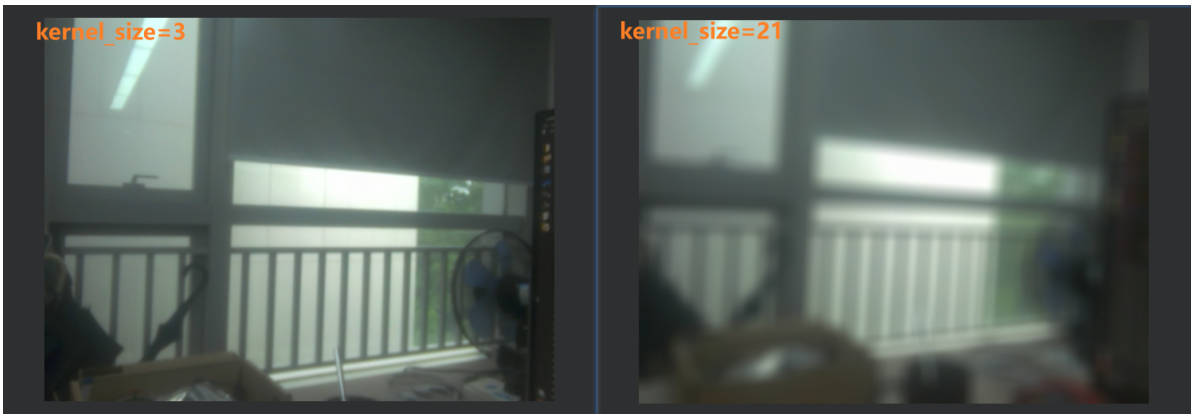
Mean Fuzzy

Example Results

Run this section's example code [Source Code/06.cv_lite/23.rgb888_mean_blur.py]

In this section, we will use the `cv_lite` extension module to implement mean blur filtering for RGB888 images on an embedded device.



Principle Explanation

What is Image Mean Blur Filtering?

Image mean blur filtering (mean blur) is a simple image smoothing technique used to reduce noise and blur details. It achieves this smoothing effect by averaging the values of each pixel in the image with its surrounding pixels. Mean blur is one of the most basic filtering methods in image processing and is suitable for fast denoising.

- **How it works**: Mean blur uses a convolution kernel (usually rectangular or square) to average the values of each pixel in the image and its surrounding pixels. All locations within the kernel have the same weight (i.e., uniform distribution). Unlike Gaussian filtering, mean blurring does not account for weight variations with distance from the center. As a result, each pixel is replaced with the average value of its neighborhood, smoothing the image.

- **Effect**: Mean blurring effectively removes high-frequency noise (such as random noise) from an image, but may cause loss of image detail (such as edges), resulting in a noticeable blurring effect. Compared to Gaussian filtering, mean blurring is more prone to introducing blocky artifacts because it does not account for differences in distance weights between pixels.
- **Application Scenarios**: Mean blurring is commonly used for image denoising, smoothing in preprocessing (such as reducing noise interference before edge detection), and scenarios requiring a fast blurring effect. Due to its simple computation, it is suitable for embedded devices with limited computing resources.

With mean blurring, the size of the convolution kernel (which must be an odd number) significantly affects the result: larger kernels result in a stronger blurring effect, but also increase computational effort and potentially lose more detail.

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 (includes mean_blur API)
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(width=1280, height=960)
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 and set the resolution to 1280x960 (the frame rate is not specified, so the default value is used).
- Resize the output frame to 640x480 and set it to RGB888 format (a three-channel color image, suitable for processing mean blur filters on color images).

Initialize the display module

```
Display.init(Display.ST7701, width=image_shape[1], height=image_shape[0],
to_ide=True, quality=100)
```

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=100` sets the image transmission quality to the highest.

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 mean blur parameters

```
kernel_size = 3
clock = time.clock() # Start FPS timer
```

- `kernel_size`: The convolution kernel size for the mean blur. Must be an odd number (such as 3, 5, 7, etc.). Larger values produce stronger blurring but increase computational complexity. In this code, it is set to 3, a smaller kernel suitable for light smoothing and denoising.
- `clock`: Used to calculate the frame rate.

Image processing and mean blur filter

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

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

    # Apply mean blur using cv_lite
    denoised_np = cv_lite.rgb888_mean_blur_fast(
        image_shape,
        img_np,
        kernel_size
    )

    # Wrap processed image for display
    img_out = image.Image(image_shape[1], image_shape[0], image.RGB888,
                          alloc=image.ALLOC_REF, data=denoised_np)

    # Display the blurred image
    Display.show_image(img_out)

    # Clear memory and print the frame rate / Collect garbage and show FPS
    gc.collect()
    print("mean blur fps:", clock.fps())
```

- **Image Capture**: Acquire an image frame using `sensor.snapshot()` and convert it to a NumPy array reference using `to_numpy_ref()`.
- **Mean Blur Processing**: Call `cv_lite.rgb888_mean_blur_fast()` to perform a mean blur filter, returning a NumPy array of the processed image.

- **Image Packaging and Display**: Pack the processed result into an RGB888 format image object and display it on the screen or in an IDE virtual window.
- **Memory Management and Frame Rate Output**: Call `gc.collect()` to clear memory and print the current frame rate using `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`: The convolution kernel size for mean blur. A larger value results in a stronger blur effect, suitable for processing higher levels of noise or requiring smoother images. An odd number (such as 3, 5, 7, etc.) must be used. It is recommended to start with 3 and gradually increase to larger values such as 7 or 9 to observe the effect and performance impact. Larger kernels significantly increase the amount of computation and may reduce frame rate.