OpenCV Installation in Jupyter Notebook

In [2]:

```
!pip install opencv-contrib-python
```

```
Collecting opencv-contrib-python
  Downloading opencv_contrib_python-4.8.0.76-cp37-abi3-macosx_10_16_x86_6
4.whl (64.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 64.3/64.3 MB 804.0 kB/s eta
0:00:0000:0100:03
Requirement already satisfied: numpy>=1.19.3 in /Users/jvtaylar/opt/anaco
nda3/lib/python3.9/site-packages (from opencv-contrib-python) (1.21.5)
Installing collected packages: opencv-contrib-python
Successfully installed opencv-contrib-python-4.8.0.76
```

In [9]:

```
!pip install opencv-python
```

```
Requirement already satisfied: opencv-python in /Users/jvtaylar/opt/anaco
nda3/lib/python3.9/site-packages (4.7.0.72)
Requirement already satisfied: numpy>=1.17.0 in /Users/jvtaylar/opt/anaco
nda3/lib/python3.9/site-packages (from opencv-python) (1.21.5)
```

In [1]:

```
import cv2

print(cv2.__version__)
```

```
4.8.0
```

Handling Files, Cameras, and GUIs

Specifically, in this chapter, our code samples and discussions will cover the following tasks:

- Reading images from image files, video files, camera devices, or raw bytes of data in memory
- Writing images to image files or video files
- Manipulating image data in NumPy arrays
- Displaying images in windows

Basic I/O scripts

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.

Anatomy of the Representation of Images in OpenCV and Numpy

An image is a multidimensional array; it has columns and rows of pixels, and each pixel has a value.

In [2]:
```
import numpy
img = numpy.zeros((3, 3), dtype=numpy.uint8)
img

```

Out[2]:
```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]], dtype=uint8)
```

Here, 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.

Let's now convert this image into blue-green-red (BGR) format using the `cv2.cvtColor` function:

In [3]:
```
img = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
img

```

Out[3]:
```
array([[[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]],

       [[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]],

       [[0, 0, 0],
        [0, 0, 0],
        [0, 0, 0]]], dtype=uint8)
```

As you can see, 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.

You can check the structure of an image by inspecting the shape property, which returns rows, columns, and the number of channels (if there is more than one).

Consider this example:

In [5]:
```python
img = numpy.zeros((5, 3), dtype=numpy.uint8)
print(img.shape)

```

```
(5, 3)
```

The preceding code will print (5, 3); in other words, we have a grayscale image with 5 rows and 3 columns. If you then converted the image into BGR, the shape would be (5, 3, 3), which indicates the presence of three channels per pixel.

Loading an Image

Images can be loaded from one file format and saved to another. For example, let's convert an image from PNG into JPEG:

In [8]:
```python
import cv2
image = cv2.imread('mypic.png')
cv2.imwrite('mypic.jpg', image)

```

Out[8]:
```
True
```

OpenCV's Python module is called `cv2` even though we are using `OpenCV 4.x` and not `OpenCV 2.x.` Historically, OpenCV had two Python modules: `cv2` and `cv`. The latter wrapped a legacy version of OpenCV implemented in C. Nowadays, OpenCV has only the `cv2 Python module`, which wraps the current version of OpenCV implemented in C++.

As an example, let's load a PNG file as a grayscale image (losing any color information in the process), and then save it as a grayscale PNG image:

In [7]:
```python
import cv2
grayImage = cv2.imread('mypic.png', cv2.IMREAD_GRAYSCALE)
cv2.imwrite('mypicGray.png', grayImage)

```

Out[7]:
```
True
```

The path of an image, unless absolute, is relative to the working directory (the path from which the Python script is run), so, in the preceding example, `mypic.png` would have to be in the working directory or the image would not be found. If you prefer to avoid assumptions about the working directory, you can use absolute paths, such as `C:\Users\Joe\Pictures\mypic.png` on Windows, `/Users/Joe/Pictures/mypic.png` on Mac, or `/home/joe/pictures/mypic.png` on Linux.

We can access the properties of numpy.array, as shown in the following code:

In [10]:
```python
import cv2
img = cv2.imread('mypic.png')
print(img.shape)
print(img.size)
print(img.dtype)
```

```
(828, 902, 3)
2240568
uint8
```

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.

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. Let's consider the following example, which captures 10 seconds of video from a camera and writes it to an AVI file. The code is similar to the previous section's sample (which was captured from a video file instead of a camera) but changes are marked in bold:

In [19]:
```python
import cv2
cameraCapture = cv2.VideoCapture(0)
fps = 30 # An assumption
size = (int(cameraCapture.get(cv2.CAP_PROP_FRAME_WIDTH)),
 int(cameraCapture.get(cv2.CAP_PROP_FRAME_HEIGHT)))
videoWriter = cv2.VideoWriter(
 'MyOutputVid.avi', cv2.VideoWriter_fourcc('I','4','2','0'),
 fps, size)
success, frame = cameraCapture.read()
numFramesRemaining = 10 * fps - 1 # 10 seconds of frames
while success and numFramesRemaining > 0:
 videoWriter.write(frame)
 success, frame = cameraCapture.read()
 numFramesRemaining -= 1

```

Displaying an image 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.

Here is a very simple sample script to read an image from a file and display it:

In [ ]:
```python
import cv2
import numpy as np
img = cv2.imread('mypic.png')
cv2.imshow('my image', img)
cv2.waitKey()
cv2.destroyAllWindows()
```

The `imshow` function takes two parameters:

- the name of the window in which we want to display the image and the image itself.
- The aptly named `destroyAllWindows` function disposes of all of the windows created by OpenCV

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. Let's look at an example where we show the frames captured from a live camera:

In [1]:
```python
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.
```

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, refer to http://www.asciitable.com/ (http://www.asciitable.com/). Also, note that Python provides a standard function, ord, which can convert a character into its ASCII keycode. For example, ord('a') returns 97.

Again, note that 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.