INF101 – Introdução à Programação II
Roteiro de Prática: 17 de agosto de 2023

Introdução

O objetivo principal desta prática é exercitar o uso de funções para estruturar a implementação de um programa em Python, também a utilização da estrutura de dados *arranjo bidimensional*. Para tanto, vamos repetir o assunto sobre processamento de imagem que foi usado em INF100, no período passado. Porém vamo-nos ater apenas à solução da operação de *detecção de bordas*. Esta (em inglês, *edge detection*) é uma técnica de processamento de imagem usada para identificar pontos em uma imagem digital com descontinuidades ou, simplesmente, com mudanças bruscas no brilho da imagem. A detecção de bordas é fundamental para resolver problemas mais substanciais na área de *visão computacional* (ciência e tecnologia das máquinas que "enxergam").

Antes de obter uma solução para o problema proposto, vamos primeiro transformar a imagem, se colorida, em tons de cinza. Para que o código do programa fique bem estruturado, vamos criar e usar funções apropriadas tanto para o efeito de tons de cinza quanto para a detecção de bordas.

Para quem ainda tem dificuldades de como representar imagens por meio de matrizes, assista à aula gravada no site: https://youtu.be/QF-qjVRf3i8os.

Algoritmo para efeito de tons de cinza

Reponha cada componente `(r, g, b)` da imagem pela luminância (brilho) do respectivo pixel:

```
luminancia = int(0.299 * r + 0.587 * g + 0.114 * b)
```

Lembre-se de que cada pixel `(r, g, b)` é obtido do elemento `im[y][x]` da matriz `im` contendo a imagem colorida original. Use os atributos `im.altura` e `im.largura` para obter as dimensões da imagem. Esses atributos são disponibilizados pelo arquivo fonte `imagens.py` que você deverá obter do *site* de entrega de trabalhos.

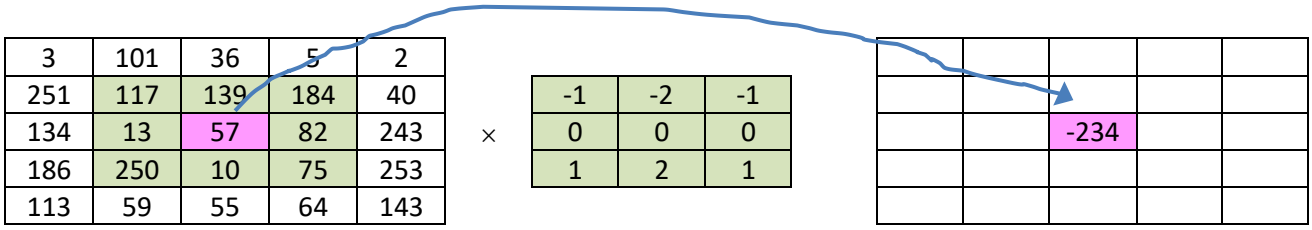
Algoritmo para a detecção de bordas (Filtro de Sobel)

Existem vários métodos para obter as bordas de uma imagem. Teoricamente, são operações matemáticas de convolução usando matrizes como elementos discretos. Para simplificar, nesta aula, vamo-nos ater ao método de detecção elaborado por Irwin Sobel & Gary Feldman no Laboratório de Inteligência Artificial de Stanford (SAIL), em 1968.

Agora, em vez de alterar cada pixel de forma independente, usaremos uma pequena matriz, chamada de *matriz* ou *filtro de convolução*, que será "passada" em toda a imagem, transformando cada pixel de acordo não só com o valor dele, mas também com o de seus vizinhos. O Filtro de Sobel usa duas matrizes de convolução: uma na direção vertical, a outra na direção horizontal. As matrizes que serão usadas aqui são as seguintes (Obs.: As matrizes já estão preparadas para trabalhar no sistema de coordenadas y-x, conforme usado em INF100):

```
filtro_vertical =   [[-1, -2, -1],
                     [ 0,  0,  0],
                     [ 1,  2,  1]]

filtro_horizontal = [[-1,  0,  1],
                     [-2,  0,  2],
                     [-1,  0,  1]]
```

Cada bloco de 3 x 3 pixels da imagem será multiplicado por um desses filtros para alterar apenas o pixel “central” da matriz de cada vez, como ilustrado abaixo com o filtro vertical:



3	101	36	5	2
251	117	139	184	40
134	13	57	82	243
186	250	10	75	253
113	59	55	64	143

×

-1	-2	-1
0	0	0
1	2	1

		-234		

Antes de começar o processamento dos pixels, precisamos fazer uma cópia da imagem para armazenar os pixels transformados. Isso pode ser feito com o comando:

```
im2 = im.copiar()
```

que cria uma cópia da imagem armazenada em **`im`** e atribui à variável **`im2`**. Faremos isso só para ter outra imagem com as mesmas dimensões e características da original. Iremos então pegar cada pixel de **`im`**, processá-lo em relação a seus vizinhos usando os filtros acima, e colocar o resultado em **`im2`**. Isso pode ser feito da seguinte forma:

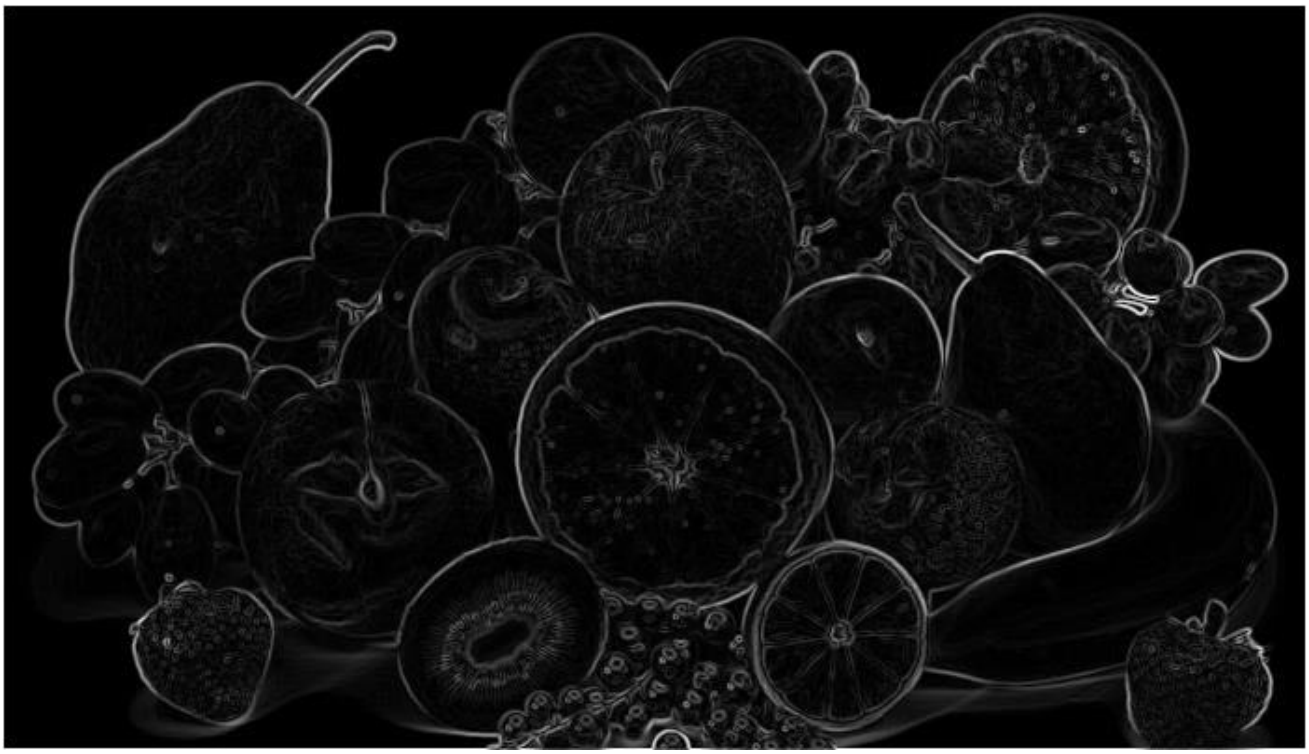
```
# Inicializa a matriz para conter as pontuações de borda.
bordas_im = np.zeros((im.altura, im.largura))

# Percorre todas as coordenadas da matriz original, deixando de fora as
# extremidades da imagem.
for y in range(1, im.altura-1):
    for x in range(1, im.largura-1):
        # Aplica o filtro vertical.
        pix = 0  # inicia somatório
        for i in range(0, 3):
            for j in range(0, 3):
                # Pega pixel “embaixo” da coordenada do filtro vertical.
                r, g, b = im[y-(1-i)][x-(1-j)]
                # Multiplica pelo filtro e soma. Para isso podemos usar apenas
                # um dos componentes (r, g ou b), já que todos são iguais.
                pix = pix + r * filtro_vertical[i][j]
        pontuacao_vert = pix/4
        # Aplica o filtro horizontal.
        ... (Complete o código)

        # Congrega as duas pontuações e armazena-as na matriz de pontuações
        # de borda.
        pontuacao_borda = sqrt(pontuacao_vert**2 + pontuacao_horiz**2)
        bordas_im[y][x] = pontuacao_borda

# Normaliza as pontuações de borda para o intervalo [0, 1].
bordas_im = bordas_im/bordas_im.max()

# Pós-processamento: produz a imagem com as bordas detectadas.
for y in range(1, im.altura-1):
    for x in range(1, im.largura-1):
        tom_cinza = int(bordas_im[y][x] * 255)
        im2[y][x] = (tom_cinza, tom_cinza, tom_cinza)
```

Depois de fazer todo o processo acima, retorne a imagem `im2`.

É necessário ter um pouco de paciência, pois essa operação demora um bom tempo, uma vez que, para cada pixel, precisamos fazer um somatório com todos os seus vizinhos. Apenas fique atento ao `Python Shell` para ver se aparece alguma mensagem de erro enquanto você aguarda o processamento.

Ao aplicar o Filtro de Sobel à imagem abaixo:

obteremos a respectiva imagem com as bordas detectadas a seguir:



Instruções

1. Faça o *download* dos arquivos fontes `imagens.py` e `p01.py` e do arquivo de imagem `fruits-700.jpg` para seu diretório de trabalho, certamente, o diretório `/home/alunos`, mas pode ser qualquer um dentro do seu diretório *home*.
2. Não mexa no arquivo `imagens.py`. Deixe-o intacto no mesmo diretório de trabalho.
3. Abra a IDE IDLE, clicando duas vezes no respectivo ícone na área de trabalho.
4. Abra o arquivo fonte `p01.py`. Este contém ainda só o esqueleto do trabalho a ser entregue. As funções `tonal()` e `sobel()` precisam ser implementadas por você. Para tanto, use os respectivos algoritmos que foram apresentados no início deste roteiro.
5. Complete os códigos das funções pedidas nos locais apropriados onde há reticências (...). Remova as reticências antes de tentar executar sua implementação. Não modifique a função `main()` nem a chamada dela no final do código fonte.
6. Retire todos os erros de sua implementação e teste-a. Se seu programa estiver funcionando bem, vão aparecer a imagem original `fruits-700.jpg` e, depois de muitos segundos, a imagem com somente as bordas detectadas.
7. Para melhorar a confiabilidade da correção de sua implementação, teste seu programa com mais uma imagem. Use o arquivo `pinwheel.jpg` disponível no sítio de entrega de trabalhos do LBI. A imagem neste arquivo tem mais resolução, portanto o tempo de processamento será maior ainda. Tenha paciência!
8. Idem ao item anterior, usando o arquivo `baboon_dip.png`.

☞ Não se esqueça de preencher o cabeçalho do código fonte com seus dados e uma breve descrição do programa.

Após certificar-se de que seu programa esteja correto, envie o arquivo do programa fonte (`p01.py`) através do sistema de entrega do LBI.