Image Processing: A Step-by-Step Documentation

Deep Amish Shah

Introduction

This document provides a professional and detailed explanation of an image processing pipeline implemented in Python. The tasks include loading and converting images, applying filters like Sobel and Gaussian, and performing Harris Corner Detection along with Non-Maximum Suppression.

1 Importing and Preparing the Image

Code

```python
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

img = Image.open("flower.jpg")
img = img.convert('RGB')
img_np = np.array(img)
```

Listing 1: Import Libraries and Load Image

Explanation

- The image is loaded using the `PIL` library and converted to RGB to ensure consistent color channel usage.
- It is then transformed into a NumPy array, allowing efficient numerical operations and matrix manipulations.
- These preprocessing steps are crucial because all further operations rely on structured pixel-wise data access.

2 Grayscale Conversion

```
gray_np = np.dot(img_np[...,:3], [0.299, 0.587, 0.114])
gray_np = gray_np.astype(np.float64)
```

Listing 2: Convert RGB to Grayscale

Explanation

- Converting an image to grayscale reduces its dimensionality from 3 channels to 1, simplifying subsequent analysis.
- The formula used is a weighted sum that reflects human brightness perception: green is weighted more than red, and blue the least.
- This transformation is important for operations like edge detection and corner detection that rely on intensity gradients.

Grayscale Image Output

Figure 1: Grayscale Converted Image

3 Custom Convolution Function

```
def my_convolve2d(image, kernel):
    kernel = np.flipud(np.fliplr(kernel))
    iH, iW = image.shape
    kH, kW = kernel.shape
    pad_h = kH // 2
    pad_w = kW // 2
    padded_image = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w)),
       mode='symmetric')
    output = np.zeros((iH, iW), dtype=np.float64)
    for i in range(iH):
        for j in range(iW):
            region = padded_image[i:i+kH, j:j+kW]
            output[i, j] = np.sum(region * kernel)
    return output
```

Listing 3: Custom 2D Convolution

Explanation

- Convolution is a fundamental operation in image processing used for filtering, blurring, sharpening, etc.
- This custom implementation performs 'same' convolution, maintaining the input image size.
- The kernel is flipped to perform true convolution (as opposed to cross-correlation).
- Symmetric padding minimizes edge distortion, and the nested loops apply the filter pixel by pixel.
- This flexibility allows convolution with arbitrary filters like Gaussian, Sobel, and more.

4 Gaussian Kernel and Filtering

```
def gaussian_kernel(kernel_size, sigma):
    ax = np.linspace(-(kernel_size - 1) / 2, (kernel_size - 1) / 2,
       kernel_size)
    xx, yy = np.meshgrid(ax, ax)
    kernel = np.exp(-(xx**2 + yy**2) / (2.0 * sigma**2))
    return kernel / np.sum(kernel)
```

Listing 4: Gaussian Kernel Generation

Explanation

- Gaussian filters are used to smooth images by removing high-frequency noise while preserving structure.
- The kernel is computed using a 2D Gaussian distribution and normalized so that the total sum remains 1.
- The standard deviation σ controls the amount of blur: higher values result in stronger smoothing.
- This step is typically used before edge or corner detection to improve robustness.

Filtered Image Output

```
kernel = gaussian_kernel(11, 3.0)
gaussian_filtered = my_convolve2d(gray_np, kernel)
```

Figure 2: Image After Gaussian Filtering

5 Sobel Edge Detection

```
def sobel_operator(image):
    sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
    grad_x = my_convolve2d(image, sobel_x)
    grad_y = my_convolve2d(image, sobel_y)
    gradient_magnitude = np.sqrt(grad_x**2 + grad_y**2)
```

```
    return gradient_magnitude, grad_x, grad_y
```

Listing 5: Sobel Operator

Explanation

- Sobel operators compute image gradients in horizontal and vertical directions.
- The `sobel_x` and `sobel_y` kernels highlight changes in pixel values along x and y axes.
- Combining the two using Euclidean magnitude reveals edge strength at each location.
- This technique is a crucial step in edge-based image segmentation.

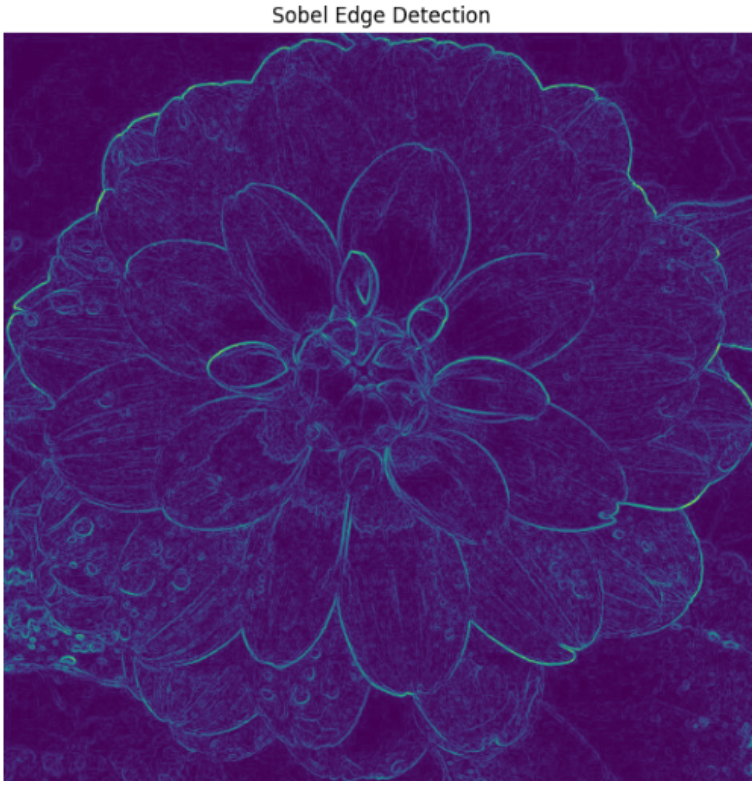

Figure 3: Edge Detection using Sobel Operator

6 Harris Corner Detection

```
def harris_detector(gray_np, k=0.04, threshold=0.01, kernel_size=11,
    sigma=3):
    sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
    Ix = my_convolve2d(gray_np, sobel_x)
```

```
    Iy = my_convolve2d(gray_np, sobel_y)
    Ixx = Ix**2; Iyy = Iy**2; Ixy = Ix * Iy
    gauss = gaussian_kernel(kernel_size, sigma)
    Sxx = my_convolve2d(Ixx, gauss)
    Syy = my_convolve2d(Iyy, gauss)
    Sxy = my_convolve2d(Ixy, gauss)
    det_M = Sxx * Syy - Sxy**2
    trace_M = Sxx + Syy
    R = det_M - k * (trace_M**2)
    corners = R > threshold * R.max()
    corner_coords = np.where(corners)
    return R, corners, corner_coords
```

Listing 6: Harris Detector

Explanation

- Harris detector identifies corners by analyzing intensity changes in multiple directions using image gradients.
- It calculates a structure tensor matrix M, then computes its determinant and trace to derive the corner response R.
- Points with high R values represent strong corners — essential for tracking, matching, and 3D reconstruction.

Figure 4: Corner Detection using Harris Operator

7 Non-Maximum Suppression

```
def non_max_suppression(R, window_size=3):
    H, W = R.shape
    pad = window_size // 2
    R_padded = np.pad(R, ((pad, pad), (pad, pad)), mode='constant',
       constant_values=-np.inf)
    suppressed = np.zeros_like(R, dtype=bool)
    for i in range(H):
        for j in range(W):
            current_value = R_padded[i + pad, j + pad]
            window = R_padded[i:i + window_size, j:j + window_size]
            if current_value == np.max(window):
                suppressed[i, j] = True
    return suppressed
```

Listing 7: Non-Maximum Suppression

Explanation

- This step enhances the output of Harris detection by keeping only the strongest corner in each neighborhood.
- It removes redundant or closely packed corner detections to ensure sparsity and clarity.
- A small window is moved across the image to select only local maxima in the response map.

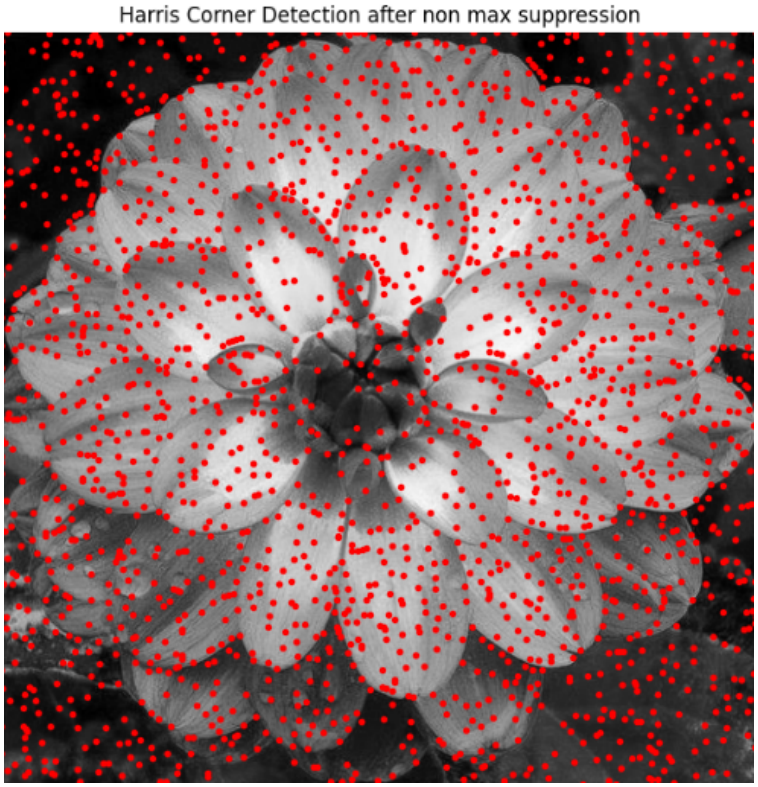

Figure 5: Refined Corners After Non-Maximum Suppression

Conclusion

This document provided a structured explanation and breakdown of a complete image processing pipeline. It included loading, grayscale conversion, convolution, smoothing, edge detection, corner detection, and feature refinement using non-maximum suppression. Each stage was explained with mathematical reasoning, code, and visual illustrations.