Image Processing in Python

Image processing is the field of study and application that deals with modifying and analyzing digital images using computer algorithms. The goal of image processing is to enhance the visual quality of images, extract useful information, and make images suitable for further analysis or interpretation.

Image Processing Using OpenCV

OpenCV (Open Source Computer Vision) is a widely-used, powerful library for image processing and computer vision. It offers a robust set of functions and tools to help build applications that work with images and videos.

While capturing photos is easy, enhancing and processing them often requires more complex code. This is where libraries like OpenCV become essential. As a popular open-source package, OpenCV covers a broad spectrum of image processing and computer vision techniques. It supports several programming languages, including Python, C++, and Java, and is highly optimized for real-time applications with a wide range of features.

How to install OpenCV

To install OpenCV for Python, you can use the following command:

```
!pip3 install opencv-python
```

Once installed, you can verify it by importing OpenCV in Python:

```
import cv2
print(cv2.__version__)
```

Lab task 1

Gray-scaling

A colored image consists of 3 color channels where a gray image only consists of 1 Color channel which carries intensity information for each pixel showing the image as black-and-white.

In this task you have to convert the image into Gray Scaling and separate the 3 color from image i.e **red** **green** **and** **blue**

Implement color channel separation and visualization with the following steps:

1. **Split the Image into Color Channels:** Use OpenCV to split the image into its red, green, and blue channels.
2. **Create a Plot with Three Subplots:** Set up a matplotlib figure with three subplots arranged horizontally.
3. **Display Each Color Channel:** Convert each color channel to RGB format and display it in its respective subplot.
4. **Add Titles to Each Plot:** Label each subplot with the corresponding color channel (Red, Green, Blue).

Image Translation

Image Translation refers to shifting an image along the x and y axes. This operation can be useful for tasks like object detection, data augmentation, or repositioning image elements.

In OpenCV, you can translate an image using the `cv2.warpAffine()` function and a translation matrix.

Implement image translation using OpenCV with the following steps:

1. **Define the Translation Matrix:** Create a translation matrix to shift the image by 100 pixels along the x-axis and 50 pixels along the y-axis.
2. **Get Image Dimensions:** Extract the number of rows and columns from the image.

3. **Apply the Translation:** Use `cv2.warpAffine` to apply the translation matrix to the image.
4. **Convert the Image for Display:** Convert the translated image from BGR to RGB format.
5. **Display the Translated Image:** Use `matplotlib` to display the translated image without axis labels.

Image rotation

Image Rotation refers to rotating an image by a certain angle around a specific center point, often the image's center. OpenCV provides the `cv2.warpAffine()` function combined with a rotation matrix to achieve this.

Implement image rotation using OpenCV with the following requirements:

1. **Obtain the Image Dimensions:** Extract the height and width of the image.
2. **Define the Rotation Center:** Set the center of the image around which the rotation will occur.
3. Copy the image into 4 times and an rotate each each one **90** degree **180** degree **270** degree and **360** degree
4. **Specify the Rotation Angle:** Rotate the image by 90,180,270,360 degrees counter-clockwise. And anti clock wise
5. **Get the Rotation Matrix:** Use `cv2.getRotationMatrix2D` to compute the matrix needed for the rotation.
6. **Perform the Rotation:** Apply the rotation matrix to the image using `cv2.warpAffine`.
7. **Convert the Image for Display:** Convert the rotated image from BGR to RGB format.
8. **Display the Rotated Image:** Use `matplotlib` to display the rotated image without axis labels.

Scaling and Resizing

Scaling and Resizing are important operations in image processing. They involve changing the dimensions (size) of an image while maintaining its visual content.

OpenCV provides the `cv2.resize()` function to scale and resize images.

In this lab you have to perform image scaling and resizing using OpenCV and Matplotlib. Your task is to complete and execute the script to demonstrate different scaling techniques.

Instructions:

1. **Load an image** using OpenCV. Ensure the image is loaded correctly before proceeding with the resizing operations.
2. **Scale the image** to 15% of its original size using linear interpolation.
3. **Enlarge the image** to twice its original size using cubic interpolation.
4. **Resize the image** to specific dimensions of 200x400 pixels using area interpolation.
5. **Visualize** all three results side by side using Matplotlib, ensuring that each subplot has a title indicating the interpolation method used.

Image Blur

Implement a Gaussian blur on an image using OpenCV with the following steps:

1. **Read the Image:** Load the image using OpenCV.
2. **Apply Gaussian Blur:** Use OpenCV's `cv2.GaussianBlur` function to apply a Gaussian blur to the image. Set the kernel size to `(15, 15)` to achieve the desired level of blurring.
3. **Display the Blurred Image:** Convert the blurred image to RGB format and display it using matplotlib.

Lab task 2

Adjust Image Brightness and Contrast

Description: You are given an image and need to adjust its brightness and contrast. Implement the following steps:

1. **Read the image** from a file path.
2. **Convert the image** to `float32` format for precision in calculations.
3. **Define the contrast (alpha)** and **brightness (beta)** Set `alpha` (contrast control) to 1.5 and `beta` (brightness control) to 50..
4. **Apply the brightness and contrast adjustments** Use `cv2.convertScaleAbs` to adjust the brightness and contrast of the image.
5. **Convert the adjusted image** to RGB format for display.
6. **Display the adjusted image** using matplotlib to Show the image with adjusted brightness and contrast without axis

Crop an Image

Description: You are given an image, and your task is to crop a specific region of it and display the cropped portion. Follow the steps below:

1. **Read an image** into a variable using OpenCV.
2. **Crop a specific region** of the image from `(startY:startY+height, startX:startX+width)` where:
    - startY = 50, height = 200
    - startX = 50, width = 200
3. **Convert the cropped image** to RGB format for proper display using `matplotlib`.
4. **Display the cropped image** using `matplotlib` without showing the axis.

Draw a Circle on an Image

Description: You are provided with an image, and your task is to draw a circle on it and display the result. Follow the steps below:

1. **Get the dimensions** of the image to find the center.
2. **Draw a circle** at the center of the image with the following specifications:
    - **Radius:** 50 pixels
    - **Color:** Green `(0, 255, 0)`
    - **Thickness:** 5 pixels
3. **Convert the image** to RGB format for proper display using `matplotlib`.
4. **Display the image** with the circle using `matplotlib`, and hide the axis.

Apply Canny Edge Detection

Description: Given an image, your goal is to perform edge detection using the Canny algorithm. Follow these steps:

1. **Convert the image** to grayscale using OpenCV.
2. **Apply Canny edge detection** with threshold values 100 and 200.
3. **Display the resulting edges** using `matplotlib` with a grayscale colormap, and hide the axis.

Lab task 3

You are developing a real-world image processing application for a smart surveillance system. This system needs to process images captured by surveillance cameras to enhance visibility, detect objects, and present useful visual information. Your task is to implement a Python script that performs a series of image processing operations to prepare and analyze images effectively.

Lab task 4

You are given an image and need to analyze and visualize its histograms. Follow these steps:

1. **Convert the Image to Grayscale**:
    - Use OpenCV to convert the given image to a grayscale image.
2. **Compute the Histograms**:
    - Calculate the histogram for the grayscale image.
    - Calculate histograms for the Blue and Green color channels of the original image.
3. **Set Up the Visualization**:
    - Create a Matplotlib figure with a 2x3 grid of subplots.
    - Plot the grayscale histogram in the top-left subplot.
    - Plot the Blue and Green channel histograms in the top-middle and top-right subplots respectively.
    - Display the original image in the bottom-left subplot.
4. **Ensure the Layout**:
    - Ensure that the axis is turned off for all subplot areas except for the histograms.
    - Adjust the layout to make sure everything is displayed correctly.

Note: Be careful with the indexing to avoid errors and ensure that you only plot histograms for the channels that exist.

Lab task 5

Linear spatial filtering is a technique used in image processing to modify an image based on the linear combination of pixel values in its neighborhood. This process involves using a filter (or kernel) that slides over the image to perform operations such as smoothing, sharpening, and edge detection.

Key Concepts

1. **Filter (Kernel):** A small matrix (e.g., 3x3, 5x5) used to process the image. Each element in the kernel represents a weight that modifies the corresponding pixel in the image.
2. **Convolution:** The core operation in linear spatial filtering. It involves overlaying the filter on the image, multiplying the filter values with the corresponding pixel values, summing the results, and replacing the central pixel with this sum.
3. **Filtering Operations:**
    - **Smoothing (Blurring):** Reduces noise and detail by averaging pixel values, which results in a softening effect. Common kernels include the average filter and Gaussian filter.

- **Sharpening:** Enhances edges and fine details by emphasizing differences between pixel values. Common kernels include the Laplacian filter and the unsharp mask.
- **Edge Detection:** Highlights the boundaries of objects within an image. Filters like the Sobel and Prewitt operators are commonly used for this purpose.

You have been provided with a colored image. Your task is to apply linear spatial filtering techniques to process this image. Specifically, you need to perform the following steps:

1. **Convert the image to grayscale** to simplify the processing.
2. **Apply Gaussian blur** to the grayscale image. Use a kernel size of 5x5 for the blurring operation.
3. **Display both the original grayscale image and the blurred image** side by side for comparison.

Implement these steps using OpenCV and Matplotlib in Python. Ensure that the images are displayed in a side-by-side layout with appropriate titles