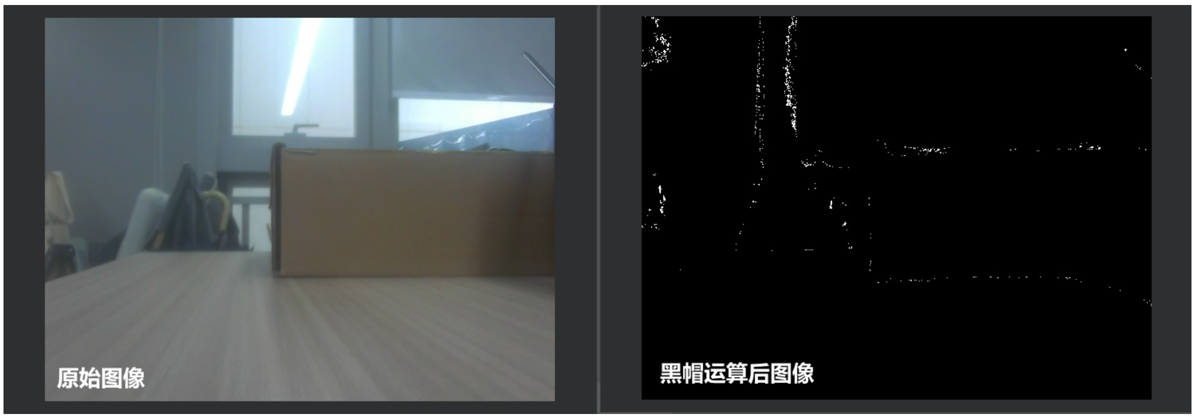
Image Black Hat Operations

Example Results

Run this section's example code [Source Code/06.cv_lite/16.rgb888_blackhat.py]

In this section, we will use the `cv_lite` extension module to implement black-hat operations on RGB888 images on an embedded device.



Principle Explanation

What is Black-Hat Operation on Images?

Black-hat operations are a morphological image processing method used to highlight dark details or small areas in an image, particularly those that are darker than the surrounding area. The result of a black hat operation is the difference between the closing operation and the original image, which can be expressed mathematically as:

Black hat = Closing result - Original image

- **Closing**: Closing is a morphological operation that first dilates an image and then erodes it. This operation can fill small holes or disconnected areas in the image while maintaining the overall shape.
- **Effects of black hat operations**: By subtracting the closing result from the original image, black hat operations can extract details in dark areas that were "filled" or "ignored" during

the closing operation. These areas are typically small dark structures or textures in the image.

Applications of black hat operations in image processing include:

- Detecting dark details or small defects in images.
- Enhancing dark textures or edges in images.
- Using them to separate small, dark objects in object detection.

In practical applications, black hat operation parameters (such as convolution kernel size and number of iterations) affect the range of detail and effectiveness of detection, while the binarization threshold determines which areas in the resulting image are considered "dark."

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 # AI CV extension (Black-Hat functionality)
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.

Initialize the camera (RGB888 format)

```
sensor = Sensor(id=2, width=1280, height=960, fps=90)
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 90 FPS.
- Resize the output frame to 640x480 and set it to RGB888 format (three-channel color image, suitable for black hat 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 black hat operation parameters

```
kernel_size = 3 # Kernel size (odd recommended)
iterations = 1 # Number of morphological passes
threshold_value = 100 # Binarization threshold (0 = Otsu)
clock = time.clock() # Start FPS timer
```

- `kernel_size`: Kernel size, used for morphological operations (dilation and erosion). An odd value is recommended. Larger values increase the processing range and affect the size of detected details.
- `iterations`: Number of iterations, indicating the number of times the morphological operation is repeated. Larger values produce more significant results but increase the computational effort.
- `threshold_value`: Binarization threshold, used to convert the processed image to a binary image. A value of 0 uses the Otsu thresholding algorithm.
- `clock`: used to calculate frame rate.

Image processing and black-hat operation

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

    # Capture a frame and convert to ndarray
    img = sensor.snapshot()
    img_np = img.to_numpy_ref()

    # Apply Black-Hat operation
    # Black-Hat = Closing - Original
    result_np = cv_lite.rgb888_blackhat(
        image_shape,
        img_np,
        kernel_size,
        iterations,
        threshold_value
    )

    # Wrap result as image and display
    img_out = image.Image(image_shape[1], image_shape[0], image.GRAYSCALE,
                          alloc=image.ALLOC_REF, data=result_np)
    Display.show_image(img_out)

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

- **Image Capture**: Acquire an image frame using `sensor.snapshot()` and convert it to a NumPy array reference using `to_numpy_ref()`.

- **Blackhat Operation**: Call `cv_lite.rgb888_blackhat()` to perform blackhat operation, 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 memory and print the current frame rate using `clock.fps()`.

Resource release

```
sensor.stop() # Stop sensor
Display.deinit() # Deinit display
os.exitpoint(os.EXITPOINT_ENABLE_SLEEP) # Set safe exit point
time.sleep_ms(100) # Short delay
MediaManager.deinit() # Release media manager
```

Parameter adjustment instructions

- `kernel_size`: Convolution kernel size. A larger value results in a larger processed area, suitable for detecting larger dark details. It is recommended to use an odd number (such as 3, 5, 7) and start testing with 3.
- `iterations`: Number of operation iterations. A larger value results in a more pronounced morphological operation, but the computational effort increases. It is recommended to start with 1 and gradually increase it based on demand.
- `threshold_value`: Binarization threshold. Higher values require brighter pixels to be retained. A value of 0 uses Otsu's automatic threshold. It is recommended to test from 50 to 150 based on the image brightness, or set it to 0 to use the automatic threshold.