Aim

To perform edge detection on an image using the Canny edge detection algorithm, visualize the results, and demonstrate preprocessing steps like resizing, grayscale conversion, and blurring.

Objective

To apply edge detection techniques to identify and highlight edges in an image.

To understand the effect of different preprocessing steps on the edge detection process.

To visualize and compare the results of edge detection with various stages of preprocessing.

Summary

The project involves loading an image, resizing it, converting it to grayscale, applying Gaussian blur, and finally using the Canny edge detection algorithm to identify edges. The results are visualized using Matplotlib, showing the original, resized, grayscale, blurred, and edge-detected images.

Tools and Libraries Used

OpenCV: For image processing tasks like resizing, grayscale conversion, Gaussian blur, and edge detection.

Matplotlib: For visualizing images at various stages of processing.

imageio: Installed but not utilized in the provided code.

gdown: Installed but not utilized in the provided code.

Procedure

Install Necessary Libraries:

CODE:

```
!pip install opencv-python matplotlib imageio gdown
```

Import Libraries:

CODE:

```
import cv2
import os
from matplotlib import pyplot as plt
```

Load and Display the Image:

CODE:

```
IMG_PATH = os.path.join('data','images','bike.jpg')
```

```
img = cv2.imread(IMG_PATH)
plt.imshow(img)
```

Load the image from the specified path.

Display the image using Matplotlib.

Resize the Image:

CODE:

```
resized_img = cv2.resize(img, (int(img.shape[1]/2), int(img.shape[0]/2)))
plt.imshow(resized_img)
```

Resize the image to half its original dimensions.

Display the resized image.

Convert Image to Grayscale:

CODE:

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Convert the image to grayscale.

Apply Gaussian Blur:

CODE:

```
blur = cv2.GaussianBlur(gray, (5, 5), 0)
```

Apply Gaussian blur to the grayscale image to reduce noise.

Apply Canny Edge Detection:

CODE:

```
edges = cv2.Canny(blur, 100, 200)
resized_img1 = cv2.resize(edges, (int(img.shape[1]/2), int(img.shape[0]/2)))
```

Detect edges in the blurred image using the Canny edge detector.

Resize the edge-detected image for visualization.

Visualize Results:

CODE:

```
plt.imshow(edges)
plt.imshow(blur)
plt.imshow(gray)
```

Display the edge-detected, blurred, and grayscale images.

Highlights

Preprocessing: The project showcases the importance of preprocessing steps such as resizing, grayscale conversion, and blurring in improving the performance of edge detection algorithms.

Canny Edge Detection: The use of the Canny edge detection algorithm, a popular and effective method for detecting edges in images, highlights the application of advanced image processing techniques.

Visualization: The use of Matplotlib for visualizing images at different stages provides a clear understanding of how preprocessing affects edge detection.

Conclusion

The edge detection project effectively demonstrates the process of identifying edges in an image through various preprocessing steps and the Canny edge detection algorithm. By visualizing each step, it becomes evident how resizing, grayscale conversion, and blurring contribute to enhancing edge detection performance. The project provides a solid foundation for understanding edge detection and preprocessing techniques in image processing.