Aim: To perform Handling Files, Cameras and GUIs

Objective: To perform Basic I/O Scripts, Reading/Writing an Image File, Converting Between an Image and raw bytes, Accessing image data with numpy.array,Reading /writing a video file, Capturing camera, Displaying images in a window ,Displaying camera frames in a window

Theory:

Basic I/O script :

Most CV applications need to get images as input. Most also produce images as output. An interactive CV application might require a camera as an input source and a window as an output destination. However, other possible sources and destinations include image files, video files, and raw bytes. For example, raw bytes might be transmitted via a network connection, or they might be generated by an algorithm if we incorporate procedural graphics into our application. Let's look at each of these possibilities.each pixel is represented by a single 8-bit integer, which means that the values for each pixel are in the 0-255 range, where 0 is black, 255 is white, and the in-between values are shades of gray. This is a grayscale image.each pixel is now represented by a three-element array, with each integer representing one of the three color channels: B, G, and R, respectively. Other common color models, such as HSV, will be represented in the same way, albeit with different value ranges. For example, the hue value of the HSV color model has a range of 0-180.

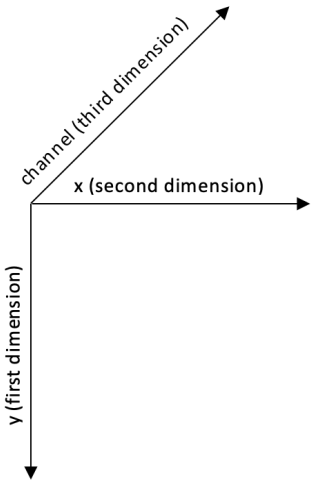
Reading/Writing an Image File:

OpenCV provides the imread function to load an image from a file and the imwrite function to write an image to a file. These functions support various file formats for still images (not videos). The supported formats vary as formats can be added or removed in a custom build of OpenCV—but normally BMP, PNG, JPEG, and TIFF are among the supported formats.The arguments to the constructor of the VideoWriter class deserve special attention. One of the most basic operations in OpenCV is displaying an image in a window. This can be done with the imshow function. If you come from any other GUI framework background, you might think it sufficient to call imshow to display an image. However, in OpenCV, the window is drawn (or re-drawn) only when you call another function, waitKey. The latter function pumps the window's event queue (allowing various events such as drawing to be handled), and it returns the keycode of any key that the user may have typed within a specified timeout. To some extent, this rudimentary design simplifies the task of developing demos that use video or webcam input; at least the developer has manual control over the capture and display of new frames.

Converting Between an Image and raw bytes:

A byte is an integer ranging from 0 to 255. Throughout real-time graphic applications today, a pixel is typically represented by one byte per channel, though other representations are also possible.

An OpenCV image is a 2D or 3D array of the numpy.array type. An 8-bit grayscale image is a 2D array containing byte values. A 24-bit BGR image is a 3D array, which also contains byte values. We may access these values by using an expression such as image[0, 0] or image[0, 0, 0]. The first index is the pixel's y coordinate or row, 0 being the top. The second index is the pixel's x coordinate or column, 0 being the leftmost. The third index (if applicable) represents a color channel. The array's three dimensions can be visualized in the following Cartesian coordinate system:



For example, in an 8-bit grayscale image with a white pixel in the upper-left corner, image[0, 0] is 255. For a 24-bit (8-bit-per-channel) BGR image with a blue pixel in the upper-left corner, image[0, 0] is [255, 0, 0].

Accessing image data with numpy. Array:

The numpy.array class is greatly optimized for array operations, and it allows certain kinds of bulk manipulations that are not available in a plain Python list. These kinds of numpy.array

type-specific operations come in handy for image manipulations in OpenCV. However, let's explore image manipulations step by step, starting with a basic example. Say you want to manipulate a pixel at coordinates (0, 0) in a BGR image and turn it into a white pixel

Reading/Writing a video file :

OpenCV provides the VideoCapture and VideoWriter classes, which support various video file formats. The supported formats vary depending on the operating system and the build configuration of OpenCV, but normally it is safe to assume that the AVI format is supported. Via its read method, a VideoCapture object may be polled for new frames until it reaches the end of its video file. Each frame is an image in a BGR format.

Conversely, an image may be passed to the write method of the VideoWriter class, which appends the image to a file in VideoWriter. Let's look at an example that reads frames from one AVI file and writes them to another with a YUV encoding:

The arguments to the constructor of the VideoWriter class deserve special attention. A video's filename must be specified. Any preexisting file with this name is overwritten. A video codec must also be specified. The available codecs may vary from system to system.

Capturing camera frames:

A stream of camera frames is represented by a VideoCapture object too. However, for a camera, we construct a VideoCapture object by passing the camera's device index instead of a video's filename.

Displaying images in a window :

One of the most basic operations in OpenCV is displaying an image in a window. This can be done with the imshow function. If you come from any other GUI framework background, you might think it sufficient to call imshow to display an image. However, in OpenCV, the window is drawn (or re-drawn) only when you call another function, waitKey. The latter function pumps the window's event queue (allowing various events such as drawing to be handled), and it returns the keycode of any key that the user may have typed within a specified timeout. To some extent, this rudimentary design simplifies the task of developing demos that use video or webcam input, at least the developer has manual control over the capture and display of new frames.

Displaying camera frames in a window :

OpenCV allows named windows to be created, redrawn, and destroyed using the namedWindow, imshow, and destroyWindow functions. Also, any window may capture keyboard input via the waitKey function and mouse input via the setMouseCallback function.

The argument for waitKey is a number of milliseconds to wait for keyboard input. By default, it is 0, which is a special value meaning infinity. The return value is either -1 (meaning that no key has been pressed) or an ASCII keycode, such as 27 for Esc. For a list of ASCII keycodes, Also, Python provides a standard function, ord, which can convert a character into its ASCII keycode. For example, ord('a') returns 97.

OpenCV's window functions and waitKey are interdependent. OpenCV windows are only updated when waitKey is called. Conversely, waitKey only captures input when an OpenCV window has focus.

Conclusion:

In this experiment we have covered the fundamental concepts and operations involved in handling files, cameras, and graphical user interfaces (GUIs) for computer vision applications using OpenCV. how to efficiently work with different image and video sources, process them, and display the results in windows. importance of I/O scripts for interacting with images and videos, and covered related different functions.The conversion between images and raw bytes explored