Image Expansion

Example Results

Run this section's example code [Source Code/06.cv_lite/19.rgb888_dilate.py]

In this section, we'll use the `cv_lite` extension module to implement image dilation in RGB888 format on an embedded device.

Principle Explanation

What is Image Dilation?

Image dilation is a morphological image processing operation used to enlarge white areas (foreground) in an image to fill small holes, disconnected areas, or enhance object boundaries. Dilation is one of the fundamental operations in morphological processing and is commonly used on binary or grayscale images.

- **How it works**: Dilation uses a structuring element (often called a convolution kernel) to slide across the image. If at least one pixel within the kernel's coverage area is white (or has a high value), the pixel at the kernel's center is set to white (or its maximum value). This causes the white area to expand outward, thickening its borders and filling small holes.
- **Effect**: The dilation operation can enlarge white objects in an image, connect adjacent but disconnected areas, and remove small black noise (such as tiny black dots). In practical applications, dilation is often used to enhance connectivity, fill gaps, or serve as the basis for other complex morphological operations (such as closing).
- **Application Scenarios**: The dilation operation is very common in image preprocessing, for example, connecting broken edges in object detection, filling small holes in medical image processing, and enhancing text continuity in character recognition.

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

Code Overview

Importing Modules

```
import time, os, gc
from machine import Pin
from media.sensor import * # Camera interface
from media.display import * # Display interface
from media.media import * # Media manager
import cv_lite # cv_lite extension module (including dilation interface) / AI CV
extension (dilate 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)
```

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 transferred to the IDE for virtual display.

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 dilation parameters

```
kernel_size = 3 # Kernel size (preferred odd number, but some implementations
support even numbers)
iterations = 1 # Number of dilation passes
threshold_value = 100 # Binarization threshold (0 = Otsu)
clock = time.clock() # Start FPS timer
```

- `kernel_size`: Kernel size, used for dilation. An odd number is recommended (some implementations support even numbers). Larger values increase the dilation range.
- `iterations`: Number of dilation iterations. Larger values increase the dilation effect but increase the computational effort.
- `threshold_value`: Binarization threshold, used before or after dilation. A value of 0 uses the Otsu thresholding algorithm.
- `clock`: Used for frame rate calculation.

Image processing and dilation 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

    # Apply dilation (converts RGB to gray internally) / Apply dilation (converts
RGB to gray internally)
    result_np = cv_lite.rgb888_dilate(
        image_shape,
        img_np,
        kernel_size,
        iterations,
        threshold_value
    )
    # Wrap dilated grayscale image for display (grayscale format) / Wrap dilated
grayscale image for display
    img_out = image.Image(image_shape[1], image_shape[0], image.GRAYSCALE,
                          alloc=image.ALLOC_REF, data=result_np)

    # Display dilated image
    Display.show_image(img_out)

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

- **Image capture**: Acquire an image frame using `sensor.snapshot()` and convert it to a NumPy array reference using `to_numpy_ref()`.
- **Dilation processing**: Call `cv_lite.rgb888_dilate()` to perform the dilation operation (internaly converting the RGB image to grayscale), returning the processed image as a NumPy array.
- **Image packaging and display**: Package 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 expansion area, which is suitable for processing larger disconnected areas or filling larger holes. It is recommended to use an odd number (such as 3, 5, 7) and start with 3 to balance the details and expansion effect.
- `iterations`: The number of iterations. A larger value results in a stronger expansion effect, but it may cause over-expansion 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.