Computer Vision

2.- Imaging Tools

1 Imaging edition

In this topic, we will work with changes in image resolution, histograms, brightness/contrast, and color space conversion.

1.1 Spatial and Radiometric Resolution Changes

In computer vision, resizing an image refers to the process of changing the dimensions of the image. This typically involves adjusting the number of pixels in the image, either by increasing or decreasing its size. The main reasons for resizing images in computer vision include:

- Computational Efficiency: Smaller images require fewer computational resources for processing. Resizing can be crucial in real-time applications where processing speed is essential, such as in video analysis or on resource-constrained devices like mobile phones.
- Memory Constraints: Resizing helps in managing memory constraints, especially when working with large datasets. It allows for more efficient storage and retrieval of images.
- Standardization: Resizing is often done to standardize the input size for machine learning models. Many deep learning models, especially convolutional neural networks (CNNs), require fixed-size inputs. Resizing ensures that all input images have the same dimensions, facilitating model training and inference.
- Preprocessing for Models: Some machine learning models, particularly those used in computer vision tasks, may be sensitive to the input image size. Resizing helps preprocess images into a format compatible with the model's requirements.
- Data Augmentation: Resizing is a common component of data augmentation strategies. It allows for the generation of additional training samples by randomly cropping or resizing images, which can improve the generalization ability of the model

- Display and Visualization: Resizing may be necessary for displaying images on screens with specific dimensions or for visualization purposes. It helps maintain a consistent appearance when presenting images in a user interface.

When resizing an image, it's important to consider the aspect ratio to avoid distortion. The aspect ratio is the ratio of the width to the height of the image. Maintaining the original aspect ratio during resizing helps preserve the visual content of the image without distorting its proportions.

In OpenCV, we can change spatial resolution (image size) using the 'resize' function:

```
# Resize the image 'img' to 75% of its original size
dst = cv.resize(img, (0, 0), fx=0.75, fy=0.75)

# Resize the image to a specific size (640 x 480)
dst = cv.resize(img, (640, 480))
```

To change only the data type of the image (related to radiometric resolution or depth), we can perform a conversion with NumPy:

```
converted = np.float32(img)  # Convert to float and save in 'dst'
```

1.2 Brightness/Contrast

Adjusting brightness and contrast are common image processing techniques used in computer vision and image analysis.

Changing the brightness of an image involves making it lighter or darker. This adjustment is useful for improving visibility, enhancing details, or correcting underexposed or overexposed images. This technique adjustment is often achieved by adding or subtracting a constant value to each pixel intensity in the image. Increasing the value makes the image brighter, while decreasing it makes the image darker. It uses including image enhancement, object recognition, and video processing.

Contrast refers to the difference in intensity between the brightest and darkest parts of an image. Adjusting contrast helps in making objects and details more distinguishable. This adjustment is typically done by stretching or compressing the range of pixel intensities. This can be achieved through techniques like histogram equalization, where the distribution of pixel values is modified. It is essential for improving the visual quality of images, especially in scenarios where there is poor lighting or when certain features need to be emphasized.

Both brightness and contrast adjustments are part of the broader field of image enhancement, where the goal is to improve the visual quality of images for better analysis and interpretation by computer vision algorithms. The specific techniques used may vary based on the requirements of the application and

the characteristics of the images being processed.

Brightness and contrast of an image are defined by the following equations:
Brightness: $b = \frac{1}{N} \sum \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} I(x, y)$
Contrast: $c = \sqrt{\frac{1}{N} \sum \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} (I(x, y) - b)^2}$

Moreover, we have the possibility of joint brightness and contrast adjustment simultaneously, being beneficial to achieve the desired visual effect. This can be done by applying a linear transformation to the pixel values, which combines both brightness and contrast adjustments. The transformation is often expressed as $I_{new} = \alpha \cdot I_{old} + \beta$, where I_{old} is the original pixel intensity, α controls contrast, and β controls brightness. This double adjustment is useful in scenarios where both global and local image features need enhancement.

1.3 Histograms

The histogram of an image is a graphical representation of the distribution of pixel intensities. It shows how many pixels in the image have a particular intensity value. The x-axis represents the intensity values, and the y-axis represents the frequency (or number of pixels) at each intensity level.

There are different histograms in image processing:

- Intensity Histogram: provides insight into the overall brightness and contrast characteristics of an image. It can be used to identify the dominant intensity levels, the presence of peaks or valleys, and the overall distribution of pixel intensities. To create it, the intensities of all pixels in the image are counted and grouped into bins based on their intensity values. The resulting histogram can be visualized as a bar graph. It is useful for image preprocessing tasks, such as adjusting brightness and contrast, thresholding, and image equalization.

- RGB Histogram (for Color Images): in color images, separate histograms can be generated for each color channel (Red, Green, and Blue). Analyzing these histograms helps understand the color distribution in an image. Similar to intensity histograms, RGB histograms are created by counting the occurrences of intensity values for each color channel. These histograms are valuable for color correction, white balance adjustment, and understanding the color composition of an image.

- Histogram Equalization: is a technique used to enhance the contrast of an image by redistributing pixel intensities. It aims to make the histogram as uniform as possible. During histogram equalization, the cumulative distribution function (CDF) of the histogram is modified to achieve a more uniform distribution, resulting in improved contrast. This histogram

is often applied in image processing to enhance the visibility of features in images with poor contrast.

- Histogram Matching: is a method for transforming the intensity distribution of an image to match a specified target histogram. This can be useful for aligning the appearance of different images or achieving a desired look. The transformation function is adjusted to map the intensity values of the input image to those of the target histogram. This method is employed in tasks like image registration, where aligning the intensity distributions of images from different sources is crucial.

The histogram analysis is a fundamental tool in image processing, providing valuable insights and serving as a basis for various image enhancement and manipulation techniques.

In OpenCV, we can calculate the histogram of an image using the 'calcHist' function. The following code calculates the histogram of a grayscale image, displays it in the terminal, and creates a graph in a window:

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt


# Load the image
img = cv.imread('image.jpg', cv.IMREAD_GRAYSCALE)

if img is None:
    print("Error reading the image")
else:
    # Calculate the histogram
    hist = cv.calcHist([img], [0], None, [256], [0, 256])

    # Show the image and the histogram
    plt.figure(figsize=(12, 6))

    plt.subplot(121), plt.imshow(img, 'gray')
    plt.title('Image'), plt.xticks([]), plt.yticks([])

    plt.subplot(122), plt.plot(hist, color='blue')
    plt.title('Histogram'), plt.xlabel('Pixel Value'), plt.ylabel('
                                    Frequency')

    plt.tight_layout()
    plt.show()
```

1.4 Color Space Conversion

Color space conversion is a fundamental operation in computer vision and image processing. It involves transforming an image from one color representation to another. Different color spaces are used for various purposes, and each has its advantages and is suitable for specific applications. For instance, OpenCV

supports more than 150 color spaces.

Some common color spaces and their typical applications are:

- RGB (Red, Green, Blue): used in displays and cameras. Representation: Each pixel is represented by its red, green, and blue intensity values.
- Grayscale: Simplifies image processing, especially when color information is not crucial. Representation: Single intensity value per pixel, usually representing luminance.
- HSV (Hue, Saturation, Value): Used for color manipulation and segmentation tasks. Representation: Describes colors in terms of their hue (color), saturation (vividness), and value (brightness).
- LAB: Useful for color-based image segmentation, color correction, and color space transformations. Representation: Separates image information into three channels: L* (luminance), a* (green to red), and b* (blue to yellow).
- YUV: Common in video compression and transmission. Separates image information into a luminance (Y) channel and two chrominance (U and V) channels.
- CMY and CMYK (Cyan, Magenta, Yellow, Key/Black): Used in color printing. Representation: Describes colors based on subtractive color mixing.

In OpenCV, the 'cvtColor' function performs conversions and accepts up to four parameters:

1. src: Input image.
2. code: Conversion code for the color space.
3. dst (optional): Output image with the same size and resolution as the input.
4. dstCn (optional): The number of channels in the destination image.

Example:

```
grayImg = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
```

1.5 Pseudocolor

Pseudocoloring in computer vision refers to the process of assigning artificial colors to grayscale images to enhance visual interpretation. This technique is often used to represent additional information in an image or highlight specific features. Pseudocoloring is different from true color images, where each pixel directly corresponds to a color in the RGB (Red, Green, Blue) color space.
The choice of colormap depends on the specific characteristics you want to emphasize.

Keep in mind that for true quantitative analysis, it's essential to understand the mapping between the pseudocolors and the underlying data. Colormaps are often chosen based on the nature of the data being visualized and the desired emphasis on certain features (e.g., temperature variations, elevation levels, or other scalar values).

In OpenCV, using the 'applyColorMap' function, we can pseudocolor grayscale images using predefined color maps in OpenCV:

```
img = cv.imread('image.jpg', cv.IMREAD_GRAYSCALE)
imgray = cv.applyColorMap(img, cv.COLORMAP_JET)
```