(cores):
                if img2[i,j] < vetorLimiares[x]:
                    img2[i,j] = vetorCores[x]
                    break

    return cv2.imshow('Limiarizacao', img2)
```

FIGURA 28

A figura 28 mostra como foi feita a implementação da limiarização da imagem e a figura 29 mostra o resultado desse processamento para um limiar de 3.

FIGURA 29

16 RUÍDO GAUSS - CORRIGIR

Consiste em um filtro para reduzir ruídos.

```
def gauss(img2):

    rows = img2.shape[0]
    cols = img2.shape[1]

    cv2.imshow('Imagem Original', img2)

    for i in range(rows):
        for j in range(cols):
            aleatorio = (randint(0,10))
            print ("Aleatorio >> ", aleatorio)
            if (aleatorio == 0):
                img2[i,j] = random()%255
            elif (aleatorio >= 1):
                img2[i,j] = img2[i,j]


    return cv2.imshow('Gauss', img2)
```

FIGURA 30

A figura 30 mostra como foi feita a implementação do método gauss e a figura 31 mostra o resultado desse processamento.

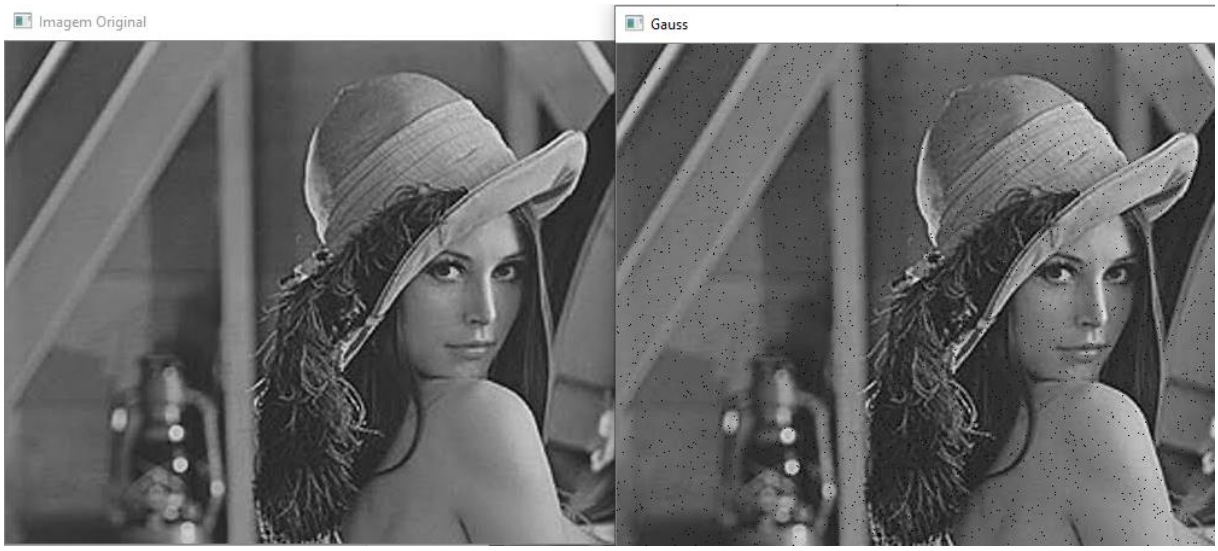

FIGURA 31