Atividade 1 – Processamento Digital de Sinais
27/09/2023
Processamento de Imagens

Aluno: Gabriel Almeida Santos de Oliveira.
Nº de matrícula: 2021000042.

faça programas (scripts) em Matlab (ou phyton) para aplicar os processamentos digitais abaixo. O trabalho deve possuir uma explicação básica do processamento (um parágrafo), o código em Matlab as imagens (entrada e saída) resultantes do processamento.

1. Binarização de uma imagem

A binarização de uma imagem é um processamento simples, toda imagem é constituída pela junção de pixels distribuídos no espaço, estes pixels, para uma imagem em escala de cinza, podem adotar um número inteiro positivo entre 0 e 255, sendo 0 preto e 255 branco. Em uma Imagem Colorida, cada pixel é formado pela junção de 3 inteiros (um para a cor azul, outro para vermelho e o terceiro para verde), variando também entre 0 e 255. Para realizar a binarização de uma imagem, basta analisar pixel a pixel e redefini-lo como 0 caso o valor médio do mesmo esteja abaixo de um certo threshold (para o algoritmo realizado escolheu-se 140, que está próximo da metade de 255), e 255 caso acima, abaixo está o código feito seguido da imagem obtida:

```
def show_image(img_obj, title=None):
    img_resized = ResizeWithAspectRatio(img_obj, 480)
    if title:
        cv.imshow(title, img_resized)
    else:
        cv.imshow("image", img_resized)
    cv.waitKey(0)
    cv.destroyAllWindows()


#1. Binarização de uma imagem
def image_binarization(image_file: str):
    #leitura da imagem
    img = cv.imread(image_file, cv.IMREAD_UNCHANGED)
    show_image(img, "Imagem Original")

    rows, cols, _ = img.shape

    for row in range(rows):
        for col in range(cols):
            pixel = sum(img[row,col])/3
            if pixel >= 140: #127
                img[row][col] = [255, 255, 255]
            else:
                img[row][col] = [0, 0, 0]

    print("\nImagem depois do filtro: ")
    show_image(img)
```

Imagem Original

Imagem Processada

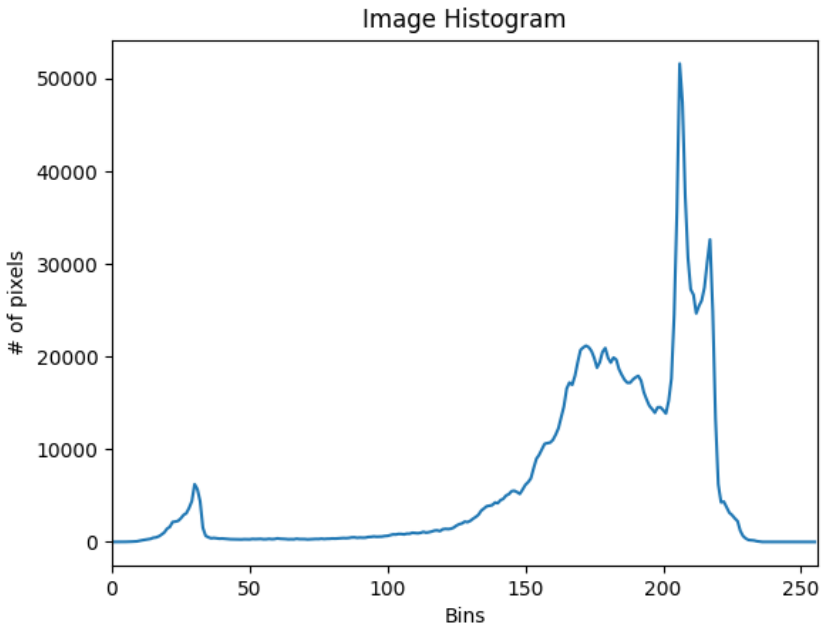
2. histograma de uma imagem em escala de cinza

O histograma consiste é uma forma de representar e analisar os pixels de uma imagem, os mesmo serão divididos pelo seu valor e será a quantidade de pixels com aquele valor. Pode observar, pelo histograma, se há muitos pixels de cores mais claras (valores próximo de 255), ou de cores mais escuras, ou se a imagem tem um bom contraste (pixels mais igualmente distribuídos pelo eixo horizontal).

```
def show_bw_histogram(img_obj):
    gray_hist = cv.calcHist([img_obj], [0], None, [256], [0, 256])

    plt.figure()
    plt.title('Image Histogram')
    plt.xlabel('Bins')
    plt.ylabel('# of pixels')
    plt.plot(gray_hist)
    plt.xlim([0, 256])
    plt.show()


#2. histograma de uma imagem em escala de cinza
def black_white_img_histogram(image_file: str):
    img = cv.imread(image_file)
    print("Imagem original: ")
    show_image(img)

    img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    print("Imagem Preto e Branco: ")
    show_image(img)

    show_bw_histogram(img)
```

Imagem Original

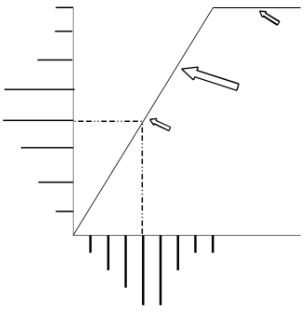
Imagem Processada



3. alargamento de contraste:

a) linear

Para se aumentar o contraste é necessário aumentar o espaço de distribuição dos pixels no histograma, para o método linear, isso é atingido multiplicando cada pixel por uma constante, de forma que o valor do novo pixel pode ser obtido pela imagem de uma reta como na imagem abaixo:

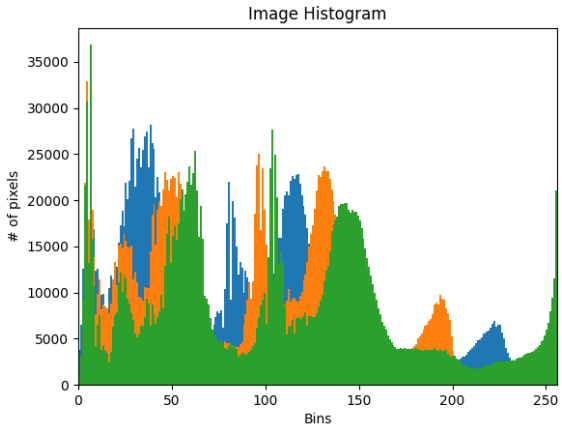


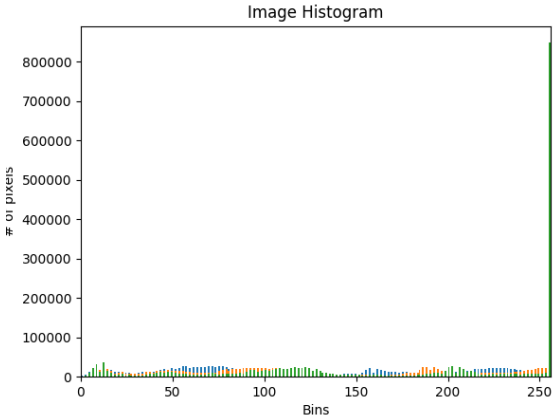
Segue abaixo o código escrito:

```
def map_value(var, var_min, var_max, ret_min, ret_max):
    return (ret_max - ret_min)*((var - var_min)/(var_max - var_min)) + ret_min


def show_color_histogram(img_obj):
    B, G, R = cv.split(img_obj)

    plt.figure()
    plt.title('Image Histogram')
    plt.xlabel('Bins')
    plt.ylabel('# of pixels')
    plt.hist(B.ravel(), 256, [0,256])
    plt.hist(G.ravel(), 256, [0,256])
    plt.hist(R.ravel(), 256, [0,256])
    plt.xlim([0, 256])
    plt.show()


#3.a) Alargamento de Contraste linear
def contrast_linear(image_file, porcentagem):
    angulo = map_value(porcentagem, 0, 100, 0, 90)
    img = cv.imread(image_file, cv.IMREAD_UNCHANGED)
    show_image(img)
    show_color_histogram(img)

    table_pixels = {}
    for ind in range(256):
        y_scale = round(ind*math.tan(angulo * (math.pi / 180)))
        if y_scale <= 255:
            table_pixels[ind] = np.uint8(y_scale)
        else:
            table_pixels[ind] = np.uint8(int(255))

    rows,cols,_ = img.shape

    for row_ind in range(rows):
        for col_ind in range(cols):
            #pega o valor correspondente salvo na tabela
            val_B = table_pixels[img[row_ind][col_ind][0]]
            val_G = table_pixels[img[row_ind][col_ind][1]]
            val_R = table_pixels[img[row_ind][col_ind][2]]

            img.itemset((row_ind, col_ind, 0), val_B) #Set B
            img.itemset((row_ind, col_ind, 1), val_G) #Set G
            img.itemset((row_ind, col_ind, 2), val_R) #Set R

    show_image(img)
    show_color_histogram(img)
```

Imagem Original

Imagem Processada





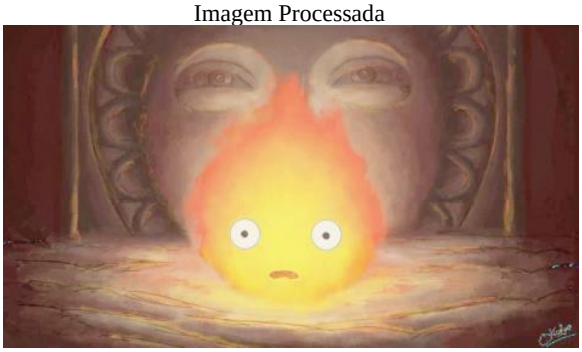
b) Logarítmico

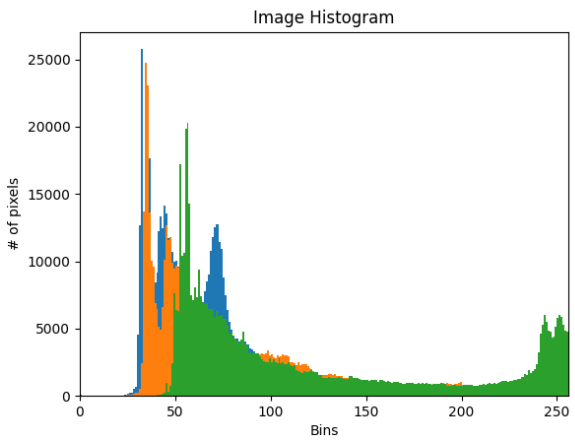
O efeito logarítmico funciona a partir do mesmo principio da mudança de contraste linear, ao efetuar uma operação matemática logarítmica em cada pixel, se obtém uma nova tabela de matrizes onde os pixels com maiores valores são menos alterados do que aqueles com menor valor.

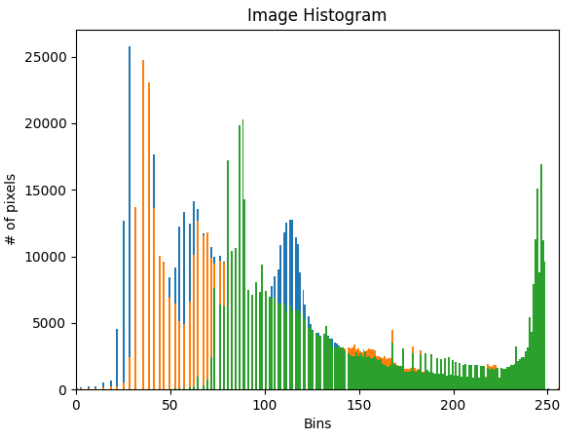
```python
#3.b) contraste logaritimico
def contrast_logaritmic(image_file, porcentagem):
    taxa = map_value(porcentagem, 0, 100, 230, 250)
img = cv.imread(image_file, cv.IMREAD_UNCHANGED)
show_image(img)
show_color_histogram(img)

table_pixels = {}
table_pixels[0] = 0
for ind in range(1,256):
    table_pixels[ind] = np.clip(np.uint8(np.log10(ind/(taxa/10))*taxa),0,255)

rows,cols,_ = img.shape

for row_ind in range(rows):
    for col_ind in range(cols):
        #pega o valor correspondente salvo na tabela
        val_B = table_pixels[img[row_ind][col_ind][0]]
        val_G = table_pixels[img[row_ind][col_ind][1]]
        val_R = table_pixels[img[row_ind][col_ind][2]]

        img.itemset((row_ind, col_ind, 0), val_B) #Set B
        img.itemset((row_ind, col_ind, 1), val_G) #Set G
        img.itemset((row_ind, col_ind, 2), val_R) #Set R

show_image(img)
show_color_histogram(img)
```

Imagem Original

Imagem Processada







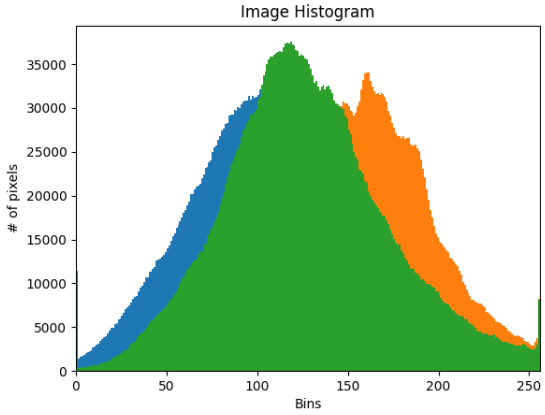
c) Quadrático

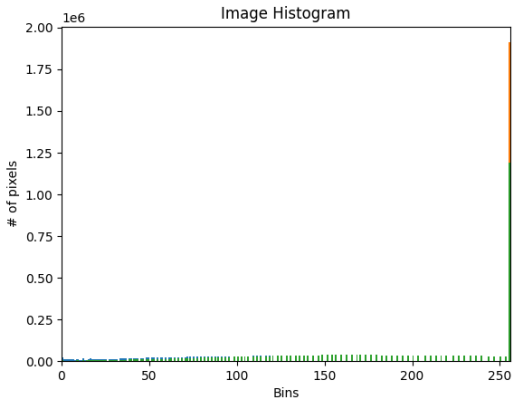
O contraste quadrático segue a mesma lógica dos anteriores, com a diferença que a equação que realoca o valor dos pixels é uma função quadrática. O efeito disso é que os pixels de maior valor tem uma taxa de variação maior, enquanto os pixels de menor valor são mais brandamente afetados. Segue o código abaixo

```python
#3.c)quadratico
def contrast_quadratic(image_file, porcentagem):
    taxa = map_value(porcentagem, 0, 100, 0.0001, 0.02)
    img = cv.imread(image_file, cv.IMREAD_UNCHANGED)
    show_image(img)
    show_color_histogram(img)

    table_pixels = {}
    for ind in range(256):
        y_scale = round(taxa*ind*ind)
        if y_scale <= 255:
            table_pixels[ind] = np.uint8(y_scale)
        else:
            table_pixels[ind] = np.uint8(int(255))

    rows,cols,_ = img.shape

    for row_ind in range(rows):
        for col_ind in range(cols):
            val_B = table_pixels[img[row_ind][col_ind][0]]
            val_G = table_pixels[img[row_ind][col_ind][1]]
            val_R = table_pixels[img[row_ind][col_ind][2]]

            img.itemset((row_ind, col_ind, 0), val_B) #Set B
            img.itemset((row_ind, col_ind, 1), val_G) #Set G
            img.itemset((row_ind, col_ind, 2), val_R) #Set R

    show_image(img)
    show_color_histogram(img)
```

Imagem Original

Imagem Processada





d) Exponencial

A função exponencial segue a mesma lógica das anteriores, a diferença é que a mesma irá acentuar ainda mais a variação dos pixels de valores mais altos. Segue o código abaixo:

```python
#3.d)exponencial
def contrast_exponencial(image_file, porcentagem):
    taxa = 56 - map_value(porcentagem, 0, 100, 10, 46)
    #quanto maior a taxa menor é o crescimento da exponencial
    img = cv.imread(image_file, cv.IMREAD_UNCHANGED)
    show_image(img)
    #show_color_histogram(img)

    table_pixels = {}
    for ind in range(256):
        y_scale = round((np.exp(ind/taxa)-1))
        if y_scale <= 255:
            table_pixels[ind] = np.uint8(y_scale)
        else:
            table_pixels[ind] = np.uint8(int(255))

    rows,cols,_ = img.shape

    for row_ind in range(rows):
        for col_ind in range(cols):
            #pega o valor correspondente salvo na tabela
            val_B = table_pixels[img[row_ind][col_ind][0]]
            val_G = table_pixels[img[row_ind][col_ind][1]]
            val_R = table_pixels[img[row_ind][col_ind][2]]

            img.itemset((row_ind, col_ind, 0), val_B) #Set B
            img.itemset((row_ind, col_ind, 1), val_G) #Set G
            img.itemset((row_ind, col_ind, 2), val_R) #Set R

    show_image(img)
    #show_color_histogram(img)
```

Imagem Original

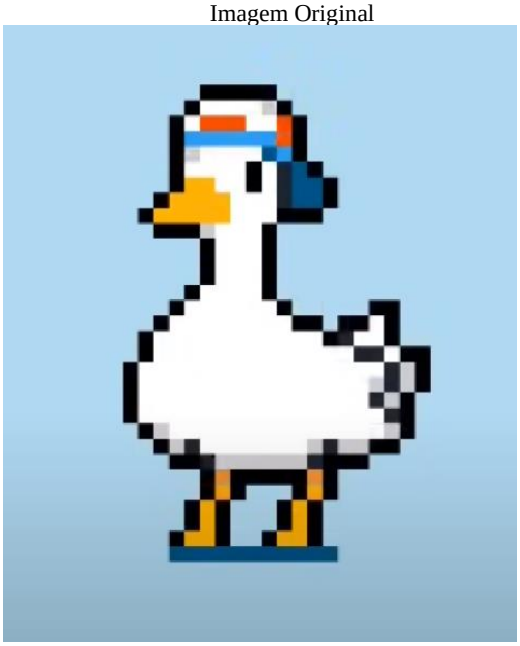
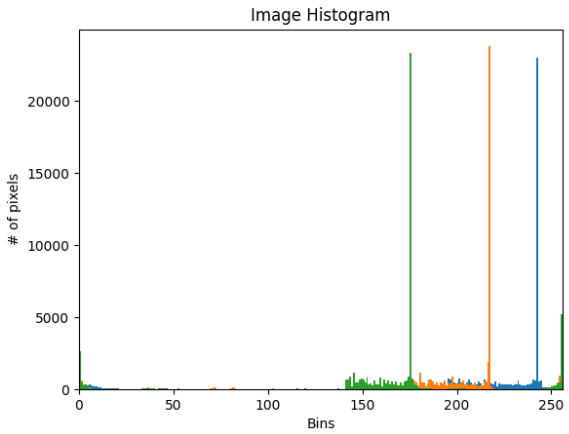
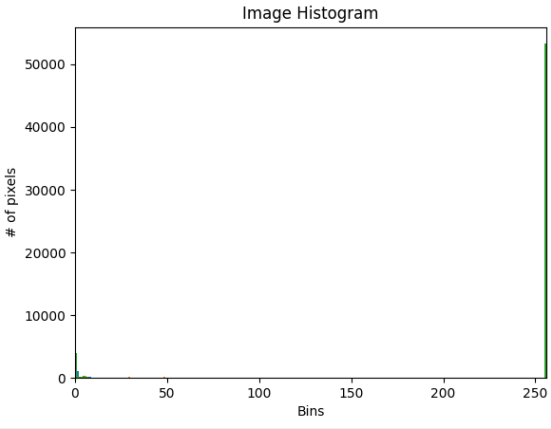


Imagem Processada





4. Equalização de contraste por histograma

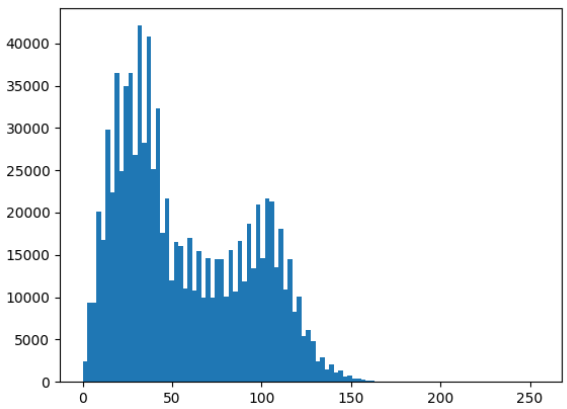
Para melhorar o contraste de uma imagem, o objetivo é melhor distribuir no histograma, isto é, evitar o acumulo grande de pixels de cor/intensidade similar (o que se traduz no histograma em picos localizados), a equalização do histograma realiza essa operação de distribuir mais uniformemente os pixels pelo histograma da imagem. Segue o código abaixo:

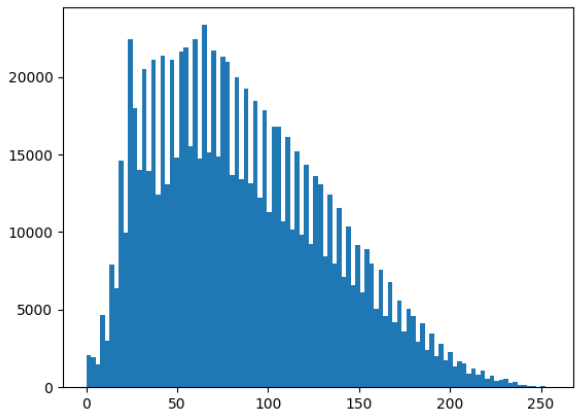
```
def hist_equalization(image_file):
    img = cv.imread(image_file, 1)
    img = ResizeWithAspectRatio(img, 480)
    cv2.imshow("original", img)
    plt.hist(img.flat, bins=100, range=(0, 255))
    plt.show()
```

```
lab_img = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
l,a,b = cv2.split(lab_img)

equa = cv2.equalizeHist(l)

merged_lab_img = cv2.merge((equa,a,b))
#color_equa_img = cv2.cvtColor(merged_lab_img, cv2.COLOR_LAB2BGR)
#cv2.imshow("equalized", color_equa_img)

clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8,8))
clahe_img = clahe.apply(l)
clahe_lab_img = cv2.merge((clahe_img, a, b))
clahe_img = cv2.cvtColor(clahe_lab_img, cv2.COLOR_LAB2BGR)
cv2.imshow("clahe", clahe_img)
plt.hist(clahe_img.flat, bins=100, range=(0, 255))
plt.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Imagem Original

Imagem Processada



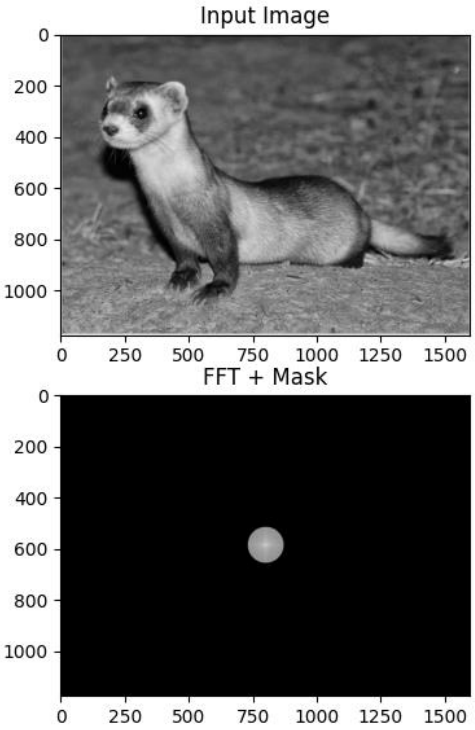


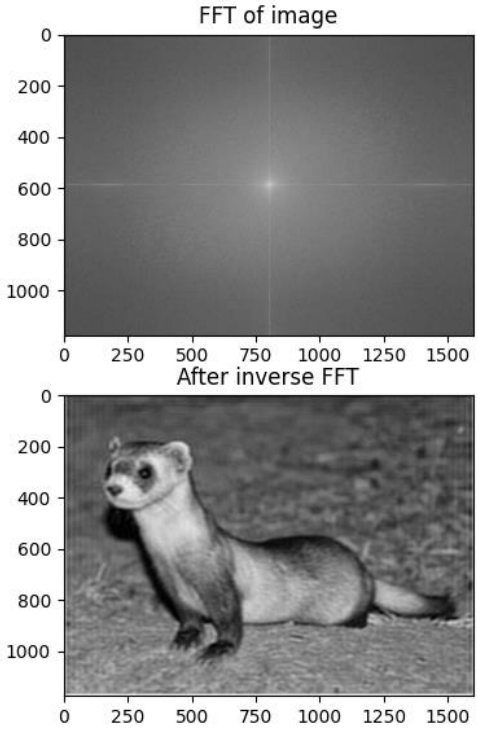
5. Aplicação de filtro passa-baixa (média, gaussiano ou fourier)

O filtro de passa baixa possui como objetivo filtrar as porções da imagem que possuem alta frequência, permitindo somente a passagem de regiões com baixa frequência. Em uma imagem, alta frequência corresponde a grande quantidade de informações, ou muitas mudanças bruscas de cor, cantos, quinas regiões detalhadas etc., o efeito do filtro passa-baixa é essencialmente borrar essas regiões, reduzindo a riqueza de detalhes. Segue abaixo o código escrito:

```
def low_pass_mask(img):
    # filtros - marcara circular concentrica, apenas os pontos localizados no
centro são 1
    rows, cols = img.shape
    crow, ccol = int(rows / 2), int(cols / 2)

    mask = np.zeros((rows, cols, 2), np.uint8)
    r = 70
    center = [crow, ccol]
    x, y = np.ogrid[:rows, :cols]
    mask_area = (x - center[0]) ** 2 + (y - center[1]) ** 2 <= r * r
    mask[mask_area] = 1

    return mask


def show_spectrum_maks(img, magnitude_spectrum, fshift_mask_mag, img_back):
    fig = plt.figure(figsize=(12, 12))
    ax1 = fig.add_subplot(2, 2, 1)
    ax1.imshow(img, cmap='gray')
    ax1.title.set_text('Input Image')
    ax2 = fig.add_subplot(2, 2, 2)
    ax2.imshow(magnitude_spectrum, cmap='gray')
    ax2.title.set_text('FFT of image')
    ax3 = fig.add_subplot(2, 2, 3)
    ax3.imshow(fshift_mask_mag, cmap='gray')
    ax3.title.set_text('FFT + Mask')
    ax4 = fig.add_subplot(2, 2, 4)
    ax4.imshow(img_back, cmap='gray')
    ax4.title.set_text('After inverse FFT')
    plt.show()


def low_pass_filter(image_file):
    img = cv.imread(image_file, 0)

    dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)

    dft_shift = np.fft.fftshift(dft)

    magnitude_spectrum = 20*np.log(cv2.magnitude(dft_shift[:,:,0],
dft_shift[:,:,1]) + 1)

    #fshift = dft_shift * high_pass_mask(img)
    fshift = dft_shift * low_pass_mask(img)

    fshift_mask_mag = 2000*np.log(cv2.magnitude(fshift[:, :, 0], fshift[:, :, 1]) +
1)
    # '+ 1' para evitar casos de log(0)

    f_ishift = np.fft.ifftshift(fshift) #inverse shift, retornar os cantos do
centro de volta para os cantos
```

```
    img_back = cv2.idft(f_ishift) #trasnformada de furrier inversa
    img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])

    show_spectrum_maks(img, magnitude_spectrum, fshift_mask_mag, img_back)
```

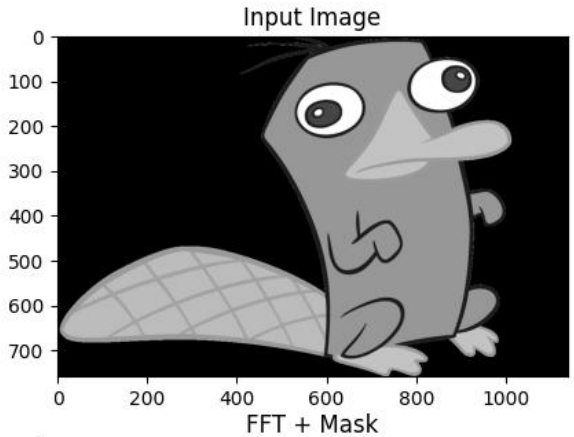


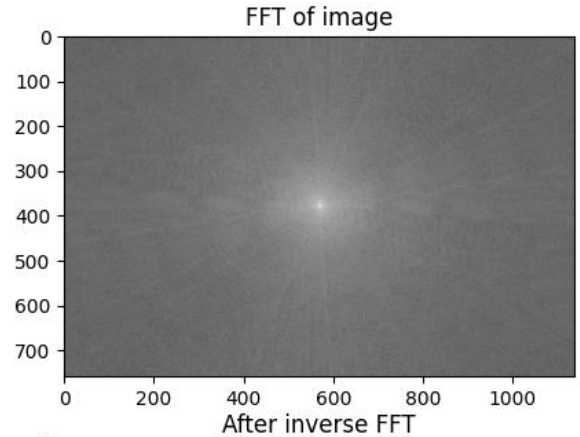


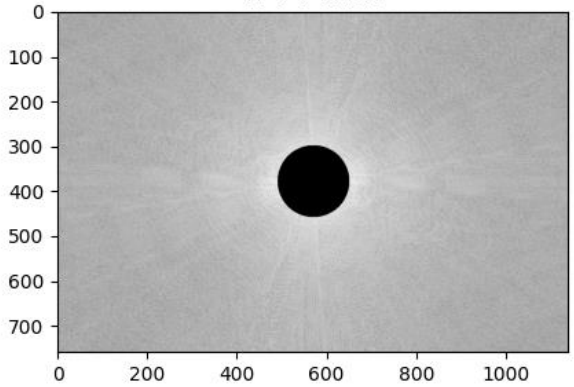
6. Aplicação de filtro passa-alta (sobel, prewitt, fourier ou canny)

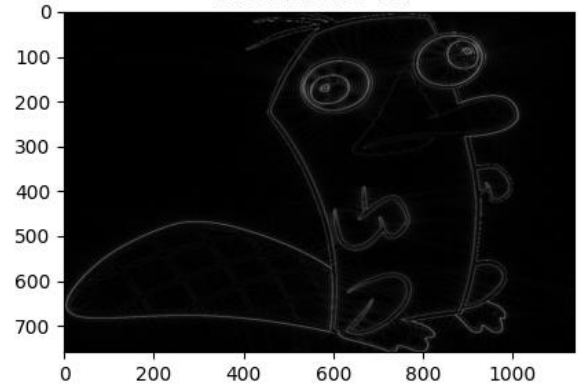
O filtro de passa alta tem como objetivo filtrar as regiões de baixa frequência da imagem (regiões com pouca informação), em termos práticos, ele irá apagar as regiões sem riqueza de detalhes e enfatizar principalmente os contornos e bordas. Segue abaixo o código utilizado:

```
def high_pass_mask(img):
    rows, cols = img.shape
    crow, ccol = int(rows / 2), int(cols / 2)

    mask = np.ones((rows, cols, 2), np.uint8)
    r = 80
    center = [crow, ccol]
    x, y = np.ogrid[:rows, :cols]
    mask_area = (x - center[0]) ** 2 + (y - center[1]) ** 2 <= r * r
    mask[mask_area] = 0

    return mask

def high_pass_filter(image_file):
    img = cv.imread(image_file, 0)

    dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)

    dft_shift = np.fft.fftshift(dft)

    magnitude_spectrum = 20*np.log(cv2.magnitude(dft_shift[:,:,0],dft_shift[:,:,1])
+ 1)

    #fshift = dft_shift * high_pass_mask(img)
    fshift = dft_shift * high_pass_mask(img)

    fshift_mask_mag = 2000*np.log(cv2.magnitude(fshift[:, :, 0], fshift[:, :, 1]) +
1)
    # '+ 1' para evitar casos de log(0)

    f_ishift = np.fft.ifftshift(fshift)
    img_back = cv2.idft(f_ishift) #trasnformada de furrier inversa
    img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])

    show_spectrum_maks(img, magnitude_spectrum, fshift_mask_mag, img_back)
```


Input Image
0
100
200
300
400
500
600
700
0
200
400
600
800
1000


FFT of image
0
100
200
300
400
500
600
700
0
200
400
600
800
1000


FFT + Mask
0
100
200
300
400
500
600
700
0
200
400
600
800
1000


After inverse FFT
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

7. Inverter a imagem (horizontal e vertical)

Inverter uma imagem é um processo simples, considerando que a mesmo é armazenada em memória como uma matriz de pixels, basta inverter a ordem dessa matriz (vertical ou horizontalmente) e a imagem será invertida. Segue abaixo o código utilizado:

```
def invert_image(image_file, orientation):
    img = cv.imread(image_file, 1)
    img = ResizeWithAspectRatio(img, 480)
    #show_image(img, "original")

    img2 = copy.deepcopy(img)
    rows, cols = img.shape[:2]

    if orientation == 'vertical':
        for row in range(rows):
            for col in range(cols):
                img2[(rows-1)-row][col] = img[row][col]

    elif orientation == 'horizontal':
        for row in range(rows):
            for col in range(cols):
                img2[row][(cols-1) - col] = img[row][col]

    elif orientation == 'vertical-horizontal':
        img3 = copy.deepcopy(img)
        for row in range(rows):
            for col in range(cols):
                img2[(rows-1)-row][col] = img[row][col]

        for row in range(rows):
            for col in range(cols):
                img3[row][(cols - 1) - col] = img2[row][col]

    else:
        print("-<! Invalid Input, enter 'vertical' or 'horizontal' in seconde
argument !>-")
        exit(0)

    show_image(img3, "inverted")
```

Imagem original	Imagem invertida horizontalmente	Imagem invertida verticalmente	Imagem invertida horizontalmente e verticalmente
			

8. Aumento de brilho e contraste

O aumento de Brilho e contraste é alcançado aumentando o valor dos pixels, somando uma constante aos mesmos se aumenta o brilho, e multiplicando os mesmo por um valor se melhora o contraste, pois ao se multiplicar por uma constante. Segue abaixo o código utilizado:

```
def adjust_brigh_contrast_2(image_file, contrast, bright):
    #def map_value(var, var_min, var_max, ret_min, ret_max):
    taxa_contrast = map_value(contrast,0,100,0,1.5)
    taxa_bright = int(map_value(bright,0,100,0,150))
    img = cv.imread(image_file, 1)
    img = ResizeWithAspectRatio(img, 480)
    show_image(img, "original")

    rows, cols, _ = img.shape

    mat_sum = np.ones(img.shape,dtype="uint8")*taxa_bright
    mat_multiple = np.ones(img.shape)*taxa_contrast

    img = np.clip(cv2.add(img,mat_sum), 0, 255)
    img = np.uint8(np.clip(cv2.multiply(np.float64(img),mat_multiple), 0, 255))

    show_image(img)
```

Imagem original

Imagem processada