Image to Pencil Sketch Converter

Project Documentation

Team Members

- Abdelrhman Sherif Mahmoud
- Ahmed Osama Mohammed Khairy
- Omar Arshad Mohammed
- Wael Bahaa Aldien Mostafa
- Mohamed Ali Hassan Aref

Table of Contents

1. Introduction

This project demonstrates an image processing application that converts a given image into a pencil sketch. Using OpenCV and other Python libraries, it performs a series of transformations on the image and displays each step. This documentation describes the technical and functional details of the project.

2. Features

- Interactive image upload within Google Colab.
- Image transformation to grayscale.
- Inversion of grayscale image to highlight features.
- Gaussian blurring to soften the image.
- Inversion of the blurred image for sketch effect.
- Blending the original grayscale with the processed image to generate the final pencil sketch.
- Visualization of each transformation step using Matplotlib.

3. System Requirements

- Python 3.x
- Google Colab or Jupyter Notebook (Recommended for display functions)
- OpenCV
- NumPy
- Matplotlib

4. Installation Guide

Step 1: Clone the repository using Git:

```
git clone https://github.com/WaelAlfnan/Image_to_Sketch.git
```

Step 2: Navigate to the project directory:

```
cd Image_to_Sketch
```

Step 3: Install required Python libraries:

```
pip install opencv-python-headless numpy matplotlib
```

5. Usage Instructions

1. Run the script in Google Colab.
2. The notebook will prompt you to upload an image.

3. The script processes the image and displays six different stages of transformation.
4. Final output is a pencil sketch version of your uploaded image.

6. Workflow Explanation

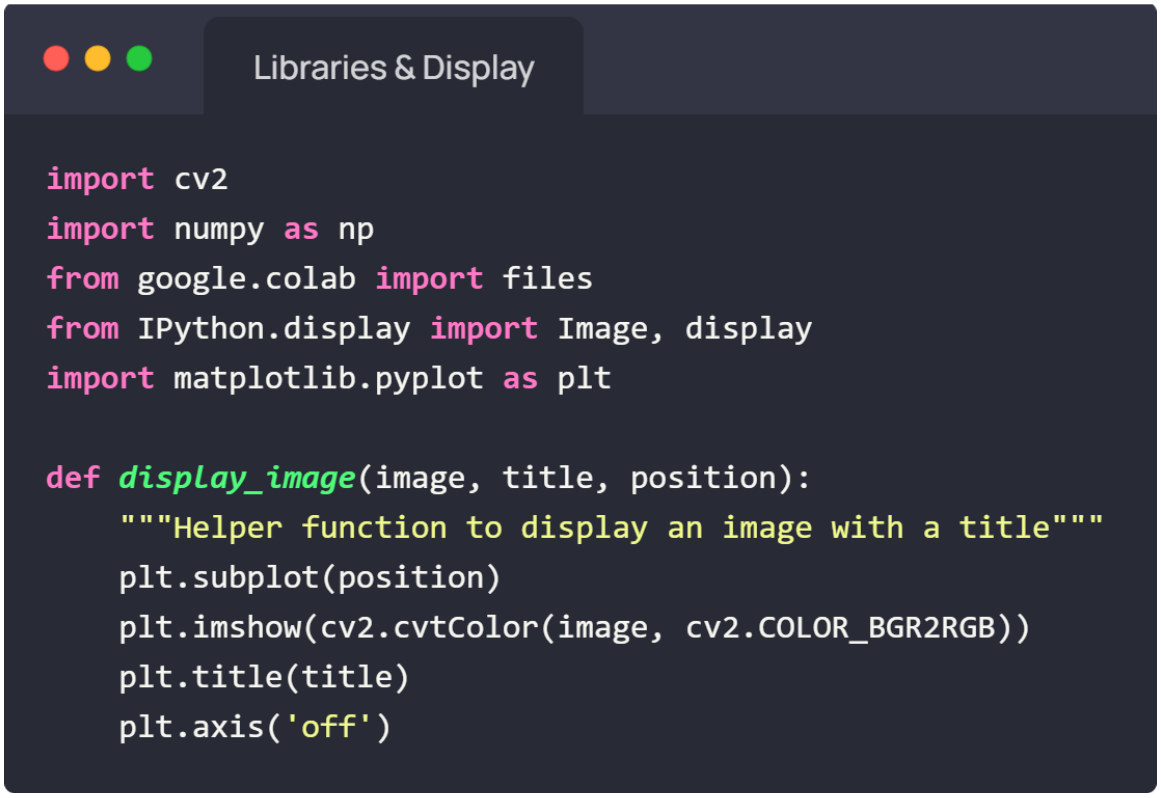
11. Original Image: The uploaded image.
12. Grayscale Conversion: Simplifies image by reducing color information.
13. Image Inversion: Inverts grayscale to create negative effect.
14. Gaussian Blur: Smoothens image to prepare for sketch blending.
15. Inverted Blur: Re-inverts the blurred image to simulate pencil shades.
16. Sketch Creation: Divides grayscale by inverted blur to create final sketch.

7. Example Workflow Output

Here are the six output stages from the sketch creation process:
1. Original Image
2. Grayscale Image
3. Inverted Grayscale Image
4. Blurred Image
5. Inverted Blurred Image
6. Final Pencil Sketch

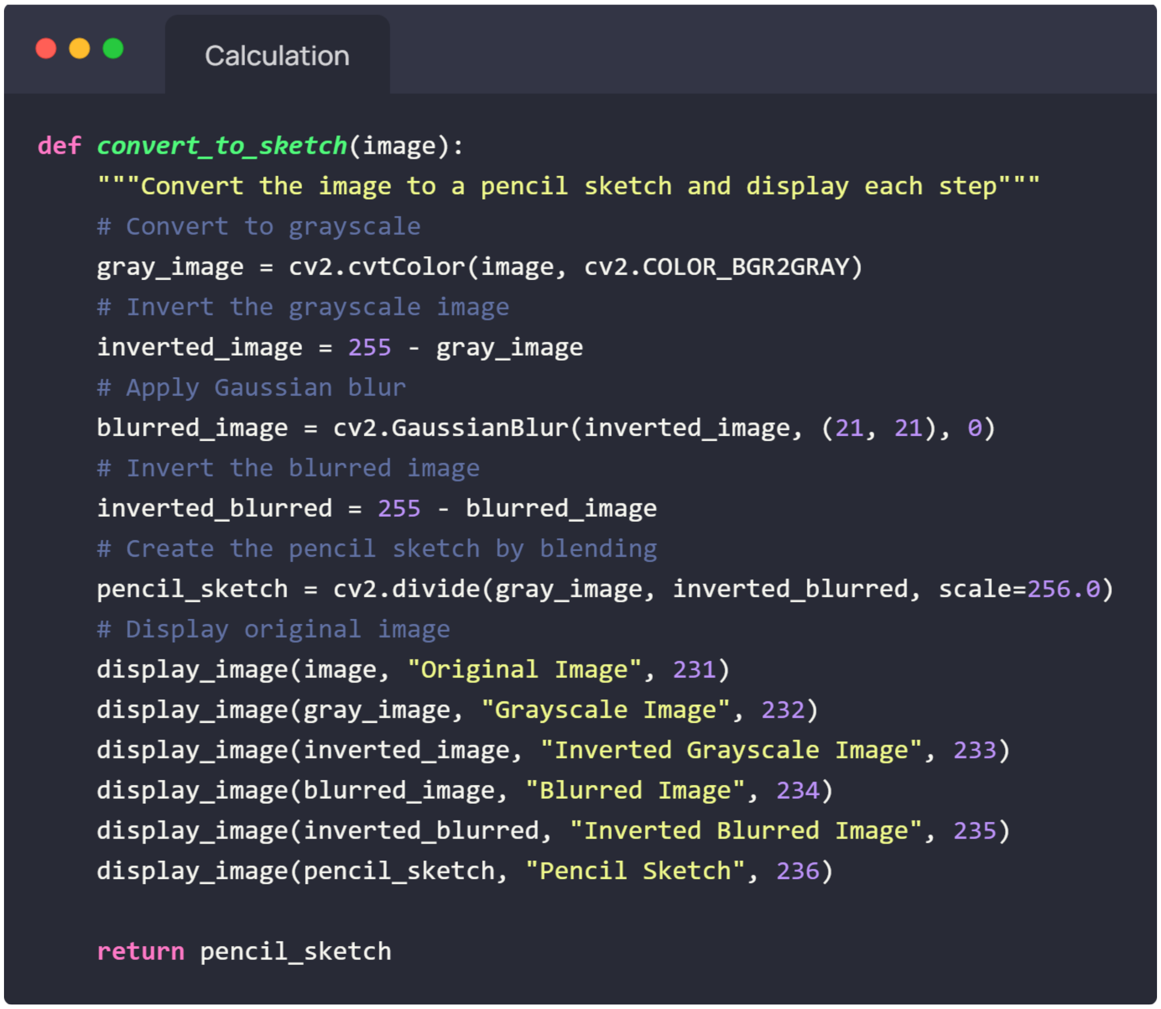
8. Code Breakdown

The following Python code demonstrates the full sketch conversion process:



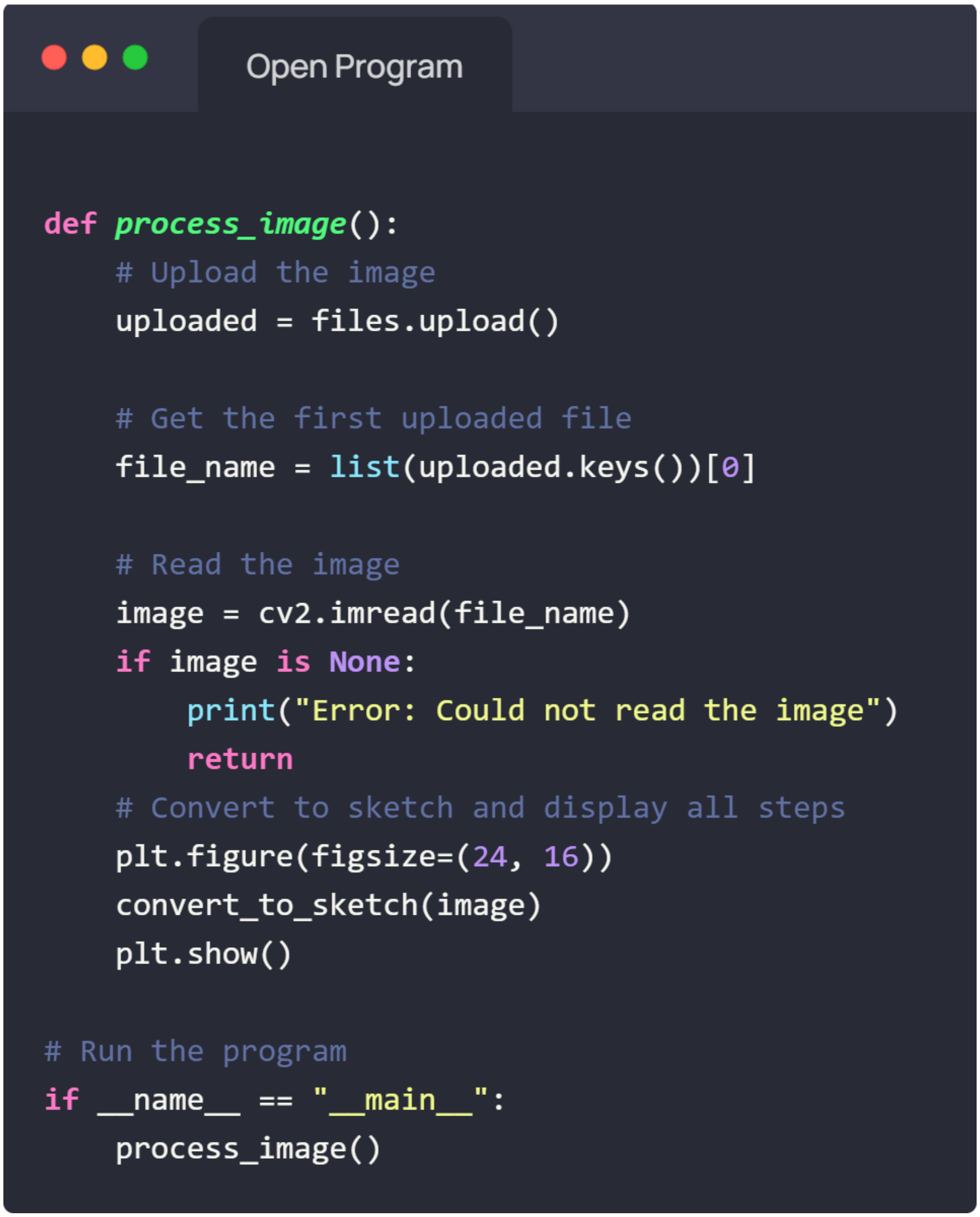
```python
import cv2
import numpy as np
from google.colab import files
from IPython.display import Image, display
import matplotlib.pyplot as plt

def display_image(image, title, position):
    """Helper function to display an image with a title"""
    plt.subplot(position)
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title(title)
    plt.axis('off')
```



```python
def convert_to_sketch(image):
    """Convert the image to a pencil sketch and display each step"""
    # Convert to grayscale
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Invert the grayscale image
    inverted_image = 255 - gray_image
    # Apply Gaussian blur
    blurred_image = cv2.GaussianBlur(inverted_image, (21, 21), 0)
    # Invert the blurred image
    inverted_blurred = 255 - blurred_image
    # Create the pencil sketch by blending
    pencil_sketch = cv2.divide(gray_image, inverted_blurred, scale=256.0)
    # Display original image
    display_image(image, "Original Image", 231)
    display_image(gray_image, "Grayscale Image", 232)
    display_image(inverted_image, "Inverted Grayscale Image", 233)
    display_image(blurred_image, "Blurred Image", 234)
    display_image(inverted_blurred, "Inverted Blurred Image", 235)
    display_image(pencil_sketch, "Pencil Sketch", 236)

    return pencil_sketch
```



```python
def process_image():
    # Upload the image
    uploaded = files.upload()

    # Get the first uploaded file
    file_name = list(uploaded.keys())[0]

    # Read the image
    image = cv2.imread(file_name)
    if image is None:
        print("Error: Could not read the image")
        return
    # Convert to sketch and display all steps
    plt.figure(figsize=(24, 16))
    convert_to_sketch(image)
    plt.show()

# Run the program
if __name__ == "__main__":
    process_image()
```

9. License

This project is licensed under the MIT License. See the LICENSE file for more details.

10. Why These Libraries Were Used

- OpenCV (cv2):

Used for image processing operations such as reading the image, converting to grayscale, inverting the image, applying Gaussian blur, and performing the division to create the sketch. It provides fast, efficient, and comprehensive tools for image manipulation.

- NumPy:

Used for numerical operations on image arrays. It provides high-performance operations that are essential for manipulating pixel values and performing mathematical transformations efficiently.

- Matplotlib:

Used for visualizing the steps in the image transformation process. It allows displaying multiple images in a subplot layout, helping users understand how each stage affects the final output.

- Google Colab:

A cloud-based platform that simplifies running Python code interactively. It facilitates file uploads and displays results inline, making it ideal for demonstration and visualization

11. Process Flow Diagram

The following diagram outlines the image processing pipeline from upload to sketch output:

[User Uploads Image]
↓
[Convert to Grayscale]
↓
[Invert Grayscale Image]
↓
[Apply Gaussian Blur]
↓
[Invert Blurred Image]
↓
[Blend with Grayscale Image using cv2.divide]
↓
[Final Pencil Sketch Output]

---

12. Comparison with Other Methods

Below is a comparison of this method with two other common image sketching techniques:

Technique	Description	Pros	Cons
OpenCV Division (This Project)	Uses grayscale + invert + blur + divide to simulate pencil shading.	Simple, real-time, visually pleasing results.	Lacks fine-art realism or color nuance.
Canny Edge Detection	Detects edges to simulate sketch contours.	Good for outlines, fast.	No shading or pencil fill effects.
Deep Learning Style Transfer	Uses pre-trained models to apply artistic filters.	Highly realistic, mimics hand drawing.	Requires high computational resources and dataset training.