

Bulk Image Editor

Project Implementation Report

Índice

Introdução	2
Visão Inicial	3
Bibliotecas Utilizadas	4
1. <i>Open CV</i>	4
2. <i>Pillow</i>	5
3. <i>Tkinter</i>	6
4. <i>Pathlib</i>	7
Explicação Detalhada do Código	8
<i>def (carregar_imagem)</i>	8
<i>def (ajustar_imagem)</i>	8
<i>def (guardar_def)</i>	8
<i>def (aplicar_pasta)</i>	9
<i>def (tkinter)</i>	9
Desafios Encontrados	10
Resultados Obtidos	11
<i>Edição em tempo real</i>	11
<i>Processamento de lote eficiente</i>	11
<i>Interface intuitiva</i>	11
<i>Conversão segura de formatos</i>	11

Introdução

Durante a análise das ferramentas de edição de imagem disponíveis na internet, constatei que existem inúmeras opções de elevada qualidade, mas praticamente todas se focam na edição manual e individual de imagens.

Não encontrei soluções que combinassem qualidade de edição com a capacidade de processar várias imagens em simultâneo.

Este projeto surge precisamente para colmatar essa lacuna, permitindo a um utilizador editar uma imagem de exemplo, guardar as definições aplicadas à mesma e, a partir delas, replicar os ajustes em dezenas ou centenas de fotografias de forma automática.

Visão Inicial

No início do projeto, a ideia era criar uma janela o mais simples possível, na qual fosse possível utilizar uma fotografia para aplicar edições e oferecer a funcionalidade de **recorte da imagem**. No entanto, após alguns dias de desenvolvimento, ficou evidente que a implementação do recorte seria mais complexa e pouco prática, pelo que o objetivo foi ajustado para se focar apenas na edição de imagens.

Desde o início que se sabia que seria necessário ter uma parte inicial que permitisse **carregar a fotografia-tipo**. Inicialmente, a ideia era programar a seleção através de um caminho de ficheiro, mas com o uso do **Tkinter** tornou-se possível inserir a imagem diretamente a partir da memória do computador, tornando o processo mais intuitivo.

Outro aspeto essencial era disponibilizar ao utilizador controlos deslizantes para ajustar o brilho e o contraste da fotografia. Para esse efeito, recorreremos às funções **alpha** e **beta** da biblioteca **OpenCV**.

Surgindo, de seguida, a necessidade de criar uma função que guardasse as definições utilizadas, permitindo reaplicar as mesmas edições posteriormente. Com base nisso, foi ainda implementada uma função capaz de abrir uma pasta, iterar sobre as imagens em loop, aplicar as definições guardadas e guardar os resultados. Inicialmente, pensou-se em criar uma pasta final separada, mas acabou por se optar por guardar os ficheiros na mesma pasta e alterar os nomes para evitar a sobrescrita.

Por fim, o **Tkinter** foi responsável pelo aspecto estético, criando uma janela prática e funcional que tornasse a utilização do programa simples para qualquer utilizador.

Bibliotecas Utilizadas

1. Open CV

Biblioteca avançada de visão computacional e processamento de imagem. Neste projeto, é utilizada para manipular **arrays de imagens** de forma eficiente.

Algumas das funções utilizadas foram:

`cv2.imread()` - Lê uma imagem e retorna um array **NumPy**.

`cv2.resize()` - Redimensiona a imagem para largura e altura específicas.
`INTER_AREA` é recomendado para redução de tamanho, preservando detalhes.

`cv2.convertScaleAbs()` - Faz ajuste linear de pixel. **Alpha** controla o contraste e o **Beta** controla a luminosidade.

`cv2.imwrite()` - Guarda o array de imagem no disco num formato suportado.

```
if caminho:
    imagem_original = cv2.imread(caminho)
    largura_nova = 400
    altura_nova = 580
    imagem_original = cv2.resize(imagem_original, (largura_nova, altura_nova), interpolation = cv2.INTER_AREA)
    ajustar_imagem()
```

Imagem 1 - Exemplos de uso da biblioteca **Open CV**

```
alpha = contraste.get()
beta = luminosidade.get()

imagem_editada = cv2.convertScaleAbs(imagem_original, alpha = alpha, beta = beta)
```

Imagem 2 - Exemplos de uso da biblioteca **Open CV**

2. Pillow

Pillow foi usado como ponte entre o **Open CV** e o **Tkinter**.

Uma biblioteca para manuseamento de imagens.

Algumas das funções utilizadas foram:

`Image.fromarray()` - Converte um array **NumPy** num objeto **Pillow Image**.

`cv2.cvtColor(cv2.COLOR_BGR2RGB)` - Converte uma imagem de **BGR (Open CV)** para **RGB (Pillow)**.

`ImageTk.PhotoImage()` - Converte **Pillow Image** numa imagem compatível com **Tkinter**.

```
imagem_PILLOW = Image.fromarray(cv2.cvtColor(imagem_editada, cv2.COLOR_BGR2RGB))
```

Imagem 3 - Exemplos de uso da biblioteca **Pillow**

```
tk_imagem = ImageTk.PhotoImage(imagem_PILLOW)
```

Imagem 4 - Exemplos de uso da biblioteca **Pillow**

3. Tkinter

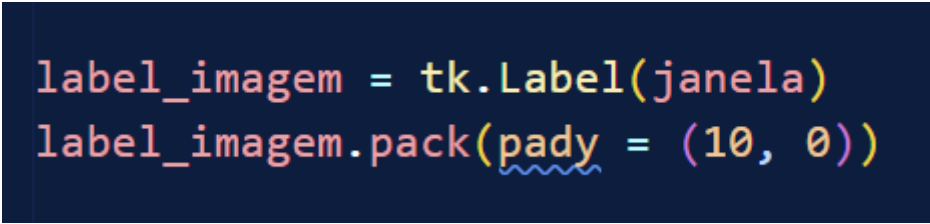
Biblioteca padrão do Python para criar interfaces gráficas (GUI).

Algumas das funções utilizadas foram:

`filedialog.askopenfilename()` - Abre o explorador de arquivos para escolher uma imagem individual. Pode-se usar um `filetypes` para limitar os tipos de arquivos selecionáveis.

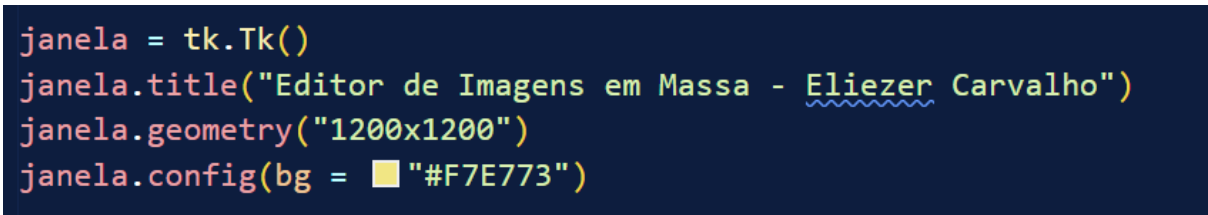
`label_imagem = tk.Label(janela)` - Widget usado para exibir a imagem convertida via `Pillow ImageTk`.

`janela.mainloop()` - Mantém a janela aberta e escuta os eventos.

A imagem mostra um fundo escuro com código Python colorido. O código consiste em duas linhas: a primeira cria um widget Label a partir de uma variável chamada 'janela', e a segunda chama o método 'pack' no mesmo widget com o parâmetro 'pady' igual a '(10, 0)'.

```
label_imagem = tk.Label(janela)
label_imagem.pack(pady = (10, 0))
```

Imagem 5 - Exemplos de uso da biblioteca *Tkinter*

A imagem mostra um fundo escuro com código Python colorido. O código cria uma janela Tk, define seu título como 'Editor de Imagens em Massa - Eliezer Carvalho', define sua geometria como '1200x1200' e configura o fundo com a cor hexadecimal '#F7E773'.

```
janela = tk.Tk()
janela.title("Editor de Imagens em Massa - Eliezer Carvalho")
janela.geometry("1200x1200")
janela.config(bg = "#F7E773")
```

Imagem 6 - Exemplos de uso da biblioteca *Tkinter*

4. Pathlib

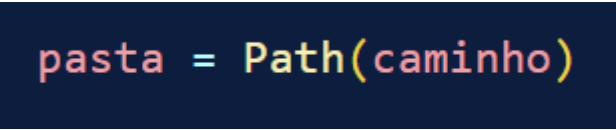
Pathlib facilita a **manipulação** de caminhos e arquivos.

Algumas das funções utilizadas foram:

`Path()` - Cria um objeto representando diretórios ou arquivos.

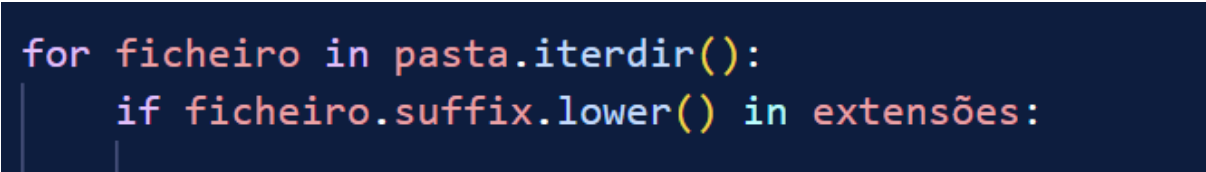
`pasta.iterdir()` - Itera sobre todos os arquivos dentro do diretório permitindo processar cada imagem.

`saida.mkdir(exist_ok = True)` - Cria uma pasta válida. `exist_ok = True` evita erros caso uma pasta com o mesmo nome já exista.



```
pasta = Path(caminho)
```

Imagem 7 - Exemplos de uso da biblioteca *Pathlib*



```
for ficheiro in pasta.iterdir():  
    if ficheiro.suffix.lower() in extensões:
```

Imagem 8 - Exemplos de uso da biblioteca *Pathlib*

Explicação Detalhada do Código

def (carregar_imagem)

O utilizador selecciona uma imagem, o OpenCV lê a imagem como um array e redimensiona para exibição na interface.

No fim do bloco de código chama `ajustar_imagem()` para aplicar brilho/contraste iniciais.

def (ajustar_imagem)

Lê valores de `alpha` (contraste) e `beta` (brilho) dos sliders do **Tkinter** e aplica ajustes com `cv2.convertScaleAbs`.

Converte para **Pillow Image** e depois para `ImageTk.PhotoImage` e atualiza o `Label` da interface para mostrar a imagem editada.

def (guardar_def)

Armazena `alpha` e `beta` num dicionário global `definições` para aplicar os mesmos ajustes a outras imagens.

def (aplicar_pasta)

Permite ao utilizador seleccionar uma pasta, criar uma pasta de saída e iterar sobre todos os ficheiros da pasta com extensões válidas.

Lê cada imagem e aplica os ajustes guardados (**definições**) com **OpenCV**.

Por fim, a imagem é guardada na pasta de saída, preservando os ficheiros originais.

def (tkinter)

Os sliders estão ligados diretamente à função **ajustar_imagem()**, permitindo **visualização em tempo real** das alterações.

Os botões acionam funções que manipulam imagens (**carregar_imagem**, **guardar_def**, **aplicar_pasta**).

Widgets são dispostos verticalmente na ordem de adição.

Pode ser expandido futuramente usando **grid() ou **place()** para layouts mais complexos.**

Uso consistente de cores (**bg**, **fg**, **troughcolor**) e fontes (**font=("Arial", 9, "bold")**) garante um tema **unificado**.

Desafios Encontrados

Inicialmente, tive algumas dificuldades em compreender a biblioteca **Tkinter**, pois parecia-me um pouco antiquada em termos de estrutura e fluxo. Implementar a exposição da imagem e a sua atualização em tempo real, à medida que o utilizador ajustava o brilho e o contraste, tornou-se um verdadeiro desafio.

Outro obstáculo importante surgiu do facto de, após a edição, a imagem ser transformada num **array NumPy** pelo **OpenCV**, o que tornava impossível exibi-la diretamente no **Tkinter** sem conversão. Encontrar a forma correta de realizar essa conversão com o **Pillow + ImageTk** exigiu algum tempo e experimentação.

Além disso, a ideia inicial de implementar o recorte automático em massa mostrou ser bastante complexa. Muito tempo foi investido nessa funcionalidade, mas, devido à dificuldade de implementação e à necessidade de concluir a edição em massa de forma funcional, acabou por ser deixada de lado.

Resultados Obtidos

O projeto alcançou os objetivos propostos, proporcionando uma solução **prática** e **funcional** para a edição em massa de imagens. Entre os principais resultados obtidos, destacam-se os seguintes:

Edição em tempo real

Os ajustes de brilho e contraste efetuados através dos controlos deslizantes no **Tkinter** são aplicados instantaneamente à imagem de exemplo, permitindo ao utilizador visualizar as alterações antes de as aplicar a várias fotografias.

Processamento de lote eficiente

O programa consegue iterar sobre **dezenas** ou **centenas** de imagens dentro de uma pasta, aplicando as mesmas definições de edição de forma consistente e preservando os ficheiros originais.

Interface intuitiva

A utilização do **Tkinter** permitiu criar uma janela simples e funcional, onde todas as ações — carregar imagem, ajustar parâmetros, guardar definições e aplicar a uma pasta inteira — são facilmente acessíveis.

Conversão segura de formatos

A utilização de **OpenCV** e **Pillow** garantiu que as imagens fossem processadas e exibidas corretamente, superando o desafio de compatibilidade entre **arrays NumPy** e **Tkinter**.

Em suma, este projeto mostra que é possível combinar simplicidade de utilização, processamento eficiente e resultados visuais consistentes, mesmo com um grande volume de imagens.

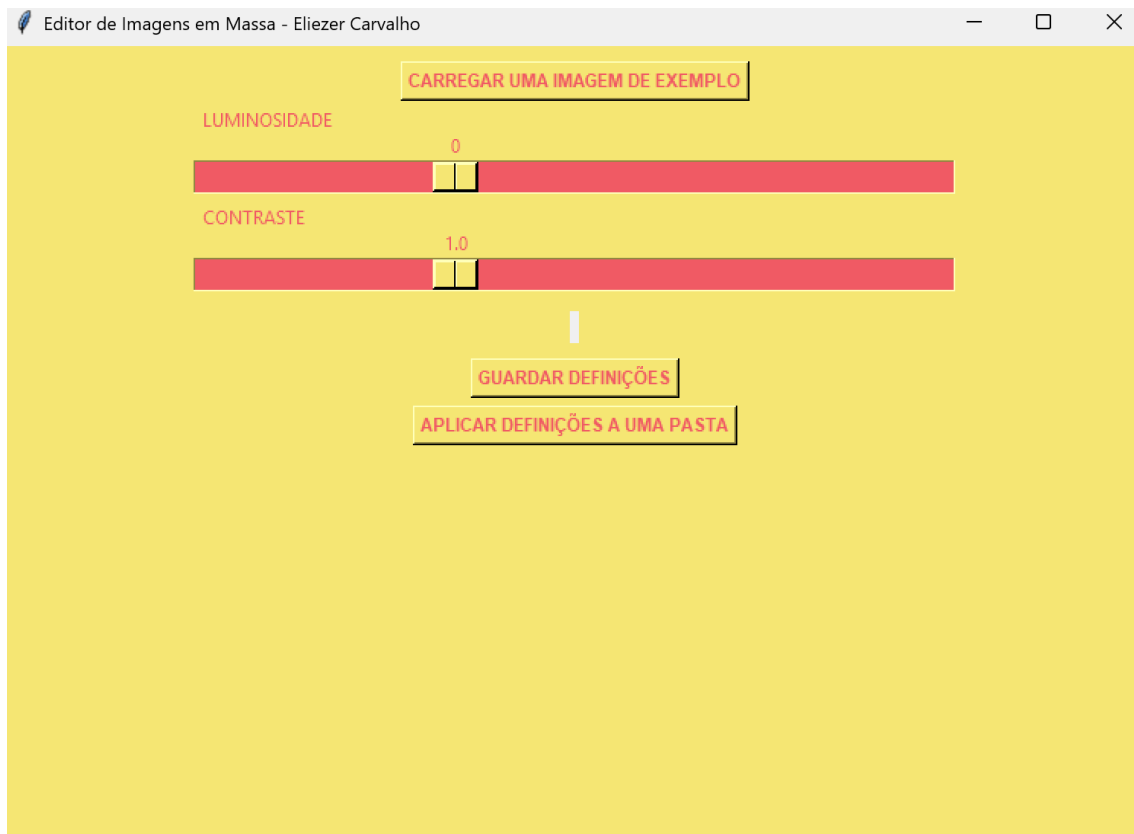


Imagem 9 - Resultado Final