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 -

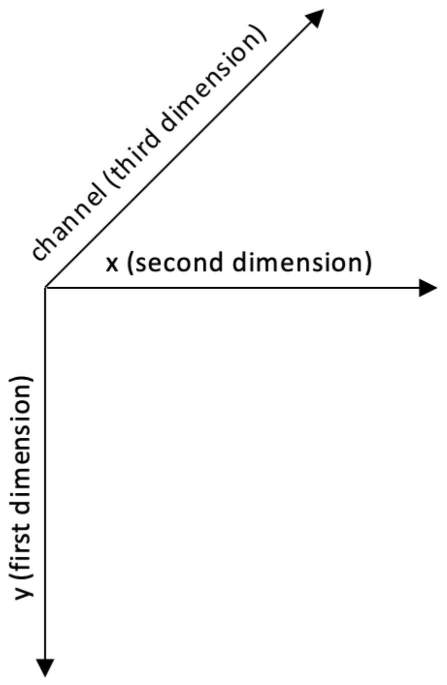
Images are typically required as an input for CV applications. Most also produce images as output. A window may serve as the output destination and the camera as the input source for an interactive CV application. Image files, video files, and raw bytes are among more sources and destinations that are possible. For instance, if our program uses procedural graphics, raw bytes may be created by an algorithm and communicated over a network connection.

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.

Converting Between an Image and raw bytes -

A byte is an integer with a value between 0 and 255. A pixel is commonly represented by one byte per channel in real-time graphic applications nowadays, however alternate formats are also feasible. 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].

We can use imencode() which will encode the Numpy ndarray in the specified format. This method will return two values, the first is whether the operation is successful, and the second is the encoded image in a one-dimension Numpy array. Then we can convert the returned array to real bytes either with the tobytes() method or io.BytesIO().

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. Suppose we want to change the blue value of a particular pixel, say, the pixel at coordinates, (150, 120). The numpy.array type provides a handy method, item, which takes three parameters: the *x* (or left) position, the *y* (or top) position, and the index within the array at the (*x*, *y*) position (In a BGR image, the data at a certain position is a three-element array containing the B, G, and R values in this order) and returns the value at the index position. Another method, itemset, sets the value of a particular channel of a particular pixel to a specified value. itemset takes two arguments: a three-element tuple (*x*, *y*, and index) and the new value. When we need to manipulate an entire image or a large region of interest, it is advisable that we utilize either OpenCV's functions or NumPy's array slicing.

Reading/Writing a video file -

The VideoCapture and VideoWriter classes in OpenCV handle a number of different 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. 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 VideoCapture object also represents a stream of camera frames. However, for a camera, we construct a VideoCapture object by passing the camera's device index instead of a video's filename. Unfortunately, in most cases, the get method of VideoCapture does not return an accurate value for the camera's frame rate; it typically returns 0. "Value 0 is returned when querying a property that is not supported by the backend used by the VideoCapture instance."

To create an appropriate VideoWriter class for the camera, we have to either make an assumption about the frame rate (as we did in the preceding code) or measure it using a timer.

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 we come from any other GUI framework background, we might think it sufficient to call imshow to display an image. However, in OpenCV, the window is drawn (or re-drawn) only when we 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. 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.

Conclusion:

In this, we have explored fundamental operations in OpenCV, which is a versatile library for computer vision and image processing in Python. These operations include reading/writing image files, converting between images and raw bytes, accessing image data using NumPy arrays, working with video files, capturing frames from a camera, and displaying images and camera frames in windows. With these capabilities, OpenCV provides a solid foundation for various computer vision applications and allows developers to efficiently manipulate and analyze visual data.