Module 18 - OPEN CV

Welcome to Module 18! OpenCV (Open Source Computer Vision Library) is a vast library of programming functions primarily aimed at real-time computer vision. It's used for everything from analysing images, recognizing faces, tracking objects, to processing videos.

Chapter 1: Introduction to Image Processing with OpenCV

This chapter lays the groundwork for working with images in OpenCV, covering fundamental concepts like image representation, installation, and basic image operations such as reading, writing, and resizing.

1.1 Images as Data

- **Digital Image Representation:**
  - In computer vision, images are typically represented as **multi-dimensional arrays (matrices)** of numbers.
  - Each number represents a **pixel's intensity** or color value at a specific coordinate.
  - **Grayscale Images:** Represented as a 2D array (Height x Width), where each pixel has a single value (e.g., 0-255 for 8-bit images, 0 being black, 255 being white).
  - **Color Images:** Typically represented as a 3D array (Height x Width x Channels).
    - Most commonly, color images use 3 channels: Red, Green, and Blue (RGB).
    - **OpenCV's Default:** OpenCV stores color images in **BGR (Blue, Green, Red)** format by default, not RGB. So, the channels are usually [Blue, Green, Red].
  - **Pixel Values (Depth):** Common image depths include:
    - `CV_8U` (8-bit unsigned integer): Pixel values range from 0 to 255. This is the most common type.
    - `CV_16U` (16-bit unsigned integer): 0 to 65535.
    - `CV_32F` (32-bit floating-point): 0.0 to 1.0 (or other ranges).

1.2 Installing OpenCV

- **Prerequisites:** Ensure you have Python installed.
- **Installation Command:** OpenCV can be installed using pip. It's recommended to install `opencv-python` which includes the main modules.

  Bash

  ```
  pip install opencv-python numpy
  ```

Note: `numpy` is often a dependency and is essential as OpenCV images are represented as NumPy arrays.

1.3 Computer Vision (CV) and OpenCV

- **Computer Vision (CV):**
  - An interdisciplinary field of artificial intelligence that trains computers to "see" and interpret visual information from the world, much like humans do.
  - It involves processing, analyzing, and understanding digital images and videos.
- **OpenCV (Open Source Computer Vision Library):**
  - A comprehensive, open-source library that provides a vast collection of algorithms and tools for various computer vision tasks.
  - It's highly optimized and written in C++, with interfaces for Python, Java, and MATLAB.
  - OpenCV is widely used in research, academia, and industry for applications like:
    - Image and video processing
    - Object detection (e.g., face detection, car detection)
    - Object tracking
    - Image segmentation
    - 3D reconstruction
    - Augmented reality
    - Medical imaging
    - Robotics

1.4 Reading an Image

- The `cv2.imread()` function is used to load an image from a specified file path.
- **Syntax:** `cv2.imread(filepath, flags)`
  - `filepath`: Path to the image file (e.g., `'my_image.jpg'`).
  - `flags`: Specifies how the image should be loaded.
    - `cv2.IMREAD_COLOR` (or `1`): Loads a color image. (Default)
    - `cv2.IMREAD_GRAYSCALE` (or `0`): Loads an image in grayscale.
    - `cv2.IMREAD_UNCHANGED` (or `-1`): Loads image including alpha channel.
- **Return Value:** A NumPy array representing the image. If the image cannot be loaded (e.g., file not found), it returns `None`.
- **Example:** *(Save an image named `sample_image.jpg` in the same directory as your Python script for this example to work.)*

  Python

```
import cv2

# Path to your image file
image_path = 'sample_image.jpg'

# Read the image in color mode
img_color = cv2.imread(image_path, cv2.IMREAD_COLOR)
```

```
# Read the image in grayscale mode
img_gray = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Check if images were loaded successfully
if img_color is None:
    print(f"Error: Could not read the image from {image_path}. Please
check the path and file.")
else:
    print(f"Color Image Shape: {img_color.shape}") # (height, width,
channels)
    print(f"Grayscale Image Shape: {img_gray.shape}") # (height,
width)

    # Display the images
    cv2.imshow('Original Color Image', img_color)
    cv2.imshow('Grayscale Image', img_gray)

    # Wait indefinitely for a key press (0 means wait forever)
    # It's crucial to have waitKey() after imshow()
    cv2.waitKey(0)

    # Destroy all opened OpenCV windows
    cv2.destroyAllWindows()
```

1.5 Writing an Image

- The `cv2.imwrite()` function is used to save an image to a specified file path.
- **Syntax:** `cv2.imwrite(filepath, image_array)`
  - `filepath`: Path where the image will be saved, including the desired file extension (e.g., `'output.png'`, `'grayscale_image.jpg'`). The extension determines the image format.
  - `image_array`: The NumPy array representing the image you want to save.
- **Return Value:** `True` if the image was successfully saved, `False` otherwise.
- **Example:**

  Python

```
import cv2

image_path = 'sample_image.jpg'
img = cv2.imread(image_path, cv2.IMREAD_COLOR)

if img is None:
    print("Error: Image not loaded for writing.")
else:
    # Save the original image as a PNG file
    output_path_png = 'output_image.png'
    cv2.imwrite(output_path_png, img)
    print(f"Image saved as {output_path_png}")

    # Convert to grayscale and save as a JPEG file
    img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Convert color
to grayscale
    output_path_gray_jpg = 'grayscale_output.jpg'
    cv2.imwrite(output_path_gray_jpg, img_gray)
    print(f"Grayscale image saved as {output_path_gray_jpg}")

    # You can optionally display the saved images
```

```
    # cv2.imshow('Saved PNG', cv2.imread(output_path_png))
    # cv2.imshow('Saved Grayscale JPG',
cv2.imread(output_path_gray_jpg))
    # cv2.waitKey(0)
    # cv2.destroyAllWindows()
```

1.6 Resizing an Image

- The `cv2.resize()` function is used to change the dimensions (width and height) of an image.
- **Syntax:** `cv2.resize(src, dsize, fx=0, fy=0, interpolation=cv2.INTER_LINEAR)`
    - `src`: The input image (NumPy array).
    - `dsize`: Desired output size as a tuple `(width, height)`. If `(0,0)`, then `fx` and `fy` must be non-zero.
    - `fx`, `fy`: Scaling factors along the horizontal and vertical axes, respectively. If `dsize` is non-zero, these are ignored. If `dsize` is `(0,0)`, then `dsize` is calculated as `(round(fx*src.cols), round(fy*src.rows))`.
    - `interpolation`: Algorithm used for resizing.
        - `cv2.INTER_AREA`: For shrinking images (fast and effective).
        - `cv2.INTER_LINEAR`: For zooming (default, good quality).
        - `cv2.INTER_CUBIC`: For zooming (slower but higher quality).
        - `cv2.INTER_NEAREST`: Simple, fast, but can lead to blocky results.
- **Example:**

  Python

```
import cv2

image_path = 'sample_image.jpg'
img = cv2.imread(image_path)

if img is None:
    print("Error: Image not loaded for resizing.")
else:
    height, width, _ = img.shape
    print(f"Original Dimensions: {width}x{height}")

    # Resize to a fixed dimension (e.g., 300x200 pixels)
    fixed_size = (300, 200) # (width, height)
    resized_img_fixed = cv2.resize(img, fixed_size,
interpolation=cv2.INTER_AREA)
    print(f"Resized to Fixed Dimensions:
{resized_img_fixed.shape[1]}x{resized_img_fixed.shape[0]}")

    # Resize using scaling factors (e.g., half size)
    scaled_img_half = cv2.resize(img, (0,0), fx=0.5, fy=0.5,
interpolation=cv2.INTER_AREA)
    print(f"Resized by Scaling Factor (0.5):
{scaled_img_half.shape[1]}x{scaled_img_half.shape[0]}")

    cv2.imshow('Original', img)
    cv2.imshow('Resized Fixed', resized_img_fixed)
    cv2.imshow('Resized Half', scaled_img_half)

    cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

Chapter 2: Image Transformations and Enhancements

This chapter explores various image transformations, including morphological operations, geometric transformations (flipping, shifting, rotating), and adding graphical elements.

2.7 - 2.8 Morphological Operations

Morphological operations are a set of image processing techniques that process images based on shapes. They are typically applied to binary images (black and white) but can be extended to grayscale images. They require two inputs: the image and a structuring element (or kernel).

- **Structuring Element (Kernel):** A small binary matrix or shape (e.g., rectangle, ellipse, cross) that defines the neighborhood to be examined around each pixel. `cv2.getStructuringElement()` is used to create these.

Erosion

- **Purpose:** Shrinks or "erodes" the boundaries of foreground objects (usually white). It effectively removes small white noises (speckles).
- **How it works:** A pixel in the output image is 1 only if *all* pixels under the structuring element in the input image are 1. Otherwise, it is 0.
- **Function:** `cv2.erode(src, kernel, iterations=1)`

Dilation

- **Purpose:** Expands or "dilates" the boundaries of foreground objects. It can be used to fill small holes in objects or to connect disconnected components.
- **How it works:** A pixel in the output image is 1 if at least one pixel under the structuring element in the input image is 1. Otherwise, it is 0.
- **Function:** `cv2.dilate(src, kernel, iterations=1)`

Opening

- **Definition:** Erosion followed by Dilation.
- **Purpose:** Useful for removing small objects or noise (speckles) while preserving the shape and size of larger objects. It smooths contours, breaks narrow isthmuses, and eliminates small protrusions.
- **Function:** `cv2.morphologyEx(src, cv2.MORPH_OPEN, kernel)`

Closing

- **Definition:** Dilation followed by Erosion.
- **Purpose:** Useful for filling small holes within foreground objects or connecting nearby objects. It smooths contours, fuses narrow breaks, and eliminates small holes.
- **Function:** `cv2.morphologyEx(src, cv2.MORPH_CLOSE, kernel)`

- **Example (Morphological Operations):** *(Use a simple binary image or convert your* `sample_image.jpg` *to binary for clearer results.)*

  Python

```
import cv2
import numpy as np

image_path = 'sample_image.jpg'
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE) # Convert to
grayscale for binary operations

if img is None:
    print("Error: Image not loaded for morphological operations.")
else:
    # Convert to binary image (e.g., thresholding)
    _, binary_img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

    # Define a 5x5 rectangular kernel
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))

    # Erosion
    eroded_img = cv2.erode(binary_img, kernel, iterations=1)

    # Dilation
    dilated_img = cv2.dilate(binary_img, kernel, iterations=1)

    # Opening (Erosion followed by Dilation)
    opened_img = cv2.morphologyEx(binary_img, cv2.MORPH_OPEN, kernel)

    # Closing (Dilation followed by Erosion)
    closed_img = cv2.morphologyEx(binary_img, cv2.MORPH_CLOSE,
kernel)

    cv2.imshow('Original Binary', binary_img)
    cv2.imshow('Eroded', eroded_img)
    cv2.imshow('Dilated', dilated_img)
    cv2.imshow('Opened', opened_img)
    cv2.imshow('Closed', closed_img)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Other Morphological Operations (Advanced)

- **Morphological Gradient:** `cv2.morphologyEx(src, cv2.MORPH_GRADIENT, kernel)`
  - **Definition:** Difference between dilation and erosion.
  - **Purpose:** Gives the outline of the object.
- **Top Hat:** `cv2.morphologyEx(src, cv2.MORPH_TOPHAT, kernel)`
  - **Definition:** Difference between the original image and its opening.
  - **Purpose:** Extracts "bright spots" (small elements) from the background that are smaller than the structuring element.
- **Black Hat:** `cv2.morphologyEx(src, cv2.MORPH_BLACKHAT, kernel)`
  - **Definition:** Difference between the closing of the input image and the input image.

- **Purpose:** Extracts "dark spots" from the background that are smaller than the structuring element.

2.9 Flipping an Image

- The `cv2.flip()` function flips an image around vertical, horizontal, or both axes.
- **Syntax:** `cv2.flip(src, flipCode)`
  - `src`: Input image.
  - `flipCode`:
    - `0`: Flip vertically (around the x-axis).
    - `1`: Flip horizontally (around the y-axis).
    - `-1`: Flip both horizontally and vertically.
- **Example:**

  Python

```
import cv2

image_path = 'sample_image.jpg'
img = cv2.imread(image_path)

if img is None:
    print("Error: Image not loaded for flipping.")
else:
    # Flip horizontally
    flipped_h = cv2.flip(img, 1)

    # Flip vertically
    flipped_v = cv2.flip(img, 0)

    # Flip both horizontally and vertically
    flipped_hv = cv2.flip(img, -1)

    cv2.imshow('Original', img)
    cv2.imshow('Flipped Horizontally', flipped_h)
    cv2.imshow('Flipped Vertically', flipped_v)
    cv2.imshow('Flipped Both', flipped_hv)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

2.10 - 2.11 Resizing an Image Based on Aspect Ratio

Maintaining the aspect ratio means that when you resize an image, you scale both the width and height by the same factor to prevent distortion.

- **Calculating new dimensions:**
  - If you target a `new_width`: `new_height = int(original_height * (new_width / original_width))`
  - If you target a `new_height`: `new_width = int(original_width * (new_height / original_height))`
  - Choose the dimension (width or height) that ensures the image fits within certain constraints or maintains quality.
- **Example:**

Python

```
import cv2

image_path = 'sample_image.jpg'
img = cv2.imread(image_path)

if img is None:
    print("Error: Image not loaded for aspect ratio resizing.")
else:
    original_height, original_width, _ = img.shape
    print(f"Original Dimensions: {original_width}x{original_height}")

    # --- Resizing based on a new width while maintaining aspect
ratio ---
    new_width = 400
    aspect_ratio = original_width / original_height
    new_height = int(new_width / aspect_ratio) # Calculate new height
    dim_by_width = (new_width, new_height)
    resized_by_width = cv2.resize(img, dim_by_width,
interpolation=cv2.INTER_AREA)
    print(f"Resized by new width ({new_width}):
{resized_by_width.shape[1]}x{resized_by_width.shape[0]}")

    # --- Resizing based on a new height while maintaining aspect
ratio ---
    new_height_alt = 300
    aspect_ratio_alt = original_width / original_height
    new_width_alt = int(new_height_alt * aspect_ratio_alt) #
Calculate new width
    dim_by_height = (new_width_alt, new_height_alt)
    resized_by_height = cv2.resize(img, dim_by_height,
interpolation=cv2.INTER_AREA)
    print(f"Resized by new height ({new_height_alt}):
{resized_by_height.shape[1]}x{resized_by_height.shape[0]}")

    cv2.imshow('Original', img)
    cv2.imshow(f'Resized (Width={new_width})', resized_by_width)
    cv2.imshow(f'Resized (Height={new_height_alt})',
resized_by_height)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

2.12 Adding Shapes and Text

OpenCV provides functions to draw basic geometric shapes and add text directly onto an image.

- **Colors:** In OpenCV, colors are typically specified as a tuple `(Blue, Green, Red)`. Each component ranges from 0 to 255.
    - Red: `(0, 0, 255)`
    - Green: `(0, 255, 0)`
    - Blue: `(255, 0, 0)`
    - White: `(255, 255, 255)`
    - Black: `(0, 0, 0)`
- **Drawing Functions:**

- **Rectangle:** `cv2.rectangle(img, pt1, pt2, color, thickness)`
  - `pt1`: Top-left corner `(x, y)`.
  - `pt2`: Bottom-right corner `(x, y)`.
  - `thickness`: Line thickness. `-1` for a filled rectangle.
- **Circle:** `cv2.circle(img, center, radius, color, thickness)`
  - `center`: Center coordinates `(x, y)`.
  - `radius`: Radius of the circle.
- **Line:** `cv2.line(img, pt1, pt2, color, thickness)`
  - `pt1`, `pt2`: Start and end points of the line.
- **Text:** `cv2.putText(img, text, org, fontFace, fontScale, color, thickness, lineType)`
  - `text`: String to be drawn.
  - `org`: Bottom-left corner of the text string in the image `(x, y)`.
  - `fontFace`: Font type (e.g., `cv2.FONT_HERSHEY_SIMPLEX`).
  - `fontScale`: Font size multiplier.

- **Example:**

Python

```
import cv2
import numpy as np

# Create a blank black image (500x500 pixels, 3 channels, 8-bit
unsigned integer)
img_canvas = np.zeros((500, 500, 3), dtype=np.uint8)

# 1. Draw a green rectangle
# Top-left (50,50), Bottom-right (200,200), Green (0,255,0),
Thickness 3
cv2.rectangle(img_canvas, (50, 50), (200, 200), (0, 255, 0), 3)

# 2. Draw a filled blue rectangle
cv2.rectangle(img_canvas, (250, 50), (450, 200), (255, 0, 0), -1)

# 3. Draw a red circle
# Center (125, 325), Radius 75, Red (0,0,255), Thickness 2
cv2.circle(img_canvas, (125, 325), 75, (0, 0, 255), 2)

# 4. Draw a filled yellow circle
cv2.circle(img_canvas, (350, 325), 100, (0, 255, 255), -1)

# 5. Draw a thick purple line
cv2.line(img_canvas, (50, 450), (450, 450), (255, 0, 255), 5)

# 6. Add text
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(img_canvas, 'OpenCV Shapes & Text', (10, 40), font, 1,
(255, 255, 255), 2, cv2.LINE_AA)
cv2.putText(img_canvas, 'Hello!', (280, 250), font, 1.5, (0, 165,
255), 3, cv2.LINE_AA) # Orange text

cv2.imshow('Shapes and Text', img_canvas)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

2.13 Shifting an Image (Translation)

- Shifting (or translating) an image moves it along the x and y axes.
- This is done using an **affine transformation**, which is represented by a 2x3 transformation matrix `M`.
- **Translation Matrix (M):** $M = \\begin{pmatrix} 1 & 0 & t\_x \\ 0 & 1 & t\_y \\end{pmatrix}$ where t_x is the shift along x-axis and t_y is the shift along y-axis.
    - Positive t_x shifts right, negative t_x shifts left.
    - Positive t_y shifts down, negative t_y shifts up.
- **Function:** `cv2.warpAffine(src, M, dsize)`
    - `src`: Input image.
    - `M`: The 2x3 translation matrix (NumPy array of `float32`).
    - `dsize`: Size of the output image `(width, height)`.
- **Example:**

  Python

```
import cv2
import numpy as np

image_path = 'sample_image.jpg'
img = cv2.imread(image_path)

if img is None:
    print("Error: Image not loaded for shifting.")
else:
    height, width, _ = img.shape

    # Define the amount to shift (tx, ty)
    # Shift 100 pixels right, 50 pixels down
    tx, ty = 100, 50

    # Create the 2x3 translation matrix
    M = np.float32([[1, 0, tx], [0, 1, ty]])

    # Apply the affine transformation
    shifted_img = cv2.warpAffine(img, M, (width, height)) # Output
size same as original

    cv2.imshow('Original', img)
    cv2.imshow(f'Shifted by ({tx},{ty})', shifted_img)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

2.14 Image Rotation

- Rotating an image is also an affine transformation.
- **Function to get rotation matrix:** `cv2.getRotationMatrix2D(center, angle, scale)`
    - `center`: Point around which the image will be rotated `(x, y)`. Often the image center `(width / 2, height / 2)`.
    - `angle`: Angle of rotation in degrees. Positive value means counter-clockwise rotation.
    - `scale`: Scaling factor for the image. `1.0` for no scaling.
- **Function to apply rotation:** `cv2.warpAffine(src, M, dsize)` (same as shifting).
- **Example:**

Python

```
import cv2
import numpy as np

image_path = 'sample_image.jpg'
img = cv2.imread(image_path)

if img is None:
    print("Error: Image not loaded for rotation.")
else:
    height, width, _ = img.shape
    center = (width // 2, height // 2) # Center of the image

    # Rotate by 45 degrees counter-clockwise, with no scaling
    angle = 45
    scale = 1.0
    M_45 = cv2.getRotationMatrix2D(center, angle, scale)
    rotated_45 = cv2.warpAffine(img, M_45, (width, height))

    # Rotate by -90 degrees (90 degrees clockwise), half scale
    angle_neg = -90
    scale_half = 0.5
    M_neg90 = cv2.getRotationMatrix2D(center, angle_neg, scale_half)
    rotated_neg90 = cv2.warpAffine(img, M_neg90, (width, height))

    cv2.imshow('Original', img)
    cv2.imshow(f'Rotated {angle} degrees', rotated_45)
    cv2.imshow(f'Rotated {angle_neg} degrees (Scaled)',
rotated_neg90)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Note: When rotating, parts of the image might get cropped if the output size `dsize` is kept the same as the original image's dimensions. To avoid cropping, you would need to calculate a new `dsize` to fit the rotated image.

Chapter 3: Image Filtering and Feature Detection

This chapter covers various image filtering techniques used for noise reduction and edge detection, crucial steps in many computer vision pipelines.

3.15 Thresholding

- Thresholding is a simple yet effective image segmentation technique that converts a grayscale image into a binary image.
- Pixels below a certain threshold value are set to one value (e.g., black), and pixels above it are set to another value (e.g., white).
- **Function:** `cv2.threshold(src, thresh, maxval, type)`
    - `src`: Input grayscale image.
    - `thresh`: The threshold value.
    - `maxval`: The value to be given if pixel value is more than the threshold value.

    - `type`: The type of thresholding to be applied.
- **Thresholding Types:**
    - `cv2.THRESH_BINARY`: `pixel > thresh` -> `maxval`, else `0`
    - `cv2.THRESH_BINARY_INV`: `pixel > thresh` -> `0`, else `maxval`
    - `cv2.THRESH_TRUNC`: `pixel > thresh` -> `thresh`, else `pixel`
    - `cv2.THRESH_TOZERO`: `pixel > thresh` -> `pixel`, else `0`
    - `cv2.THRESH_TOZERO_INV`: `pixel > thresh` -> `0`, else `pixel`
    - `cv2.THRESH_OTSU`: Automatically calculates the optimal threshold value (often used with `cv2.THRESH_BINARY`).
- **Example:**

Python

```
import cv2

image_path = 'sample_image.jpg'
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE) # Thresholding
works best on grayscale

if img is None:
    print("Error: Image not loaded for thresholding.")
else:
    # Define threshold value and max value
    thresh_val = 127
    max_val = 255

    # Binary Thresholding
    _, th_binary = cv2.threshold(img, thresh_val, max_val,
cv2.THRESH_BINARY)

    # Inverse Binary Thresholding
    _, th_binary_inv = cv2.threshold(img, thresh_val, max_val,
cv2.THRESH_BINARY_INV)

    # Truncate Thresholding
    _, th_trunc = cv2.threshold(img, thresh_val, max_val,
cv2.THRESH_TRUNC)

    # To Zero Thresholding
    _, th_tozero = cv2.threshold(img, thresh_val, max_val,
cv2.THRESH_TOZERO)

    # To Zero Inverse Thresholding
    _, th_tozero_inv = cv2.threshold(img, thresh_val, max_val,
cv2.THRESH_TOZERO_INV)

    # Otsu's Binarization (automatically finds optimal threshold)
    # It returns the optimal threshold value too
    ret_otsu, th_otsu = cv2.threshold(img, 0, max_val,
cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    print(f"Otsu's Threshold: {ret_otsu}")

    cv2.imshow('Original Grayscale', img)
    cv2.imshow('Binary', th_binary)
    cv2.imshow('Binary Inverse', th_binary_inv)
    cv2.imshow('Trunc', th_trunc)
    cv2.imshow('To Zero', th_tozero)
    cv2.imshow('To Zero Inverse', th_tozero_inv)
```

```
    cv2.imshow('Otsu Binarization', th_otsu)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

3.16 Gaussian Blur

- Gaussian blur is a widely used image smoothing technique that reduces image noise and detail.
- It uses a Gaussian function to calculate the transformation to be applied to each pixel in the image.
- **Function:** `cv2.GaussianBlur(src, ksize, sigmaX, sigmaY=0, borderType=cv2.BORDER_DEFAULT)`
    - `src`: Input image.
    - `ksize`: Gaussian kernel size. It's a tuple `(width, height)`. Both width and height must be positive and **odd**.
    - `sigmaX`: Gaussian kernel standard deviation in X direction. If 0, it's calculated from `ksize.width`.
    - `sigmaY`: Gaussian kernel standard deviation in Y direction. If 0, it's taken as equal to `sigmaX`.
- **Example:**

  Python

```
import cv2

image_path = 'sample_image.jpg'
img = cv2.imread(image_path)

if img is None:
    print("Error: Image not loaded for Gaussian blur.")
else:
    # Apply Gaussian blur with a 5x5 kernel and sigmaX=0
    blurred_img = cv2.GaussianBlur(img, (5, 5), 0)

    # Apply Gaussian blur with a larger kernel (e.g., 15x15) for more
blur
    more_blurred_img = cv2.GaussianBlur(img, (15, 15), 0)

    cv2.imshow('Original', img)
    cv2.imshow('Gaussian Blurred (5x5)', blurred_img)
    cv2.imshow('Gaussian Blurred (15x15)', more_blurred_img)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

3.17 Median Blur

- Median blur is a non-linear filtering technique effective at removing **salt-and-pepper noise** (random dark or light pixels).
- It replaces each pixel's value with the median value of all the pixels in its neighborhood (defined by the kernel size).
- **Function:** `cv2.medianBlur(src, ksize)`
    - `src`: Input image.

    - `ksize`: Aperture linear size. It must be an **odd integer greater than 1** (e.g., 3, 5, 7...).
- **Example:**

  Python

```
import cv2
import numpy as np

# Create a noisy image for demonstration (e.g., add salt-and-pepper
noise)
img = cv2.imread('sample_image.jpg')
if img is None:
    print("Error: Image not loaded for median blur example.")
    exit()

# Add artificial salt-and-pepper noise
noisy_img = np.copy(img)
num_salt = int(0.01 * img.size)
coords = [np.random.randint(0, i - 1, num_salt) for i in img.shape]
noisy_img[coords[0], coords[1], :] = 255 # Salt
num_pepper = int(0.01 * img.size)
coords = [np.random.randint(0, i - 1, num_pepper) for i in img.shape]
noisy_img[coords[0], coords[1], :] = 0 # Pepper

# Apply Median Blur with a 5x5 kernel
median_blurred_img = cv2.medianBlur(noisy_img, 5)

cv2.imshow('Original Noisy Image', noisy_img)
cv2.imshow('Median Blurred (5x5)', median_blurred_img)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

3.18 Bilateral Filter

- The bilateral filter is a non-linear, edge-preserving smoothing filter.
- Unlike other blurring filters (like Gaussian) that simply average pixel values in a neighborhood, the bilateral filter considers both spatial proximity and intensity similarity.
- This means it blurs similar colors together while preserving sharp edges.
- **Function:** `cv2.bilateralFilter(src, d, sigmaColor, sigmaSpace, borderType=cv2.BORDER_DEFAULT)`
    - `src`: Input image.
    - `d`: Diameter of each pixel neighborhood. A larger `d` means a larger neighborhood for blurring. If non-positive, it's computed from `sigmaSpace`.
    - `sigmaColor`: Filter sigma in the color space. A larger value means more colors in the neighborhood will be considered for blurring.
    - `sigmaSpace`: Filter sigma in the coordinate space. A larger value means farther pixels will influence each other as long as their colors are close enough.
- **Example:**

  Python

```
import cv2

image_path = 'sample_image.jpg'
img = cv2.imread(image_path)

if img is None:
    print("Error: Image not loaded for bilateral filter.")
else:
    # Apply Bilateral Filter
    # d=9 (neighborhood diameter), sigmaColor=75 (color similarity),
sigmaSpace=75 (spatial closeness)
    bilateral_filtered_img = cv2.bilateralFilter(img, 9, 75, 75)

    cv2.imshow('Original', img)
    cv2.imshow('Bilateral Filtered', bilateral_filtered_img)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

3.19 Edge Detection

- Edge detection algorithms identify points in a digital image at which the image brightness changes sharply or has discontinuities. These points typically correspond to object boundaries.

Canny Edge Detector

- One of the most widely used and effective edge detection algorithms.
- **Steps:**
    1. **Noise Reduction:** Applies a Gaussian blur to smooth the image and remove noise.
    2. **Gradient Calculation:** Finds the intensity gradients of the image.
    3. **Non-maximum Suppression:** Thins out the edges, keeping only the strongest response.
    4. **Hysteresis Thresholding:** Identifies strong edges and includes weak edges that are connected to strong edges.
- **Function:** `cv2.Canny(image, threshold1, threshold2, apertureSize=3, L2gradient=False)`
    - `image`: Input 8-bit image (often grayscale).
    - `threshold1`, `threshold2`: Hysteresis thresholding values. Edges with gradients greater than `threshold2` are strong edges. Edges with gradients between `threshold1` and `threshold2` are considered if they are connected to strong edges.
- **Example (Canny):**

  Python

```
import cv2

image_path = 'sample_image.jpg'
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE) # Canny works best
on grayscale

if img is None:
    print("Error: Image not loaded for Canny edge detection.")
```

```
else:
    # Apply Canny Edge Detector
    # Low threshold: 50, High threshold: 150
    edges = cv2.Canny(img, 50, 150)

    cv2.imshow('Original Grayscale', img)
    cv2.imshow('Canny Edges', edges)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Sobel and Scharr Operators

- These are derivative operators that calculate the gradient magnitude and orientation for each pixel, highlighting areas of sharp intensity change.
- **Sobel Function:** `cv2.Sobel(src, ddepth, dx, dy, ksize=3)`
  - `ddepth`: Output image depth (e.g., `cv2.CV_64F` for float type).
  - `dx`, `dy`: Order of the derivative in x and y direction (e.g., `(1,0)` for x-gradient, `(0,1)` for y-gradient).
- **Scharr Operator:** Similar to Sobel but more accurate for rotational invariance. Used with `cv2.Scharr()`.

Laplacian Operator

- Calculates the second derivative of the image, which can highlight regions of rapid intensity change.
- **Function:** `cv2.Laplacian(src, ddepth, ksize=1)`
- **Example (Sobel and Laplacian):**

  Python

```
import cv2
import numpy as np

image_path = 'sample_image.jpg'
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

if img is None:
    print("Error: Image not loaded for Sobel/Laplacian.")
else:
    # Sobel in X direction
    sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)
    sobelx = np.uint8(np.absolute(sobelx)) # Convert back to 8-bit

    # Sobel in Y direction
    sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)
    sobely = np.uint8(np.absolute(sobely)) # Convert back to 8-bit

    # Combine X and Y gradients (approximate magnitude)
    sobel_combined = cv2.addWeighted(sobelx, 0.5, sobely, 0.5, 0)

    # Laplacian
    laplacian = cv2.Laplacian(img, cv2.CV_64F)
    laplacian = np.uint8(np.absolute(laplacian)) # Convert back to 8-
bit
```

```
    cv2.imshow('Original Grayscale', img)
    cv2.imshow('Sobel X', sobelx)
    cv2.imshow('Sobel Y', sobely)
    cv2.imshow('Sobel Combined', sobel_combined)
    cv2.imshow('Laplacian', laplacian)

    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Chapter 4: Video Processing and Webcam Access

This chapter extends OpenCV's capabilities from still images to video streams, covering how to read, write, and capture video from sources like a webcam.

4.20 Reading a Video

- `cv2.VideoCapture()` is used to read video files or stream from cameras.
- **Steps:**
    1. Create a `VideoCapture` object, passing the video file path or device index (e.g., `0` for default webcam).
    2. Check if the video capture was successful.
    3. Loop to read frames one by one using `cap.read()`.
    4. Process each `frame`.
    5. Display the `frame` using `cv2.imshow()`.
    6. Set a `cv2.waitKey()` to allow for display and exit condition.
    7. Release the `VideoCapture` object and destroy windows when done.
- **Example:** *(Save a video file named `sample_video.mp4` in the same directory, or use your webcam by changing the path to `0`.)*

    Python

```
import cv2

video_path = 'sample_video.mp4' # Replace with your video file path
or 0 for webcam

# Create a VideoCapture object
cap = cv2.VideoCapture(video_path)

# Check if video opened successfully
if not cap.isOpened():
    print(f"Error: Could not open video source at {video_path}")
    exit()

print("Reading video... Press 'q' to quit.")

while True:
    # Read a frame from the video
    ret, frame = cap.read() # ret is True if frame is read correctly,
frame is the image

    # If frame is not read correctly, ret will be False, indicating
end of video
    if not ret:
```

```
            print("End of video stream or error reading frame.")
            break

        # Display the resulting frame
        cv2.imshow('Video Frame', frame)

        # Press 'q' on the keyboard to exit the video playback
        if cv2.waitKey(25) & 0xFF == ord('q'): # 25ms delay between
frames, adjusts playback speed
            break

# Release the video capture object
cap.release()

# Destroy all the windows
cv2.destroyAllWindows()
```

4.21 Writing a Video

- `cv2.VideoWriter()` is used to save video streams to a file.
- **Steps:**
    1. Define a `FourCC` code (codec) for video compression.
    2. Create a `VideoWriter` object specifying the output filename, codec, frame rate (FPS), and frame dimensions.
    3. In your video processing loop, write each processed frame using `out.write()`.
    4. Release the `VideoWriter` object when done.
- **Example (Recording Webcam Feed):**

    Python

```
import cv2

# --- Configuration for output video ---
output_filename = 'output_webcam_video.avi'
frame_width = 640  # Desired width for output video
frame_height = 480 # Desired height for output video
fps = 20 # Frames per second

# Define the codec and create VideoWriter object
# FourCC code for AVI (e.g., XVID, MJPG, DIVX). 'XVID' is common and
widely supported.
# FourCC for MP4: cv2.VideoWriter_fourcc(*'MP4V') or
cv2.VideoWriter_fourcc(*'avc1')
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter(output_filename, fourcc, fps, (frame_width,
frame_height))

# Access the default webcam (0)
cap = cv2.VideoCapture(0)

if not cap.isOpened():
    print("Error: Could not open webcam.")
    exit()

print(f"Recording video to '{output_filename}'... Press 'q' to stop
recording.")
```

```
while True:
    ret, frame = cap.read()

    if not ret:
        print("Failed to grab frame from webcam. Exiting...")
        break

    # Resize frame to match writer's dimensions if needed
(important!)
    frame_resized = cv2.resize(frame, (frame_width, frame_height))

    # Write the flipped frame to the output file
    out.write(frame_resized)

    # Display the frame (optional)
    cv2.imshow('Recording Webcam Feed', frame_resized)

    # Press 'q' to quit
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release everything when the job is finished
cap.release()
out.release()
cv2.destroyAllWindows()
print(f"Video recording finished. Saved to '{output_filename}'")
```

4.22 Accessing the Webcam

- Accessing the webcam is just a specific case of reading a video, where the video source is your camera device.
- The device index `0` typically refers to the default webcam. If you have multiple cameras, `1`, `2`, etc., might refer to others.
- **Example (Live Webcam Feed):**

  Python

```
import cv2

# Access the default webcam (0)
cap = cv2.VideoCapture(0)

# Check if the webcam opened successfully
if not cap.isOpened():
    print("Error: Could not open webcam. Make sure it's connected and
not in use.")
    exit()

print("Webcam feed active. Press 'q' to quit.")

while True:
    # Read a frame from the webcam
    ret, frame = cap.read()

    # If frame is not read correctly, ret will be False
    if not ret:
        print("Failed to grab frame. Exiting...")
```

```
            break

        # You can perform image processing on the 'frame' here
        # For example, convert to grayscale:
        # gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Display the original frame
        cv2.imshow('Live Webcam Feed (Press Q to Quit)', frame)
        # Or display the processed frame:
        # cv2.imshow('Grayscale Webcam Feed', gray_frame)

        # Wait for 1 millisecond. If 'q' is pressed, break the loop.
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

# Release the webcam resource
cap.release()

# Destroy all OpenCV windows
cv2.destroyAllWindows()
```