ShivaprasadGoli-Coder /
Python-Image-Processing

Code Issues Pull requests Actions Projects Security Insights Settings

0 stars 0 forks 1 watching Branches Activity
Tags

Private repository

1 Branch 0 Tags Go to file t Go to file Add file Code

ShivaprasadGoli-Coder Update README.md 7382f16 · 2 minutes ago

.vs/ImageProcessPart1	Initial commit - Added image proces...	2 hours ago
ImageProcessPart1	Added new line 102	1 hour ago
ImageProcessPart1.sln	Initial commit - Added image proces...	2 hours ago
README.md	Update README.md	2 minutes ago

README

Explanation of the Code

This Python script performs image processing using OpenCV and displays the results in a Tkinter GUI. The goal of the script is to:

- Load an image,
- Detect green spots,
- Segment the image into different regions, and
- Display the results in a Tkinter window.

Table of Contents

1. Installation Instructions

1. Install Python (If Not Installed)

Check if Python is installed by running the following command in Command Prompt (cmd):

```bash
python --version
```

- If Python is not installed, download and install it from the Python official website.
- Important: Make sure to check the box "Add Python to PATH" during installation.

2. Install Required Python Libraries

- To run the script, install the required libraries using pip.
- Open Command Prompt (cmd) or PowerShell and run:

```bash
pip install opencv-python numpy pillow tkinter
```

Breakdown of Required Libraries:

- Library Purpose
- opencv-python For image processing (OpenCV functions)
- numpy For handling arrays and numerical operations
- pillow For image conversion and handling (PIL)
- tkinter Built-in with Python (used for GUI display)

Note: If tkinter is missing, install it separately using:

```bash
pip install tk
```

3. Copy the Script to the New Laptop

- Transfer the Python script (.py file) and the image file (.png or other format) to the other laptop. Make sure the correct file path is set for the image in the script:

```python
image_path = r"C:\Users\user<name>\Desktop\Python Project\image1.png"
```

- Modify this path to match the correct location on the new laptop.

4. Run the Script

- Once everything is installed, run the script from Command Prompt:

```
bash
python "C:\Path\To\Your\Script.py"
```

- Replace "C:\Path\To\Your\Script.py" with the actual path where your script is located.

5. Troubleshooting Common Issues

- A. If pip is not recognized: Run this command to update pip:

```
bash
python -m ensurepip --default-pip
python -m pip install --upgrade pip
```

- B. If opencv-python Fails to Install: Try installing an alternative package:

```
bash
pip install opencv-python-headless
```

C. If tkinter Window Does Not Open:

Ensure Tkinter is installed correctly by running:

```
bash
python -m tkinter
```

- This should open a Tkinter test window.

Summary of Installation Steps

- Install Python (if not installed).
- Install required libraries using:

```
  bash
pip install opencv-python numpy pillow tkinter
```

- Copy the script and image to the new laptop.
- Run the script using:

```
bash
python script.py
```

2. Import Necessary Libraries

```python
import cv2
import numpy as np
import tkinter as tk
from tkinter import messagebox
from PIL import Image, ImageTk  # To display images in Tkinter

# 3. Import Necessary Libraries
```python
import cv2
import numpy as np
import tkinter as tk
from tkinter import messagebox
from PIL import Image, ImageTk  # To display images in Tkinter
```

- cv2: OpenCV library for image processing.
- numpy: Used for numerical operations (such as defining color ranges).
- tkinter: GUI library to display results.
- PIL (Pillow): Used to convert images to formats that Tkinter can display.

4. Load and Read the Image

```python
image_path = r"C:\Users\golis\Desktop\Python Project\image1.png"  # Replace with actual file path
image = cv2.imread(image_path)
```

- cv2.imread(image_path): Loads the image from the given file path.
- The raw image is stored in the image variable.

```python
if image is None:
    print("Error: Image not found or unable to load.")
else:
    print("Image loaded successfully.")
```

- If the image fails to load, it prints an error message.
- If successful, it prints a confirmation message.

5. Convert the Image to RGB and HSV Formats

```python
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image_hsv = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2HSV)
```

- OpenCV loads images in BGR (Blue-Green-Red) format.
- cv2.cvtColor(image, cv2.COLOR_BGR2RGB): Converts BGR to RGB (used for display).
- cv2.cvtColor(image_rgb, cv2.COLOR_RGB2HSV): Converts RGB to HSV (used for color-based segmentation).

6. Detect Green Spots in the Image

```python
lower_green = np.array([40, 100, 100])
upper_green = np.array([90, 255, 255])
```
```

- mask = cv2.inRange(image_hsv, lower_green, upper_green)
- The HSV color space is used for color-based segmentation.
- lower_green and upper_green define the range of "green" colors.
- cv2.inRange(image_hsv, lower_green, upper_green):
- Creates a binary mask where green areas are white (255) and other areas are black (0).

7. Display the Masked Image (For Debugging)

```
cv2.imshow('Mask', mask)
cv2.waitKey(0)  # Wait until a key is pressed
cv2.destroyAllWindows()
```

- cv2.imshow('Mask', mask): Displays the detected green regions.
- cv2.waitKey(0): Waits indefinitely for a key press.
- cv2.destroyAllWindows(): Closes the image window after key press.

8. Find Contours of Green Spots

```
contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

- cv2.findContours(): Finds all the connected components (blobs of green pixels).
- cv2.RETR_EXTERNAL: Retrieves only the outermost contours.
- cv2.CHAIN_APPROX_SIMPLE: Removes unnecessary points in the contour representation.

9. Segment the Image into Organs

```
height, width, _ = image.shape
row_mid = height // 2  # Divide 5um MPs (top) and 20um MPs (bottom)
col_thirds = [width // 3, 2 * (width // 3)]  # Divide Liver, Kidney, Gut
```

- Divides the image into different regions:
- row_mid: Splits the image into top (5µm MPs) and bottom (20µm MPs). c- ol_thirds: Splits the width into three equal parts (Liver, Kidney, Gut).

```
organ_masks = {
    "Liver_5um": mask[0:row_mid, 0:col_thirds[0]],
    "Kidney_5um": mask[0:row_mid, col_thirds[0]:col_thirds[1]],
    "Gut_5um": mask[0:row_mid, col_thirds[1]:width],
    "Liver_20um": mask[row_mid:height, 0:col_thirds[0]],
    "Kidney_20um": mask[row_mid:height, col_thirds[0]:col_thirds[1]],
    "Gut_20um": mask[row_mid:height, col_thirds[1]:width],
}
```

- Creates separate masks for each organ at two different depths (5µm and 20µm).

10. Count and Classify Green Spots

```
organ_results = {}
for organ, organ_mask in organ_masks.items():
    contours, _ = cv2.findContours(organ_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    organ_results[organ] = {
        "total": len(contours),
        "size_distribution": {"small (<50px)": 0, "medium (50-200px)": 0, "large (>200px)": 0},
    }
```

- Counts the number of green spots per organ.
- Stores the total count and a size distribution dictionary.

Classify Green Spots by Size

```
for contour in contours:
        area = cv2.contourArea(contour)
        if area < 50:
            organ_results[organ]["size_distribution"]["small (<50px)"] += 1
        elif 50 <= area <= 200:
            organ_results[organ]["size_distribution"]["medium (50-200px)"] += 1
        else:
            organ_results[organ]["size_distribution"]["large (>200px)"] += 1
```

- Small: Area < 50 pixels
- Medium: 50 ≤ Area ≤ 200 pixels
- Large: Area > 200 pixels

11. Prepare the Results Text

```
result_text = ""
for organ, data in organ_results.items():
    result_text += f"{organ}: {data['total']} green spots\n"
    result_text += f"   Small: {data['size_distribution']['small (<50px)']}\n"
    result_text += f"   Medium: {data['size_distribution']['medium (50-200px)']}\n"
    result_text += f"   Large: {data['size_distribution']['large (>200px)']}\n"
    result_text += "-" * 40 + "\n"
```

- Formats the results into a text-based summary.
- The results will be displayed later in a popup window.

12. Display the Results in Tkinter

```
def show_results_and_image():
    # Create the root Tkinter window
    root = tk.Tk()
    root.title("Image Processing Results")
```

- Creates a Tkinter window to show the results.

```
    # Show the result in a message box
    messagebox.showinfo("Green Spots Analysis Results", result_text)
```

- Displays a popup message box with the green spot count.

```
# Convert the mask image to a format suitable for Tkinter
mask_image_pil = Image.fromarray(mask)
mask_image_tk = ImageTk.PhotoImage(mask_image_pil)
```

```
# Create a label to display the image
label = tk.Label(root, image=mask_image_tk)
label.image = mask_image_tk  # Keep a reference to avoid garbage collection
label.pack()
```

- Converts the mask image into a format Tkinter can display.
- Adds the image to the Tkinter window.

```
# Start the Tkinter event loop to display the window
root.mainloop()
```

- Runs the Tkinter event loop (keeps the window open).
-

13. Call the Function to Display the GUI

```
show_results_and_image()
```

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

- **Python** 100.0%

Suggested workflows

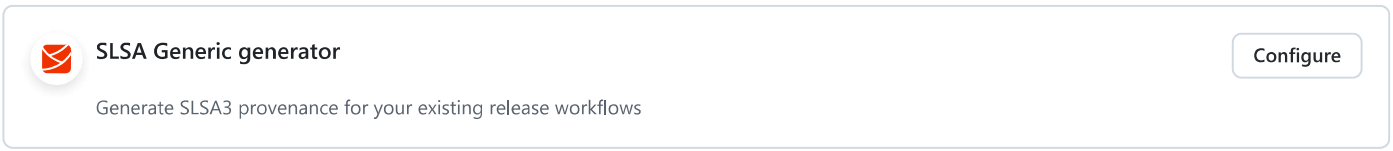
Based on your tech stack

Python Package using Anaconda — Configure

Create and test a Python package on multiple Python versions using Anaconda for package management.

Django — Configure

Build and Test a Django Project



More workflows

Dismiss suggestions