Exercice 01: Basic Image Operations

1. **Read the Image and Display its Size**

python

Copy code

```
image_path = '/mnt/data/rose_1024.tif'

image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

print("Image Size:", image.shape)
```

   - **Explanation**: cv2.imread reads the image in grayscale mode (cv2.IMREAD_GRAYSCALE). The shape attribute provides the image dimensions as (height, width).
   - **Benefit**: Knowing the size of the image helps set the parameters for other processing operations (like cropping and resizing). It confirms that the image has loaded correctly and gives information on the resolution.

2. **Display the Image and Access Pixel Values**

python

Copy code

```
pixel_values = {
   (0, 0): image[0, 0],
   (10, 10): image[10, 10],
   (15, 30): image[15, 30]
}

print("Pixel Values:", pixel_values)
```

   - **Explanation**: Accessing pixel values at specified coordinates gives insight into the pixel intensities. This operation retrieves the grayscale intensity for each specified pixel location, displaying it as a dictionary.
   - **Benefit**: By examining specific pixel values, you gain insight into the image's grayscale intensity distribution, which could be useful for understanding brightness, contrast, or patterns in particular areas of the image.

3. **Save the Image in a Different Format (e.g., PNG)**

python

Copy code

```
new_image_path = '/mnt/data/rose_1024_converted.png'

cv2.imwrite(new_image_path, image)
```

- **Explanation**: cv2.imwrite saves the image in a new format, in this case, PNG. The image variable, containing the grayscale image data, is saved with a different extension.
- **Benefit**: Saving in a different format allows compatibility with different applications or tools. PNG, for example, is widely supported and uses lossless compression, which maintains image quality.

Exercice 02: Simple Digital Image Processing Tasks

For this exercise, we perform specific transformations on the grayscale image, aiming to explore basic image manipulation operations.

1. **Read the Image in Grayscale**

python

Copy code

image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

- **Explanation**: This reads the image as a grayscale array of pixel intensities. Grayscale images are often used in image processing because they simplify computation by reducing color channels.
- **Benefit**: Grayscale images are computationally lighter and often easier to work with when performing processing tasks that do not require color.

2. **Flip the Image Vertically**

python

Copy code

flipped_image = cv2.flip(image, 0)

- **Explanation**: cv2.flip flips the image vertically when flipCode=0. This operation inverts the image along the horizontal axis.
- **Benefit**: Flipping vertically (or horizontally) is useful in applications like augmented reality or image alignment. It’s also a common data augmentation technique in machine learning to increase the diversity of training data.

3. **Crop a Specific Region of the Image**

python

Copy code

cropped_image = flipped_image[257:768, 257:768]

- **Explanation**: This crops the rectangle [257:768, 257:768] from the vertically flipped image. It captures a central region of the image.

- **Benefit**: Cropping allows you to focus on a specific area of interest. This can be useful for reducing the processing area to relevant portions or for tasks that require analyzing a subset of the image, like object recognition.

4. **Downsample the Image by Reducing Its Size by Half**

python

Copy code

```
resized_image = cv2.resize(flipped_image, (image.shape[1] // 2, image.shape[0] // 2))
```

- **Explanation**: cv2.resize resizes the image to half its original dimensions. This is achieved by dividing both the width and height by 2.
- **Benefit**: Downsampling reduces computational requirements for further processing tasks. It's also useful when working with limited resources or needing a quick overview without full-resolution details.

5. **Display the Intensity Profile of the 512th Horizontal Line**

python

Copy code

```
middle_line_profile = flipped_image[512, :]

plt.plot(middle_line_profile)
```

- **Explanation**: This retrieves intensity values along the 512th row of the vertically flipped image and plots it as a profile using matplotlib.
- **Benefit**: Intensity profiles provide a visualization of pixel intensities along a specific line in the image. This is valuable for identifying patterns, transitions, or edges along that line, and it's widely used in image analysis for edge detection or histogram profiling.

Summary of Useful OpenCV Functions

- **cv2.flip(image, flipCode)**:
  - **Parameters**:
    - image: The input image to be flipped.
    - flipCode: Determines the flipping direction:
      - 0 = Flip vertically (along the horizontal axis).
      - >0 = Flip horizontally (along the vertical axis).
      - <0 = Flip both vertically and horizontally.
  - **Usefulness**: It's beneficial for transformations that alter image orientation, augment datasets, or help align images in computer vision tasks.

Each operation here has specific benefits in image processing. They collectively provide a foundational set of tools for transforming and analyzing images, which are core to tasks in computer vision, data augmentation, and digital image analysis.