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.

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.

An image is a multidimensional array; it has columns and rows of pixels, and each pixel has a value. For different kinds of image data, the pixel value may be formatted in different ways. By default, imread returns an image in the BGR color format even if the file uses a grayscale format. BGR represents the same color model as red-green-blue (RGB), but the byte order is reversed.

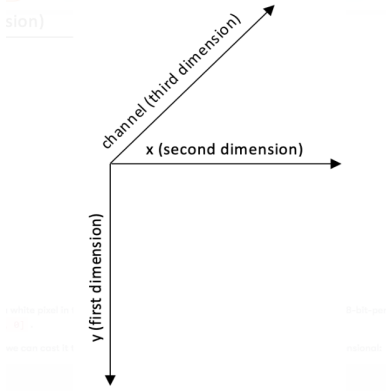
The imwrite() function requires an image to be in the BGR or grayscale format with a certain number of bits per channel that the output format can support. For example, the BMP file format requires 8 bits per channel, while PNG allows either 8 or 16 bits per channel.

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. 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 easiest (and most common) way to load an image in OpenCV is to use the imread function. We also know that this will return an image, which is really an array (either a 2D or 3D one, depending on the parameters you passed to imread).

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. 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 (remember that 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. There are several interesting things we can do by accessing raw pixels with NumPy's array slicing; one of them is defining regions of interests (ROI). Once the region is defined, we can perform a number of operations. Finally, we can access the properties of numpy.array.

These three properties are defined as follows:

shape: This is a tuple describing the shape of the array. For an image, it contains (in order) the height, width, and—if the image is in color—the number of channels. The length of the shape

tuple is a useful way to determine whether an image is grayscale or color. For a grayscale image, we have len(shape) == 2, and for a color image, len(shape) == 3.
size: This is the number of elements in the array. In the case of a grayscale image, this is the same as the number of pixels. In the case of a BGR image, it is three times the number of pixels because each pixel is represented by three elements (B, G, and R).
dtype: This is the datatype of the array's elements. For an 8-bit-per-channel image, the datatype is numpy.uint8.

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. 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. To create an appropriate VideoWriter class for the camera, we have to either make an assumption about the frame rate or measure it using a timer.
The number of cameras and their order is, of course, system-dependent. Unfortunately, OpenCV does not provide any means of querying the number of cameras or their properties. If an invalid index is used to construct a VideoCapture class, the VideoCapture class will not yield any frames; its read method will return (False, None). To avoid trying to retrieve frames from a VideoCapture object that was not opened correctly, you may want to first call the VideoCapture.isOpened method, which returns a Boolean.
The read method is inappropriate when we need to synchronize either a set of cameras or a multihead camera such as a stereo camera. Then, we use the grab and retrieve methods instead.

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.

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.

The mouse callback passed to setMouseCallback should take five arguments, as seen in our code sample. The callback's param argument is set as an optional third argument to setMouseCallback. By default, it is 0.

Conclusion:

OpenCV (Open Source Computer Vision) is a computer vision library that contains various functions to perform operations on Images or videos. It is a Python library that allows you to perform image processing and computer vision tasks. It provides a wide range of features, including object detection, face recognition, and tracking. OpenCV is a vast library that helps in providing various functions for image and video operations. With OpenCV, we can capture a video from the camera. It lets us create a video capture object which is helpful to capture videos through a webcam and then we may perform desired operations on that video.