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.

Reading an Image File

For reading an image, use the imread() function in OpenCV.
Here's the syntax:

imread(filename, flags)

It takes two arguments:

1. The first argument is the image name, which requires a fully qualified pathname to the file.
2. The second argument is an optional flag that lets you specify how the image should be represented. OpenCV offers several options for this flag, but those that are most common include:

- cv2.IMREAD_UNCHANGED or -1
- cv2.IMREAD_GRAYSCALE or 0
- cv2.IMREAD_COLOR or 1

The default value for flags is 1, which will read in the image as a Colored image. When you want to read in an image in a particular format, just specify the appropriate flag.

Writing an Image File

Let's discuss how to write/save an image into the file directory, using the imwrite()

function. Check out its syntax:

imwrite(filename, image).

It takes two Arguments:

1. The first argument is the filename, which must include the filename extension (for example .png, .jpg etc). OpenCV uses this filename extension to specify the format of the file.
2. The second argument is the image you want to save. The function returns True if the image is saved successfully.

Take a look at the code below. See how simple it is to write images to disk. Just specify the filename with its proper extension (with any desired path prepended). Include the variable name that contains the image data, and you're done

Converting Between an Image and raw bytes

Conceptually, 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.

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

Reading a video file

In OpenCV, a video can be read either by using the feed from a camera connected to a computer or by reading a video file. The first step towards reading a video file is to create a Video Capture object. Its argument can be either the device index or the name of the video file to be read.

In most cases, only one camera is connected to the system. So, all we do is pass '0' and OpenCV uses the only camera attached to the computer. When more than one camera is connected to the computer, we can select the second camera by passing '1', the third camera by passing '2' and so on.

Writing a video file

Let's now take a look at how to write videos. Just like video reading, we can write videos originating from any source (a video file, an image sequence, or a webcam).
To write a video file:

- Retrieve the image frame height and width, using the get() method.
- Initialize a video capture object (as discussed in the previous sections), to read the video stream into memory, using any of the sources previously described.
- Create a video writer object.
- Use the video writer object to save the video stream to disk.

Continuing with our running example, let's start by using the get() method to obtain the video frame width and height.

Capturing camera frames

A stream of camera frames is represented by a Video Capture object too. However, for a camera, we construct a Video Capture 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 named Window, im show, and destroy Window functions. Also, any window may capture keyboard input via the wait Key function and mouse input via the set Mouse Callback function. The argument for wait Key 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 wait Key are interdependent. OpenCV windows are only updated when wait Key is called. Conversely, wait Key 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. We have finally, grasped how a computer vision application can be developed openCV consists of mat class which is used for pixel manipulation and it has a class relationship to application buffered image class.