**Documentation**

**Image Processing App**

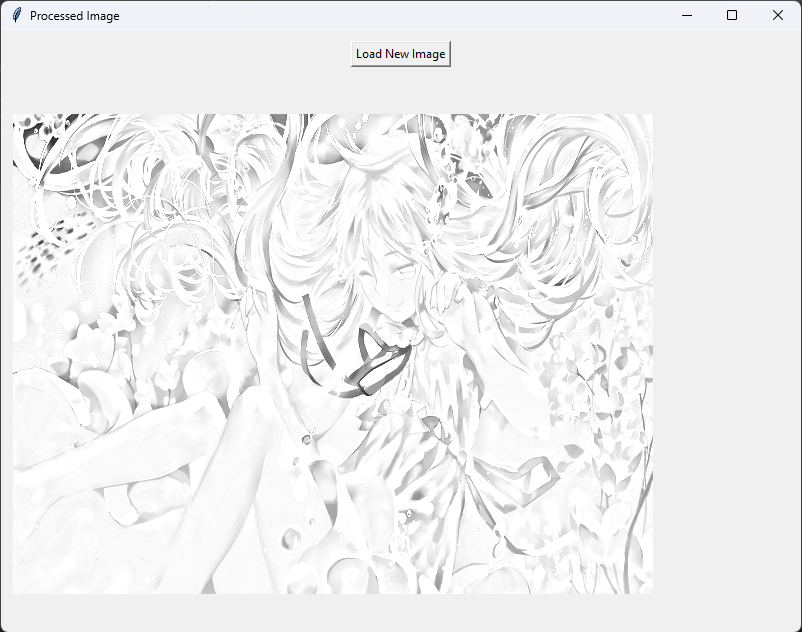


**Submitted by:**

Carl Michael C. De Taza

Sue C. Sinangote

David Frank Edison Regodos

****



**Prompt:** Introduction

The Image Processing App is a straightforward desktop application crafted with Python and the Tkinter GUI library. Designed to cater to users of varying technical backgrounds, the application enables users to seamlessly load an image, subject it to a sequence of image processing operations, and observe processed images in real-time.

**Processing Workflow**

The core functionality of the Image Processing App hinges on a predefined sequence of image processing operations implemented in the Python code. Upon loading an image, the application employs the OpenCV library to transform the image through a series of steps:

* **Grayscale Conversion (cv2.cvtColor):** The application converts the original color image into a grayscale representation.
* **Color Inversion (cv2.bitwise\_not):** The colors of the grayscale image are inverted to produce a negative effect.
* **Median Blur (cv2.medianBlur):** A median blur is applied to smooth out the image, reducing noise and enhancing visual appeal.
* **Color Inversion (Again):** The colors are inverted once more to revert the image to its original orientation.
* **Sketch Effect (cv2.divide):** The final step involves creating a sketch-like effect by dividing the grayscale image by its inverted and smoothed counterpart.

**Usage of the Application**

The primary use case of the Image Processing App lies in its simplicity and accessibility for users seeking to perform basic image manipulations. The application finds utility in various scenarios, including:

* **Quick Image Enhancement:** Users can effortlessly load images and apply fundamental processing operations to enhance visual quality or achieve artistic effects.
* **Educational Tool:** The application serves as an educational tool for individuals learning about image processing algorithms and their practical implementation.



**Prompt:** Advantage and Disadvantage

**Advantages of the Image Processing App:**

* **User-Friendly Interface:** The application provides a simple and intuitive graphical user interface (GUI) through Tkinter, making it accessible to users with varying levels of technical expertise.

**Disadvantages of the Image Processing App:**

* **Limited Image Processing Operations:** The application currently supports a basic set of image processing operations. Users looking for more advanced features may find the application lacking in terms of functionality.



**Prompt:** Recommendations

**Expand Image Processing Functionality:** To enhance the application, consider adding a broader range of image processing operations. This can be achieved by incorporating additional OpenCV functions or integrating external libraries for advanced image manipulation.

**Include User Instructions:** Provide informative tooltips or a separate help section to guide users on how to interact with the application effectively. This will enhance the user experience and ensure users understand the available features.



**Prompt:** System Requirements

**Hardware Requirements:**

* No specific hardware requirements beyond a standard computer with a display.

**Software Requirements:**

|  |  |
| --- | --- |
| **Name** | **Descriptions** |
| **Python 3.x** | Python 3.x, the latest version, brings key improvements like Unicode support, enhanced syntax, and asyncio. It's actively maintained, secure, and backward incompatible with Python 2.x. Widely used in web development, AI, and more, Python 3.x is the recommended choice for new projects. |
| **OpenCV (`cv2` module)** | OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. The cv2 module, often referred to as OpenCV in Python, is the Python interface for OpenCV. OpenCV provides a wide range of tools and functions for computer vision tasks, image and video processing, and machine learning. |
| **Tkinter** | Tkinter is the standard GUI (Graphical User Interface) toolkit that comes with Python. It provides a set of tools and widgets for creating graphical user interfaces and is based on the Tk GUI toolkit. Tkinter allows developers to create windows, dialogs, buttons, textboxes, and other GUI elements for their Python applications. |
| **Pillow (PIL)** | Pillow, originally known as the Python Imaging Library (PIL), is an open-source Python imaging library that adds image processing capabilities to the Python interpreter. It serves as an extension and enhancement of the original PIL library, providing a user-friendly and powerful set of tools for opening, manipulating, and saving various image file formats. |



**Prompt:** Installation

**Python**

Ensure that Python 3.x is installed on your system. You can download it from the official Python website: [Python Downloads](<https://www.python.org/downloads/>)

**Create a virtual environment(optional)**

* **Open Command Prompt:**

Open the Command Prompt on your Windows desktop.

* **Navigate to the Desired Directory:**

Use the **cd** command to navigate to the directory where you want to create the virtual environment. For example:



* **Create Virtual Environment:**

Run the following command to create a virtual environment named **env**:

****

* **Activate Virtual Environment:**

To activate the virtual environment, run the appropriate activation script. In the Command Prompt:****

**Clone Repository from Github**

* **Get the Repository URL:**

On the GitHub repository page, click on the "Code" button. Make sure to select the "HTTPS" option. Copy the repository URL provided: <https://github.com/Chael07/drill2_Image_Processing.git>

* **Open Terminal Command Prompt (Windows):**

Open a terminal or command prompt on your local machine.

* **Navigate to the Directory Where You Want to Clone the Repository:**

Use the cd command to navigate to the directory where you want to store the cloned repository. For example:****

* **Clone the Repository:**

Use the git clone command followed by the repository URL. Replace <https://github.com/Chael07/drill2_Image_Processing.git> with the URL you copied in step 1.

**Required Python Packages**

Install the required Python packages in same directory using the following commands:





**Prompt:** Application Overview

The application provides a simple graphical interface for loading images and applying basic image processing operations.

**Features**

* Load New Image: Allows the user to load a new image from their computer.
* Image Processing: Applies a series of image processing operations to the loaded image.
* Display: Shows processed images side.

**Code Structure**

The code is organized into a single Python script (`sketch.py`) containing a class `ImageProcessorApp`. Here's an overview of the code structure:

**Imported Libraries:**

* `cv2`: OpenCV library for image processing.
* `tkinter`: Tkinter library for GUI.
* `filedialog` from Tkinter: For opening file dialogs.
* `Image` and `ImageTk` from PIL: For handling images in Tkinter.

**Class: `ImageProcessorApp`**

* Initialization (`\_\_init\_\_`):
* Sets up the Tkinter root window.
* Creates the main GUI components.

**`create\_widgets` Method:**

* Creates and configures the "Load New Image" button.
* Creates a placeholder label for displaying images.

‘**load\_new\_image` Method:**

* Opens a file dialog for selecting an image file.
* Reads, resizes, and displays processed images.

**process\_image` Method:**

* Contains the image processing operations.
* Converts the image to grayscale, inverts colors, applies median blur, and creates a sketch effect.

**display\_image` Method:**

* Converts the processed image to Tkinter format and displays it in the GUI.
* Updates the window title with the appropriate title.

**`\_\_main\_\_` Block:**

* Creates an instance of the `ImageProcessorApp` class.
* Initiates the Tkinter main loop.



**Prompt:** Conclusion

The Image Processing App offers a user-friendly interface for basic image manipulation. By expanding functionality, improving user guidance, and optimizing the code structure, the application can evolve into a more versatile tool for users with diverse image processing needs.



**Prompt:** Source Code Used in the Project

import cv2

import tkinter as tk

from tkinter import filedialog

from PIL import Image, ImageTk

class ImageProcessorApp:

    def \_\_init\_\_(self, root):

        self.root = root

        self.root.title("Image Processing App")

        self.root.geometry("800x600")

        self.create\_widgets()

    def create\_widgets(self):

        # Create and configure the "Load New Image" button

        load\_button = tk.Button(self.root, text="Load New Image", command=self.load\_new\_image)

        load\_button.pack(pady=10)

        # Placeholder for displaying images

        self.image\_label = tk.Label(self.root)

        self.image\_label.pack(side="left", padx=10)

    def load\_new\_image(self):

        file\_path = filedialog.askopenfilename(filetypes=[("Image files", "\*.jpg;\*.png;\*.jpeg")])

        if file\_path:

            img = cv2.imread(file\_path)

            img = cv2.resize(img, (640, 480))

            # Display the original image

            self.display\_image(img, "Original Image")

            # Process and display the processed image

            processed\_img = self.process\_image(img)

            self.display\_image(processed\_img, "Processed Image")

    def process\_image(self, img):

        # Your image processing code

        gray\_img = cv2.cvtColor(src=img, code=cv2.COLOR\_BGR2GRAY)

        invert\_img = cv2.bitwise\_not(src=gray\_img)

        smooth\_img = cv2.medianBlur(src=invert\_img, ksize=27)

        ivt\_smooth\_img = cv2.bitwise\_not(smooth\_img)

        sketch\_img = cv2.divide(gray\_img, ivt\_smooth\_img, scale=250)

        return sketch\_img

    def display\_image(self, image, title):

        image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

        image = Image.fromarray(image)

        photo = ImageTk.PhotoImage(image=image)

        self.image\_label.config(image=photo)

        self.image\_label.image = photo  # Keep a reference to the image to prevent it from being garbage collected

        self.root.title(title)

if \_\_name\_\_ == "\_\_main\_\_":

    root = tk.Tk()

    app = ImageProcessorApp(root)

    root.mainloop()

**END**