INF01046 – Fundamentos de Processamento de Imagens
Aluno: Lucca Franke Kroeff
Professor: Manuel M. Oliveira

Trabalho de Implementação Etapa 1 - Operações Básicas sobre Imagens Digitais

Com a finalidade de realizar o trabalho proposto é preciso dizer que a aplicação foi feita na linguagem de programação Python e foram utilizadas as seguintes bibliotecas:

- **PIL** (biblioteca que possibilitou trabalhar com as imagens)
- **Tkinter** (biblioteca utilizada na construção da interface do programa)
- **OS** (biblioteca usada na abertura, renomeação e verificação se o arquivo digitado pelo usuário existia ou não)

Parte I – Leitura e Gravação de Arquivos de Imagens:

Para cumprir com a primeira parte do trabalho foram utilizados os trechos de código apresentados abaixo:

```
if(os.path.isfile(NomeArquivo)):     # Arquivo existe
    im = Image.open(NomeArquivo)
    width, height = im.size      # Verificando tamanho da imagem, altura
e largura
    Operacao = -1    # Operação inicia em -1 para não ser equivalente a
0 e fazer o laço while
else:
    Operacao = '0'  # Arquivo não existe então não entra no laço
```

- Aqui, primeiramente verificamos se o arquivo existe no diretório do usuário e, se sim, abrimos o arquivo de formato *jpg* com uma função da biblioteca PIL.

```
def AlteraNomeArquivo(NomeNovo):
    os.rename(NomeArquivo, NomeNovo)
```

- Parte do programa que recebe um novo nome de arquivo e altera o nome do arquivo existente.

Pergunta: "Você percebe alguma diferença visual entre elas? Alguma diferença nos tamanhos dos arquivos?"

Resposta: Há uma diferença perceptível no visual das imagens dadas, havendo imagens com mais pixels e outras com menos. Exatamente por tal fato, existe também uma diferença nos tamanhos dos arquivos.

Conclusão / Dificuldades:

Essa etapa foi concluída tranquilamente, a grande dificuldade encontrada foi escolher uma biblioteca, dentre várias disponíveis, para realizar a abertura dos arquivos *jpg.*

Parte II – Leitura, Exibição e Operações sobre Imagens (80 pontos):

a) Espelhamento horizontal e vertical da imagem original.

Para cumprir essa parte do trabalho foi utilizado o trecho de código apresentado abaixo:

```
    def Espelhamento(Opcao):
        im_nova = Image.new('RGB', (width, height), color = 'black')

        if(Opcao == '1' or Opcao =='2'):
            for x in range(width):
                for y in range(height):
                    ValorPixel = im.getpixel((x, y))
                        if(Opcao == '1'):    # Realizando espelhamento
horizontal, altera x
                        novo_x = width - x - 1
                        im_nova.putpixel((novo_x, y), ValorPixel)
                        elif(Opcao == '2'): # Realizando espelhamento
vertical, altera y
                        novo_y = height - y - 1
                        im_nova.putpixel((x, novo_y), ValorPixel)

        else:
            print("Opção inválida")

        MostrarImagens(im, im_nova)
        return im_nova
```

Sendo assim, caso o usuário escolhesse a opção 1, os pixels da largura da imagem seriam alterados, e caso escolhesse a opção 2, mudaríamos os pixels da altura, sendo que todos os pixels mapeados vão para o lado oposto do que estão, fazemos isso realizando o cálculo:

largura - posição_do_pixel - 1 ***(no caso do espelhamento horizontal) =*** *pos_do_pixel*
altura - *posição_do_pixel - 1* ***(no caso do espelhamento vertical) =*** *pos_do_pixel*

Conclusão / Dificuldades:

Essa etapa foi completada com sucesso, o verdadeiro desafio foi pensar que fórmula matemática que deveria ser utilizada para mapear os pixels.

Ademais, inicialmente encontrei um resultado errado, já que estava vasculhando os pixels da imagem do vetor e operando sobre ela mesma, então me deparei com uma imagem simétrica, não espelhada.

A fim de corrigir o erro, cheguei na solução de criar uma nova imagem e operar sobre ela enquanto olho pixel por pixel a outra.

Resultado (Opção 2)

Para prosseguir basta fechar a janela!

Imagem Anterior:

Imagem Posterior:

b) Conversão de imagem colorida para tons de cinza (luminância).

Para cumprir essa parte do trabalho foi utilizado o trecho de código apresentado abaixo:

```
    def ConversaoTonsCinza():
        im_nova = Image.new('RGB', (width, height), color = 'black')

        for x in range(width):
            for y in range(height):
                r, g, b = im.getpixel((x, y))
                r = r * 0.2989  # Alterando os canais rgb de valor
                b = b * 0.5870
                g = g * 0.1140
                L = r + g + b
                ValorPixel = int(L), int(L), int(L)
                  im_nova.putpixel((x, y), ValorPixel)     # Coloca na
nova imagem o valor de pixel cinza

        return im_nova
```

Como uma imagem colorida pode ser transformada em uma imagem em tons de cinza aplicando-se a seguinte fórmula para cada um dos pixels:

*L = 0.299*R + 0.587*G + 0.114*B*
É exatamente isso que é realizado no código acima.

Pergunta: *'O que acontecerá com uma imagem em tons de cinza (Ri = Gi = Bi = Li) caso o cálculo de luminância seja aplicado a ela?'*

Resposta:
Para responder essa pergunta, chamei a função ConversaoTonsCinza() para uma imagem já em tons de cinza, e nessa experiência foi possível observar mudanças sutis, pixels mais claros ficaram mais claros e os escuros ficaram mais escuros, aumentando o contraste. Além disso, é possível apontar que a imagem se parece mais 'fria' que a anterior. No geral não houve grandes mudanças.

Conclusão / Dificuldades:
A etapa foi concluída com sucesso. Como essa foi a primeira função que tentei implementar, a dificuldade encontrada primeiramente foi estudar como passaria pixel por pixel uma imagem e além disso, como operaria sobre outra imagem.

Resultado

Para prosseguir basta fechar a janela!

Imagem Anterior:

Imagem Posterior:

c) Quantização (de tons) sobre as imagens em tons de cinza.

Para cumprir essa parte do trabalho foi utilizado o trecho de código apresentado abaixo:

```
def Quantizacao(NumeroTons):
    ConversaoTonsCinza()
    im_nova = Image.new('RGB', (width, height), color = 'black')
    MaiorPixel, MenorPixel = 0, 0

    for x in range(width):
         for y in range(height):     # Vasculhando a imagem para saber o
maior valor de pixel e o menor valor de pixel
            L = im.getpixel((x, y))
            if(L[1] > MaiorPixel):
                MaiorPixel = L[1]
            if(L[1] < MenorPixel):
                MenorPixel = L[1]

    TamanhoIntervalo = MaiorPixel - MenorPixel + 1
    NumeroTons = int(NumeroTons) - 1
    TamanhoBin = TamanhoIntervalo / NumeroTons

    if(int(NumeroTons) >= TamanhoIntervalo):
        return im

    else:
        for x in range(width):
            for y in range(height):
                r, g, b = im.getpixel((x, y))
                Divisao = round(r / TamanhoBin)
                r = Divisao * TamanhoBin
                g = r
                b = r
                ValorPixel = int(r), int(g), int(b)
                im_nova.putpixel((x, y), ValorPixel)

        MostrarImagens(im, im_nova)
        return im_nova
```

A função se inicia transformando a imagem em tons de cinza a partir da operação definida anteriormente. Após isso, encontramos o pixel com maior e menor valor de luminância e definimos o tamanho do intervalo como MaiorPixel - MenorPixel + 1. A partir dessa operação definimos o TamanhoBin. Com essas informações, conseguimos calcular o valor que cada pixel deve receber.

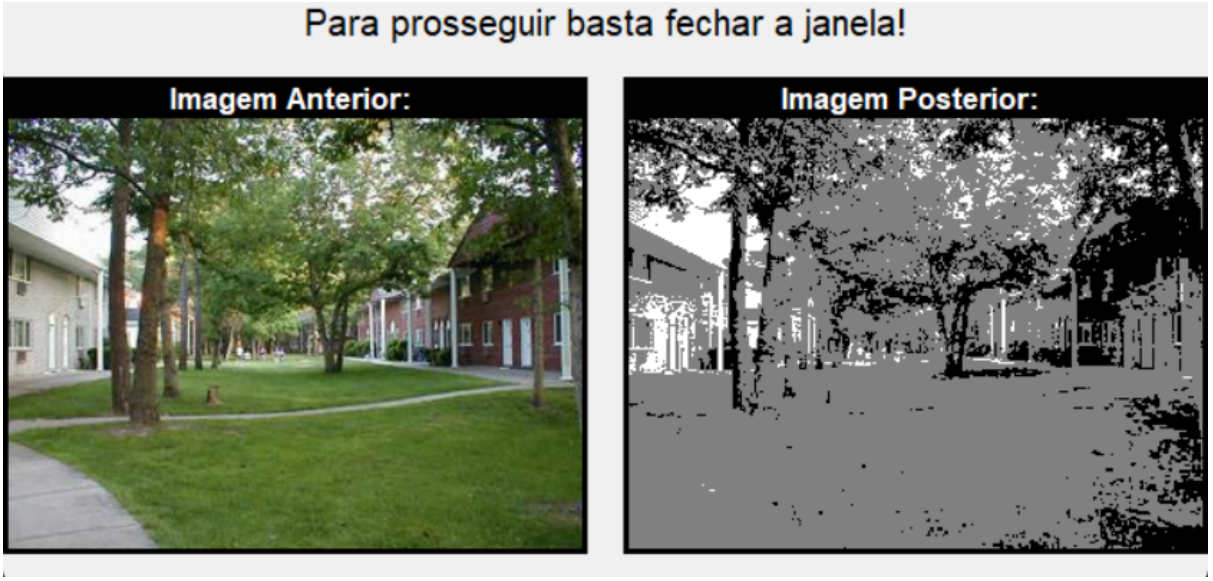
Além disso, caso o número de tons seja maior ou igual ao tamanho do intervalo, não há necessidade de realizarmos a quantização, afinal, já há tons o suficiente na imagem para representarmos todos os pixels da imagem original.

Conclusão / Dificuldades:

A etapa foi concluída corretamente. Acredito que essa tenha sido uma das etapas mais complicadas da atividade, uma vez que envolvia cálculos e fórmulas mais difíceis de serem deduzidas. Eu particularmente demorei para entender como poderia fazer isso.

Algo que faria desde o início para resolver esse problema mais facilmente seria anotar em um papel minhas ideias, a partir do momento que fiz isso, o problema foi resolvido muito mais facilmente.

Resultado



d) Salvamento da imagem resultante das operações realizadas em um arquivo JPEG

Para cumprir essa parte do trabalho foi utilizado o trecho de código apresentado abaixo:

```
    def SalvaArquivo(NomeArquivo):
        im_nova.save(NomeArquivo)
```

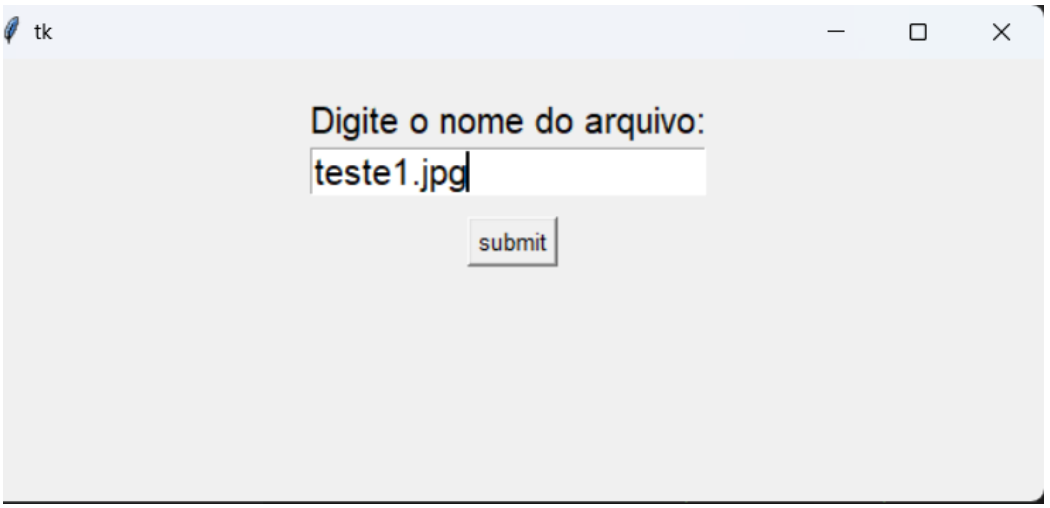
A função save permite salvar uma imagem como um arquivo *jpg* o qual o nome é dado pelo usuário.

Após realizar essa operação, um novo arquivo *jpg* é criado no diretório do usuário com o nome que ele escolher.

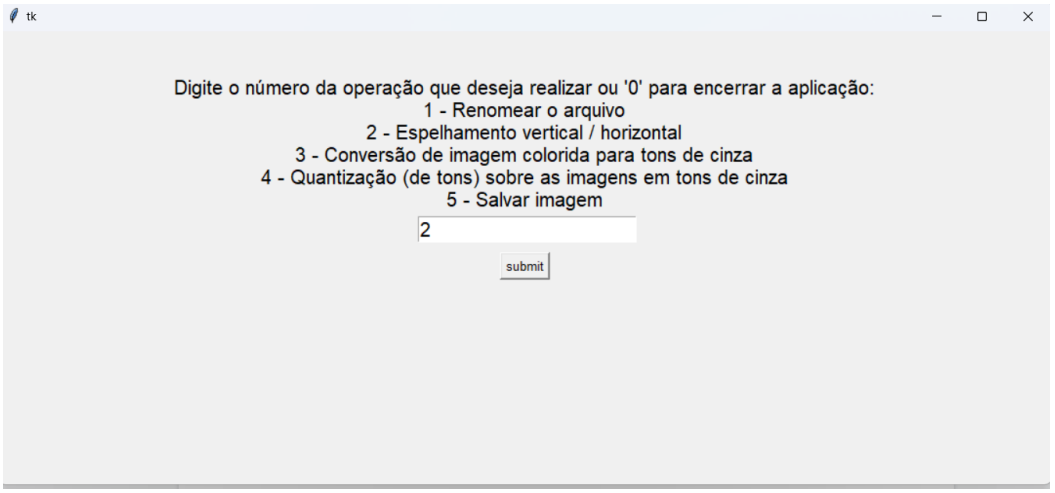
INTERFACE:

A fim de implementar um programa mais completo, foi utilizada a biblioteca Tkinter para implementar uma interface.

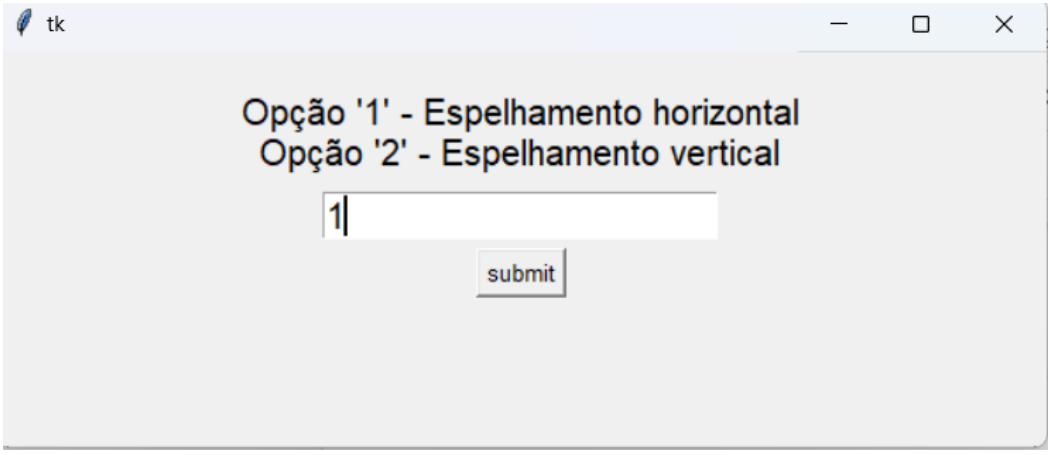
Inicialização:



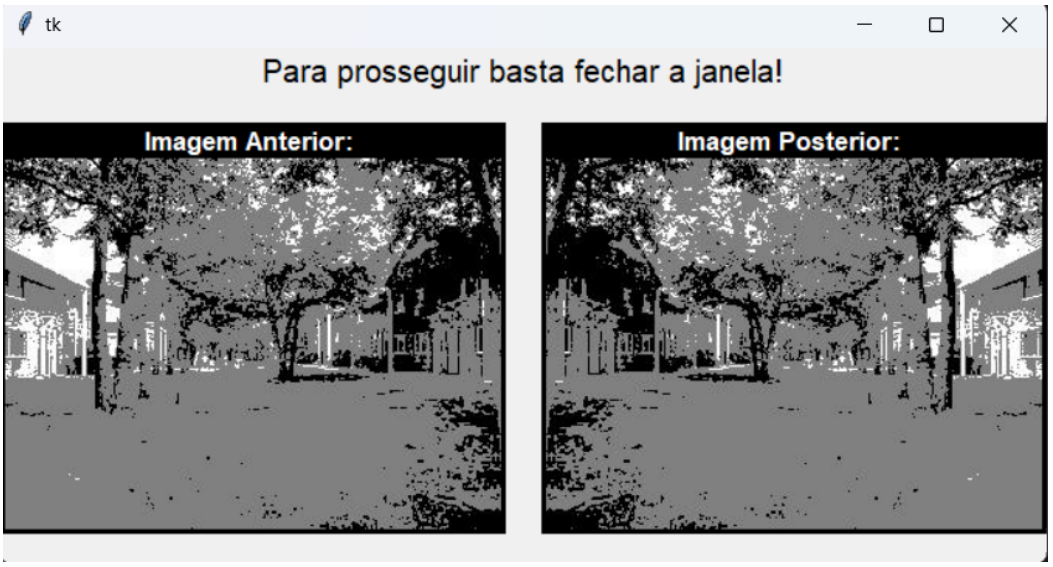
Menu Geral:



Ex de operação de espelhamento vertical / horizontal:



Fim da operação:



Conclusão / Dificuldades:

Essa parte do trabalho foi muito complicada, no entanto trouxe muito aprendizado, já que eu nunca havia programado interfaces antes, até por isso acredito que elas tenham ficado simplificadas até demais, algo que espero evoluir com o passar do semestre. Foi uma experiência muito desafiadora e excelente para minha evolução como programador.