Bitwise Operations and Masking in OpenCV

Prepared by: Samriddha Pathak

Introduction

Bitwise operations work on the binary representation of numbers. Since digital images are essentially made up of pixels (and each pixel is made of bits), we can apply bitwise operations directly to image matrices using OpenCV.

Bitwise Operations in OpenCV

OpenCV provides built-in functions to perform bitwise operations:

Operation	Function	Description
AND	`cv2.bitwise_and()`	Keeps only the white regions where both images have white pixels.
OR	`cv2.bitwise_or()`	Keeps white where either of the images has white.
XOR	`cv2.bitwise_xor()`	Keeps white where images differ (exclusive white).
NOT	`cv2.bitwise_not()`	Inverts an image (black becomes white and vice versa).

Example: Bitwise Operations on Shapes

Listing 1: Performing Bitwise Operations

```
import cv2
import numpy as np

# Create black images
img1 = np.zeros((300, 300), dtype="uint8")
img2 = np.zeros((300, 300), dtype="uint8")

# Draw a white rectangle on img1
cv2.rectangle(img1, (50, 50), (250, 250), 255, -1)

# Draw a white circle on img2
cv2.circle(img2, (150, 150), 100, 255, -1)

# Perform bitwise operations
bit_and = cv2.bitwise_and(img1, img2)
```

```
bit_or = cv2.bitwise_or(img1, img2)
bit_xor = cv2.bitwise_xor(img1, img2)
bit_not = cv2.bitwise_not(img1)

# Display results
cv2.imshow("AND", bit_and)
cv2.imshow("OR", bit_or)
cv2.imshow("XOR", bit_xor)
cv2.imshow("NOT", bit_not)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Masking in OpenCV

A **mask** is a binary image (only containing 0s and 255s) used to isolate or focus on specific parts of another image. It tells OpenCV which parts of the image to **keep** or **ignore**.

- `255 (white)` – Pixel is kept
- `0 (black)` – Pixel is ignored

Use Case: Circular Region Masking

Listing 2: Masking a circular region

```
import cv2
import numpy as np

# Load an image
img = cv2.imread('your_image.jpg')

# Create a black mask
mask = np.zeros(img.shape[:2], dtype="uint8")

# Draw a white circle on the mask
cv2.circle(mask, (150, 150), 100, 255, -1)

# Apply the mask
masked_img = cv2.bitwise_and(img, img, mask=mask)

# Show outputs
cv2.imshow("Original", img)
cv2.imshow("Mask", mask)
cv2.imshow("Masked Image", masked_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Summary Table

Bitwise AND	Keeps overlapping features between two images.
Bitwise OR	Combines features from both images.
Bitwise XOR	Highlights differences between two images.
Bitwise NOT	Inverts all pixel values.
Masking	Focuses on a specific region of the image using a binary mask.

Applications

- Face and background separation
- Logo placement or watermarking
- Color filtering and segmentation
- Region-specific image enhancements
- Selective effects (e.g., grayscale background with color object)

Conclusion

Bitwise operations and masking are powerful tools in OpenCV that allow you to manipulate and extract specific image regions. These techniques are widely used in real-time object detection, segmentation, and feature extraction.