Image Enhancement using Filters

Introduction

Image enhancement using filters involves improving the visual quality of an image by reducing noise, smoothing textures, or highlighting features. This is especially useful before applying advanced computer vision techniques. Filtering techniques apply convolution operations to images using kernels to achieve effects like edge detection, smoothing, or sharpening.

Sobel Filter:

Detects edges by computing intensity gradients in the X (horizontal) or Y (vertical) direction.

Kernels: X-direction [[-1,0,1],[-2,0,2],[-1,0,1]], Y-direction [[-1,-2,-1],[0,0,0],[1,2,1]]. Combining X and Y gradients (gradient magnitude) highlights all edges.

Used in edge-based feature extraction.

Laplacian Filter:

Detects edges by computing the second derivative (zero crossings indicate edges).

Often applied after Gaussian blur to reduce noise, forming the Laplacian of Gaussian (LoG) operation.

Used for edge detection in noisy images.

Gaussian Blur:

Smooths images by applying a Gaussian kernel, reducing noise.

Commonly used as a preprocessing step for edge detection.

-->Filters enhance or extract specific features (e.g., edges, textures), critical for computer vision tasks like object detection and segmentation.

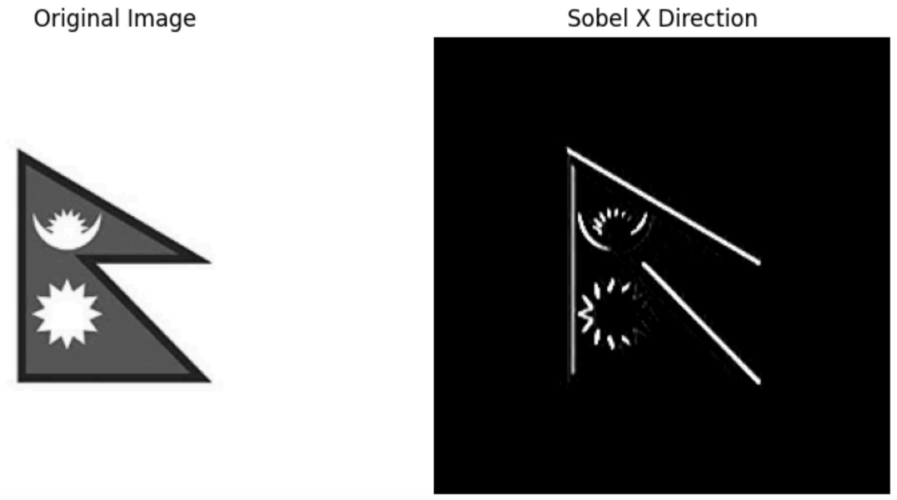
-->Combining filters (e.g., Gaussian blur + Laplacian) improves robustness in noisy conditions.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load image in grayscale
img = cv2.imread('/content/drive/MyDrive/Colab Notebooks/imageprocessingimages/flag.jpg', 0)

# Resize image
img = cv2.resize(img, (250, 250))
```

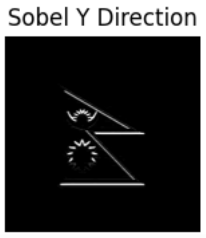
```
# Sobel kernels
kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])  # X-direction
kernel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])  # Y-direction

# Apply Sobel filters
grad_x = cv2.filter2D(img, -1, kernel_x)
grad_y = cv2.filter2D(img, -1, kernel_y)

# Compute gradient magnitude
gradient_magnitude = np.sqrt(grad_x**2 + grad_y**2)
gradient_magnitude = np.uint8(np.abs(gradient_magnitude))

# Apply Gaussian blur
blur = cv2.GaussianBlur(img, (3, 3), 0)

# Apply Laplacian filter
laplacian = cv2.Laplacian(blur, cv2.CV_64F)
```

```
# Display results
plt.figure(figsize=(15, 10))

plt.subplot(2, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(2, 3, 2)
plt.imshow(grad_x, cmap='gray')
plt.title('Sobel X Direction')
plt.axis('off')

plt.show()
```



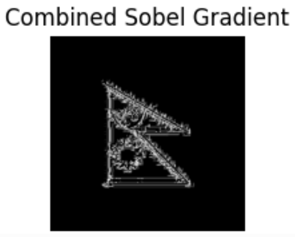
```
plt.subplot(2, 3, 3)
plt.imshow(grad_y, cmap='gray')
plt.title('Sobel Y Direction')
plt.axis('off')
```

```
(np.float64(-0.5), np.float64(249.5), np.float64(249.5), np.float64(-0.5))
```



```
plt.subplot(2, 3, 4)
plt.imshow(gradient_magnitude, cmap='gray')
plt.title('Combined Sobel Gradient')
plt.axis('off')
```

```
(np.float64(-0.5), np.float64(249.5), np.float64(249.5), np.float64(-0.5))
```



```
plt.subplot(2, 3, 5)
plt.imshow(blur, cmap='gray')
plt.title('Gaussian Blur')
plt.axis('off')
```

```
(np.float64(-0.5), np.float64(249.5), np.float64(249.5), np.float64(-0.5))
```

Gaussian Blur

```
plt.subplot(2, 3, 6)
plt.imshow(laplacian, cmap='gray')
plt.title('Laplacian')
plt.axis('off')
```

```
(np.float64(-0.5), np.float64(249.5), np.float64(249.5), np.float64(-0.5))
```

Laplacian