Image Corrosion

Example Results

Run this section's example code [Source Code/06.cv_lite/16.rgb888_erode.py]

In this section, we will use the `cv_lite` extension module to implement erosion of an RGB888 image on the K230.

Principle Explanation

What is Image Erosion?

Image erosion is a morphological image processing operation used to reduce the white area (foreground) in an image, thereby removing small white noise, separating connected objects, or refining object boundaries. Erosion is one of the basic operations in morphological processing and is commonly used on binary or grayscale images.

- **Operation Principle**: Erosion uses a structuring element (often called a convolution kernel or kernel) to slide across the image. Only when all pixels within the kernel's coverage area are white (or have a high value) are the pixels at the kernel's center set to white (or have their maximum value). This causes white areas to shrink inward, thinning their boundaries and removing small isolated white regions.
- **Effect**: The erosion operation can reduce the size of white objects in an image, remove small white noise (such as tiny white dots), and separate adjacent but connected regions. In practical applications, erosion is often used to remove noise, separate objects, or serve as the basis for other complex morphological operations (such as opening).
- **Application Scenarios**: Erosion is very common in image preprocessing, for example, for separating contiguous objects in object detection, removing small noise dots in medical image processing, and thinning text lines in character recognition.

In the erosion operation, the size of the convolution kernel and the number of iterations significantly affect the results: larger kernels and more iterations result in a more pronounced shrinking effect. Furthermore, the binarization threshold (if any) determines which pixels are considered foreground (white) or background (black).

Code Overview

Importing Modules

```
import time, os, sys, gc
from machine import Pin
from media.sensor import * # Camera interface
from media.display import * # Display interface
from media.media import * # Media manager
import _thread
import cv_lite # cv_lite extension module (including erosion interface) / AI CV
extension (erode function)
import ulab.numpy as np # NumPy-like ndarray for MicroPython
```

Setting the Image Size

```
image_shape = [480, 640] # Height x Width
```

Define the image resolution to 480x640 (height x width). The camera and display modules will be initialized based on this size later.

Initialize the camera (RGB888 format)

```
sensor = Sensor(id=2, width=1280, height=960, fps=30)
sensor.reset()
sensor.set_framesize(width=image_shape[1], height=image_shape[0])
sensor.set_pixformat(Sensor.RGB888) # Set pixel format to RGB888
```

- Initialize the camera, set the resolution to 1280x960 and the frame rate to 30 FPS.
- Resize the output frame to 640x480 and set it to RGB888 format (three-channel color image, suitable for morphological operations on color images).

Initialize the display module

```
Display.init(Display.ST7701, width=image_shape[1], height=image_shape[0],
to_ide=True, quality=50)
```

Initialize the display module, using the ST7701 driver chip, with a resolution consistent with the image size. `to_ide=True` indicates that the image will be simultaneously transmitted to the IDE for virtual display, and `quality=50` sets the image transmission quality.

Initialize the media manager and start the camera

```
MediaManager.init()
sensor.run()
```

Initialize the media resource manager and start the camera to capture the image stream.

Set erosion parameters

```
kernel_size = 3 # Kernel size (must be odd, e.g., 3, 5, 7)
iterations = 1 # Number of erosion passes
threshold_value = 100 # Threshold for binarization (0 = Otsu)
clock = time.clock() # Start FPS timer
```

- `kernel_size`: Kernel size, used for the erosion operation. Must be an odd number (e.g., 3, 5, 7). Larger values result in greater shrinkage.
- `iterations`: Number of erosion iterations. Larger values result in greater shrinkage but increase computational effort.
- `threshold_value`: Binarization threshold, used before or after erosion. A value of 0 uses the Otsu thresholding algorithm.
- `clock`: used to calculate frame rate.

Image processing and erosion operation

```
while True:
    clock.tick()  # Start timing / Start frame timing

    # Capture a frame / Capture a frame
    img = sensor.snapshot()
    img_np = img.to_numpy_ref() # Get RGB888 ndarray reference / Get RGB888
ndarray reference (HWC)

    # Apply erosion operation (automatically converted to grayscale) / Apply
erosion (converts RGB to gray internally)
    eroded_np = cv_lite.rgb888_erode(
        image_shape,
        img_np,
        kernel_size,
        iterations,
        threshold_value
    )
```

```
    # Wrap eroded grayscale image for display (grayscale format) / Wrap eroded
grayscale image for display
    img_out = image.Image(image_shape[1], image_shape[0], image.GRAYSCALE,
                          alloc=image.ALLOC_REF, data=eroded_np)

    # Show the eroded image
    Display.show_image(img_out)

    # Cleanup and print FPS
    gc.collect()
    print("erode fps:", clock.fps())
```

- **Image Capture**: Acquire an image frame using `sensor.snapshot()` and convert it to a NumPy array reference using `to_numpy_ref()`.
- **Erosion Processing**: Call `cv_lite.rgb888_erode()` to perform the erosion operation (internaly converting the RGB image to grayscale), returning the processed image as a NumPy array.
- **Image Packaging and Display**: Pack the processed result into a grayscale image object and display it on the screen or in an IDE virtual window.
- **Memory management and frame rate output**: Call `gc.collect()` to clean up the memory, and print the current frame rate through `clock.fps()`.

Resource release

```
sensor.stop()
Display.deinit()
os.exitpoint(os.EXITPOINT_ENABLE_SLEEP)
time.sleep_ms(100)
MediaManager.deinit()
```

Parameter adjustment instructions

- `kernel_size`: Convolution kernel size. A larger value results in a larger shrinkage area, which is suitable for removing larger white noise or separating larger objects. An odd number (such as 3, 5, or 7) must be used. Start with 3 to balance detail preservation and shrinkage effects.
- `iterations`: Number of iterations. A larger value results in a stronger erosion effect, but may cause over-shrinkage and loss of details. It is recommended to start with 1 and gradually increase it according to the image characteristics.
- `threshold_value`: Binarization threshold. Higher values require brighter pixels to be considered foreground. A value of 0 uses Otsu's automatic thresholding. It is recommended to test from 50 to 150 based on the image brightness, or set to 0 for automatic adaptation.