

# Bulk Image Editor

---

## Project Implementation Report

# Table of Contents

---

Introduction	2
Initial Overview	3
Libraries Used	4
1. Open CV	4
2. Pillow	5
3. Tkinter	6
4. Pathlib	7
Detailed Explanation of the Code	8
<i>def (carregar_imagem)</i>	8
<i>def (ajustar_imagem)</i>	9
<i>def (guardar_def)</i>	9
<i>def (aplicar_pasta)</i>	9
<i>def (tkinter)</i>	10
Challenges Encountered	10
Results Achieved	11
<i>Real-time editing</i>	12
<i>Efficient batch processing</i>	12
<i>Intuitive interface</i>	12
<i>Safe format conversion</i>	12

# Introduction

---

While analyzing the image editing tools available on the internet, I found that there are numerous high-quality options, but virtually all of them focus on manual and individual image editing.

**I couldn't find any solutions that combined editing quality with the ability to process multiple images simultaneously.**

This project aims to fill that gap, allowing users to edit a sample image, save the settings applied to it, and then automatically replicate those adjustments across dozens or hundreds of photos.

# Initial Overview

---

At the beginning of the project, the idea was to create a window that was as simple as possible, in which it would be possible to use a photograph to apply edits and offer the functionality of **cropping the image**. However, after a few days of development, it became clear that implementing cropping would be more complex and impractical, so the goal was adjusted to focus only on image editing.

From the outset, it was clear that an initial section would be needed to allow the user to upload the **sample photograph**. Initially, the idea was to program the selection via a file path, but with the use of **Tkinter**, it became possible to insert the image directly from the computer's memory, making the process more intuitive.

Another essential aspect was to provide the user with sliders to adjust the brightness and contrast of the photograph. For this purpose, we used the **alpha** and **beta** functions of the **OpenCV** library.

Next, there was a need to create a function that would save the settings used, allowing the same edits to be reapplied later. Based on this, a function was also implemented that could open a folder, iterate over the images in a loop, apply the saved settings, and save the results. Initially, we thought about creating a separate final folder, but we ended up choosing to save the files in the same folder and change the names to avoid overwriting.

Finally, **Tkinter** was responsible for the aesthetic aspect, creating a practical and functional window that made the program easy to use for any user.

# Libraries Used

---

## 1. Open CV

Advanced computer vision and image processing library. In this project, it is used to efficiently manipulate image arrays.

Some of the functions used were:

`cv2.imread()` - Reads an image and returns a **NumPy** array.

`cv2.resize()` - Resize the image to a specific width and height.

`INTER_AREA` is recommended for reducing size while preserving details.

`cv2.convertScaleAbs()` - Performs linear pixel adjustment. **Alpha** controls contrast and **Beta** controls brightness.

`cv2.imwrite()` - Saves the image array to disk in a supported format.

```
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()
```

Image 1 - Examples of using the **Open CV** library

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

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

Image 2 - Examples of using the **Open CV** library

## 2. Pillow

**Pillow** was used as a bridge between **Open CV** and **Tkinter**.

A library for image handling.

Some of the functions used were:

`Image.fromarray()` - Converts a **NumPy** array into a **Pillow Image** object.

`cv2.cvtColor(cv2.COLOR_BGR2RGB)` - Converts an image from **BGR (Open CV)** to **RGB (Pillow)**.

`ImageTk.PhotoImage()` - Converts **Pillow Image** into an image compatible with **Tkinter**.

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

Imagem 3 - Examples of using the **Pillow** library

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

Image 4 - Examples of using the **Pillow** library

### 3. Tkinter

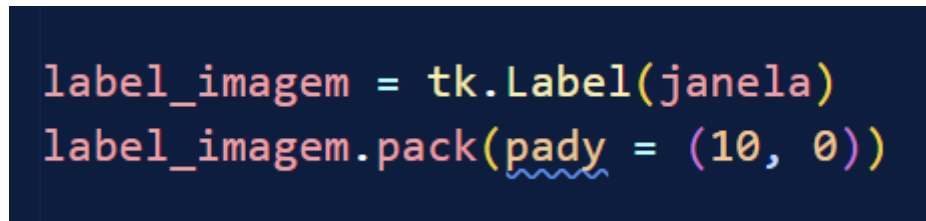
Standard Python library for creating graphical user interfaces **(GUI)**.

Some of the functions used were:

`filedialog.askopenfilename()` - Opens the file explorer to choose an individual image. You can use `filetypes` to limit the types of files that can be selected.

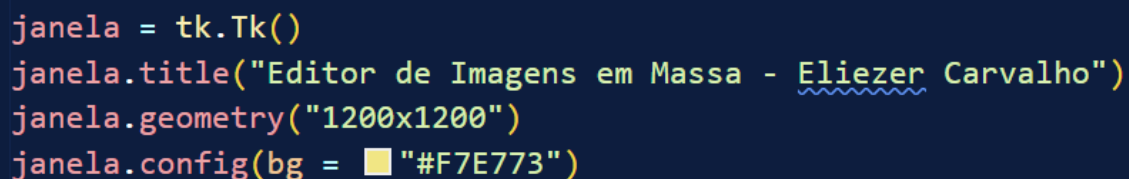
`label_imagem = tk.Label(janela)` - Widget used to display the converted image via `Pillow ImageTk`.

`janela.mainloop()` - Keep the window open and listen to the events.



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

Image 6 - Examples of using the *Tkinter* library



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

Image 7 - Examples of using the *Tkinter* library

## 4. Pathlib

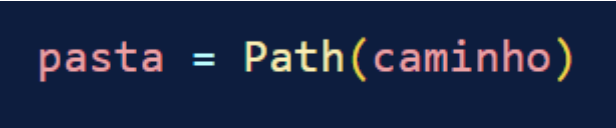
Pathlib facilitates the **manipulation** of paths and files.

Some of the functions used were:

`Path()` - Creates an object representing directories or files.

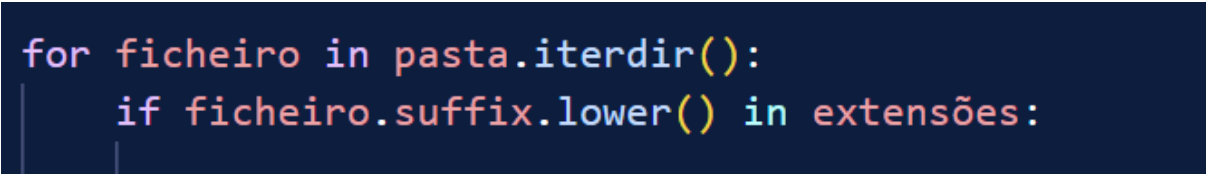
`pasta.iterdir()` - Itera over all files within the directory, allowing each image to be processed.

`saida.mkdir(exist_ok = True)` - Creates a valid folder. `exist_ok = True` prevents errors if a folder with the same name already exists.



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

Image 8 - Examples of using the *Pathlib* library



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

Image 9 - Examples of using the *Pathlib* library



## Detailed Explanation of the Code

---

### `def (carregar_imagem)`

The user selects an image, OpenCV reads the image as an array and resizes it for display in the interface.

At the end of the code block, it calls `ajustar_imagem()` to apply initial brightness/contrast.

### `def (ajustar_imagem)`

Reads `alpha` (contrast) and `beta` (brightness) values from **Tkinter** sliders and applies adjustments with `cv2.convertScaleAbs`.

Converts to **Pillow Image** and then to `ImageTk.PhotoImage` and updates the interface `Label` to show the edited image.

### `def (guardar_def)`

Stores `alpha` and `beta` in a global dictionary **definições** to apply the same adjustments to other images.

## def (aplicar\_pasta)

It allows the user to select a folder, create an output folder, and iterate over all files in the folder with valid extensions.

It reads each image and applies the saved adjustments (**definições**) with **OpenCV**.

Finally, the image is saved in the output folder, preserving the original files.

## def (tkinter)

The sliders are directly linked to the **ajustar\_imagem()** function, allowing **real-time visualization of changes**.

The buttons trigger functions that manipulate images (**carregar\_imagem**, **guardar\_def**, **aplicar\_pasta**).

Widgets are arranged vertically in the order they were added.

It can be expanded in the future using **grid()** or **place()** for more complex layouts.

Consistent use of colors (**bg**, **fg**, **troughcolor**) and fonts (**font=(“Arial”, 9, “bold”)**) ensures a **unified theme**.

# Challenges Encountered

---

Initially, I had some difficulty understanding the **Tkinter** library, as it seemed a little outdated in terms of structure and flow. Implementing the image display and its real-time updating as the user adjusted the brightness and contrast became a real challenge.

Another major obstacle arose from the fact that, after editing, the image was transformed into a **NumPy array** by **OpenCV**, which made it impossible to display it directly in **Tkinter** without conversion. Finding the right way to perform this conversion with **Pillow** + **ImageTk** took some time and experimentation.

In addition, the initial idea of implementing automatic bulk cropping proved to be quite complex. A lot of time was invested in this feature, but due to the difficulty of implementation and the need to complete bulk editing in a functional way, it was eventually abandoned.

# Results Achieved

---

The project achieved its proposed objectives, providing a **practical and functional** solution for bulk image editing. Among the main results obtained, the following stand out:

## Real-time editing

Brightness and contrast adjustments made using the sliders in **Tkinter** are applied instantly to the sample image, allowing the user to view the changes before applying them to multiple photos.

## Efficient batch processing

The program can iterate over **dozens** or **hundreds** of images within a folder, applying the same editing settings consistently and preserving the original files.

## Intuitive interface

The use of **Tkinter** allowed us to create a simple and functional window, where all actions—loading images, adjusting parameters, saving settings, and applying them to an entire folder—are easily accessible.

## Safe format conversion

The use of **OpenCV** and **Pillow** ensured that images were processed and displayed correctly, overcoming the challenge of compatibility between **NumPy arrays** and **Tkinter**.

In short, this project shows that it is possible to combine ease of use, efficient processing, and consistent visual results, even with a large volume of images.

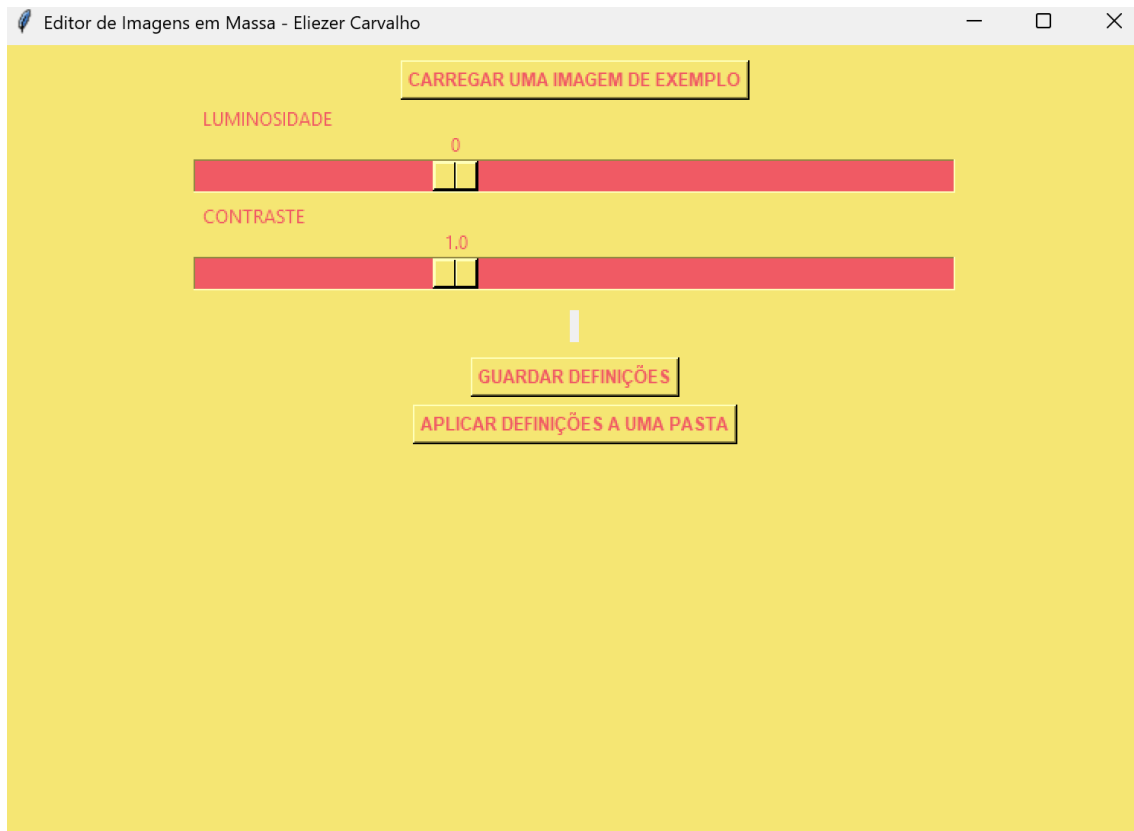


Image 9 - Final Results