Experiment No. 1
To perform Handling Files, Cameras and GUIs
Name of Student :- 09_Amruta Poojary
Date of Performance: 19/7/2023
Date of Submission: 26/7/2023

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.

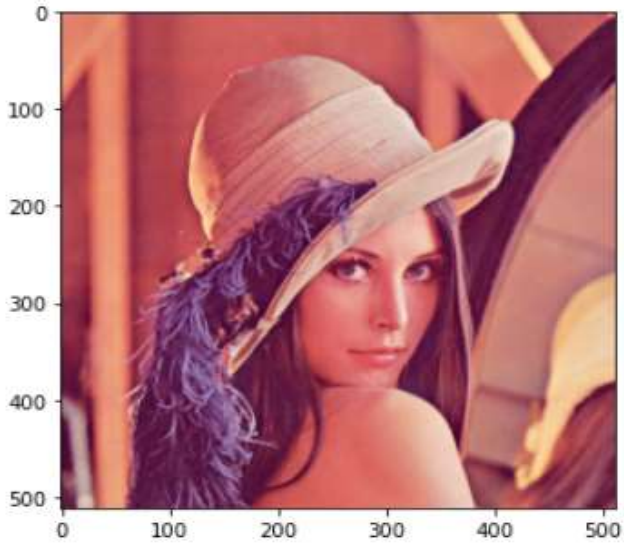
Reading/Writing an Image File

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

```
import matplotlib.pyplot as plt
import cv2
my_image = "lenna.png"
image = cv2.imread(my_image)
new_image=cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(5,5))
plt.imshow(new_image)
plt.show()
```




```
cv2.imwrite("lenna.jpg", image)
```

```
True
```

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.

```
import cv2
import numpy
import os

# Make an array of 120,000 random bytes.
randomByteArray = bytearray(os.urandom(120000))
flatNumpyArray = numpy.array(randomByteArray)

# Convert the array to make a 400x300 grayscale image.
grayImage = flatNumpyArray.reshape(300, 400)
cv2.imwrite('RandomGray.png', grayImage)

# Convert the array to make a 400x100 color image.
bgrImage = flatNumpyArray.reshape(100, 400, 3)
cv2.imwrite('RandomColor.png', bgrImage)
```

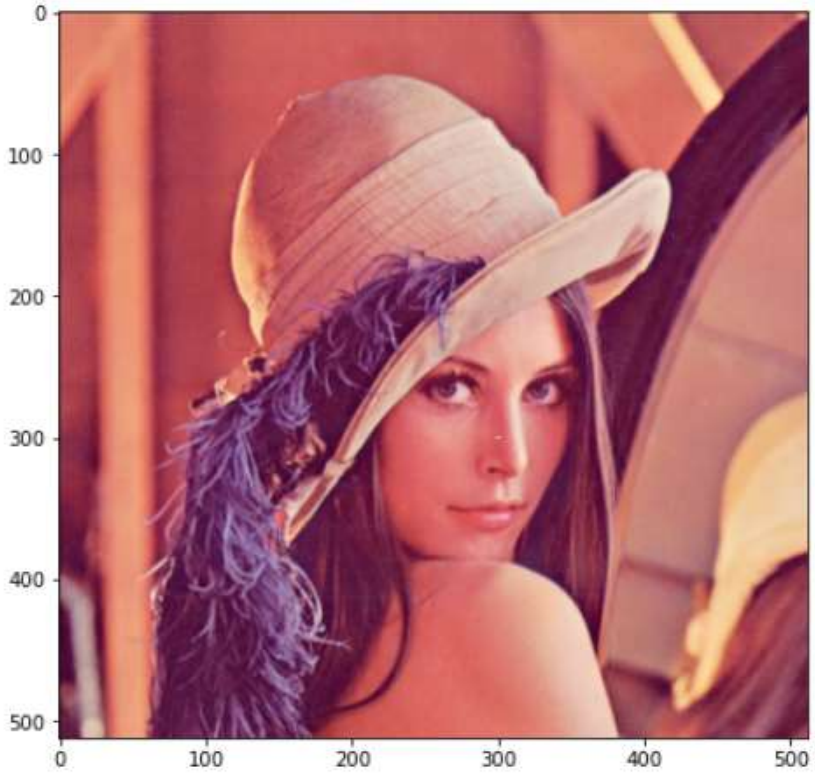
```
True
```

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 specific coordinates in a BGR image and turn it into a white pixel:

```
img = cv2.imread('lenna.png')
#img[0, 0] = [255, 255, 255]
img[300, 300] = [255, 255, 255]
img[300, 301] = [255, 255, 255]
img[300, 299] = [255, 255, 255]
new_image=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.figure(figsize=(7,7))
plt.imshow(new_image)
plt.show()
```



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:

```
import cv2
videoCapture = cv2.VideoCapture('MyInputVid.avi')
fps = videoCapture.get(cv2.CAP_PROP_FPS)
size = (int(videoCapture.get(cv2.CAP_PROP_FRAME_WIDTH)),
        int(videoCapture.get(cv2.CAP_PROP_FRAME_HEIGHT)))

videoWriter = cv2.VideoWriter(
    'MyOutputVid.avi',
    cv2.VideoWriter_fourcc('I','4','2','0'),
    fps,
    size)
success, frame = videoCapture.read()

while success: # Loop until there are no more frames.
    videoWriter.write(frame)
    success, frame = videoCapture.read()
```

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. 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.

```
import cv2
import numpy as np
img = cv2.imread('lenna.png')
new_image=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

cv2.imshow("lenna", new_image)
cv2.waitKey()
cv2.destroyAllWindows()
```

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.

```
import cv2
clicked = False
def onMouse(event, x, y, flags, param):
    global clicked
    if event == cv2.EVENT_LBUTTONUP:
        clicked = True

cameraCapture = cv2.VideoCapture(0)
cv2.namedWindow('MyWindow')
cv2.setMouseCallback('MyWindow', onMouse)

print('Showing camera feed. Click window or press any key to stop.')
success, frame = cameraCapture.read()
while success and cv2.waitKey(1) == -1 and not clicked:
    cv2.imshow('MyWindow', frame)
    success, frame = cameraCapture.read()

cv2.destroyWindow('MyWindow')
cameraCapture.release()
```

```
Showing camera feed. Click window or press any key to stop.
```

Conclusion:

OpenCV stands for Open-Source Computer Vision. It is a free computer vision library that allows to manipulate images and videos to accomplish a variety of tasks, from displaying frames from a webcam to teaching a robot to recognize real-life objects. OpenCV has good I/O capabilities which helps us to read and write image and video files, manipulate image data in Numpy arrays as well as displaying images or camera frames in a window.