SUPERIOR UNIVERSITY

Submitted By:

Name: **Um e Habiba**

Roll number **Su92-bsdsm-f23-040**

Section: **4A**

Task: **05**

Subject: **PAI(lab)**

Submitted to: **Sir Rasikh Ali**

Installing OpenCV

Using OpenCV

```
%pip install opencv-python
```

✓ 5.2s Python

```
Note: you may need to restart the kernel to use updated packages.
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: opencv-python in c:\users\my computer\appdata\roaming\python\python312\site-packages (4.10.0.84)
Requirement already satisfied: numpy>=1.21.2 in c:\users\my computer\appdata\roaming\python\python312\site-packages (from opencv-python) (1.26.4)

[notice] A new release of pip is available: 24.3.1 -> 25.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Importing Required Libraries

- `cv2`: The OpenCV library for image processing.
- `numpy`: Used for numerical operations on image data.
- `matplotlib.pyplot`: Used for displaying images.

Libraries

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

[2] ✓ 2.3s

Reading and Displaying an Image

- Loads an image from the file system.
- Displays the image in a separate window.
- Waits for a key press before closing the window.

Working with Images

2.1 Getting Started

Reading an image

```
image = cv2.imread('animal.png')
if image is None:
    print("Image is note loaded check the path!")
else:
    print("Image loaded successfully !")
```

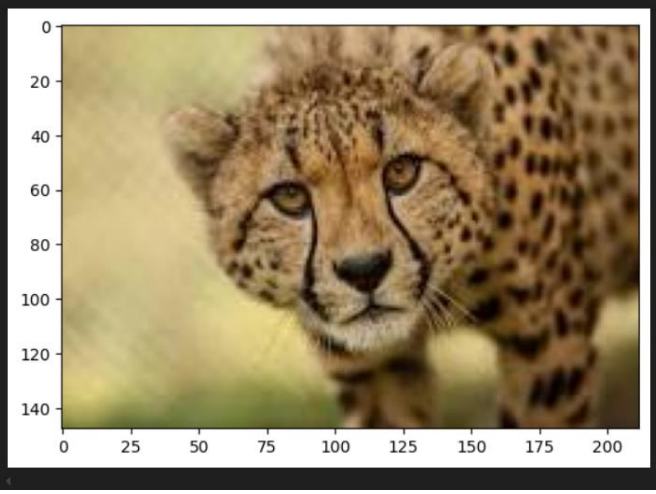
✓ 0.0s

```
Image loaded successfully !
```

Display an image

```
import matplotlib.pyplot as plt

img = cv2.imread('animal.png')  # Use the existing image path
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
```

[4] ✓ 0.4s



Converting to Grayscale

Converts the image from BGR (default in OpenCV) to grayscale.

```python
import cv2
import matplotlib.pyplot as plt
img = cv2.imread('animal.png', cv2.IMREAD_GRAYSCALE)
if img is None:
    print("Error: Image not found.")
else:
    resized_img = cv2.resize(img, (400, 300))
    plt.imshow(resized_img, cmap='gray')
    plt.axis("off")
    plt.show()
```

✓ 0.1s

```python
cv2.imwrite('animal.png', img)
print("Grayscale image saved successfully as 'animal.png'!")
```

✓ 0.0s

```
Grayscale image saved successfully as 'animal.png'!
```

Writing an image in OpenCV using Python

Image Check & Read: It checks if 'animal.png' exists. If found, it reads the image using cv2.imread().

Image Save: If the image is successfully read, it saves it as 'savedImage.jpg' in the current directory.

Image Display: The saved image is read again, converted from **BGR to RGB**, and displayed using **Matplotlib**.

```python
import cv2
import os
import matplotlib.pyplot as plt
image_path = 'animal.png'
directory = os.getcwd()
if not os.path.exists(image_path):
    print(f"Error: Image not found at '{image_path}'")
else:
    img = cv2.imread(image_path)
    if img is None:
        print("Error: Failed to read the image.")
    else:
        os.makedirs(directory, exist_ok=True)
        filename = os.path.join(directory, 'savedImage.jpg')
        success = cv2.imwrite(filename, img)
        if success:
            print("Image saved successfully as", filename)
        else:
            print("Error: Image saving failed.")
        saved_img = cv2.imread(filename, cv2.IMREAD_COLOR)
        saved_img = cv2.cvtColor(saved_img, cv2.COLOR_BGR2RGB)
        plt.imshow(saved_img)
        plt.axis("off")
        plt.show()
```

```
Image saved successfully as c:\Users\My Computer\Desktop\lab 5 task pai\savedImage.jpg
```

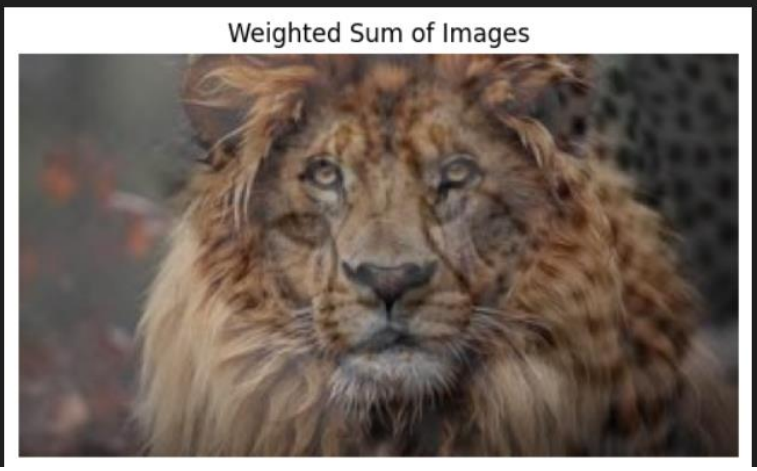
Arithmetic operations on Images

This code blends two images using **weighted addition** in OpenCV. It loads and resizes image2 to match image1, then combines them with weights (50% and 40%) using cv2.addWeighted(). The result is converted to RGB and displayed using Matplotlib.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
image1 = cv2.imread('image1.png')
image2 = cv2.imread('animal.png')
if image1 is None or image2 is None:
    print("Error: One or both images could not be loaded. Check file paths.")
else:
    image2 = cv2.resize(image2, (image1.shape[1], image1.shape[0]))
    weightedSum = cv2.addWeighted(image1, 0.5, image2, 0.4, 0)
    weightedSum_rgb = cv2.cvtColor(weightedSum, cv2.COLOR_BGR2RGB)
    plt.imshow(weightedSum_rgb)
    plt.axis("off")
    plt.title("Weighted Sum of Images")
    plt.show()
```

[ ]

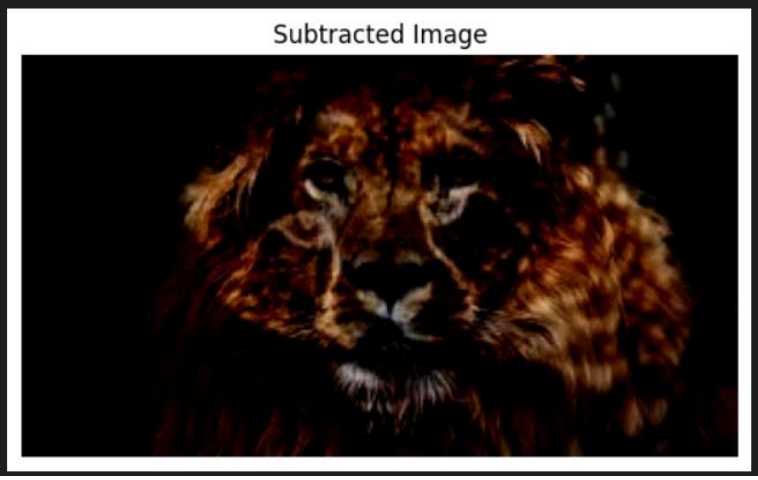
...



This code **subtracts two images** using OpenCV. It loads and resizes image2 to match image1, then applies cv2.subtract() to compute the pixel-wise difference. The result is converted to RGB and displayed using Matplotlib.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
image1 = cv2.imread('image1.png')
image2 = cv2.imread('animal.png')
if image1 is None or image2 is None:
    print("Error: One or both images could not be loaded. Check file paths.")
else:
    image2 = cv2.resize(image2, (image1.shape[1], image1.shape[0]))
    subtracted_image = cv2.subtract(image1, image2)
    subtracted_image_rgb = cv2.cvtColor(subtracted_image, cv2.COLOR_BGR2RGB)
    plt.imshow(subtracted_image_rgb)
    plt.axis("off")
    plt.title("Subtracted Image")
    plt.show()
```

[13]



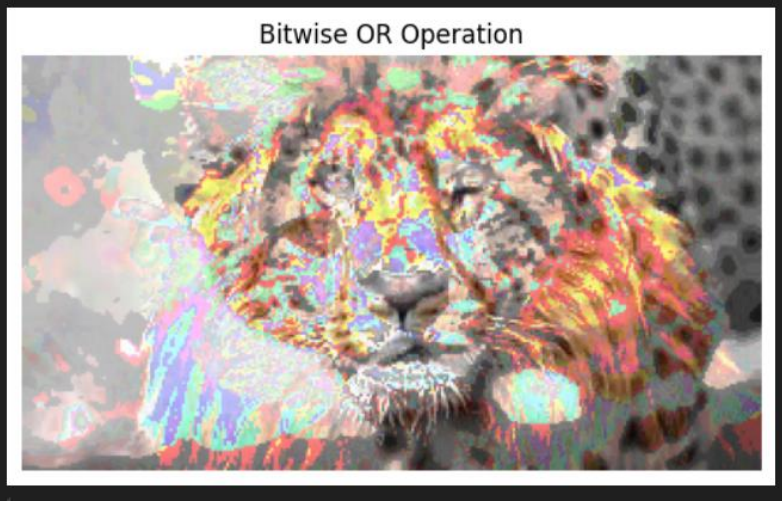
This code performs a **bitwise AND operation** on two images using OpenCV. It loads and resizes img2 to match img1, applies cv2.bitwise_and() to keep only the overlapping pixel values, converts the result to RGB, and displays it using Matplotlib.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
img1 = cv2.imread('image1.png')
img2 = cv2.imread('animal.png')
if img1 is None or img2 is None:
    print("Error: One or both images could not be loaded. Check file paths.")
else:
    img2 = cv2.resize(img2, (img1.shape[1], img1.shape[0]))
    dest_and = cv2.bitwise_and(img1, img2)
    dest_and_rgb = cv2.cvtColor(dest_and, cv2.COLOR_BGR2RGB)
    plt.imshow(dest_and_rgb)
    plt.axis("off")
    plt.title("Bitwise AND Operation")
    plt.show()
```

[14]

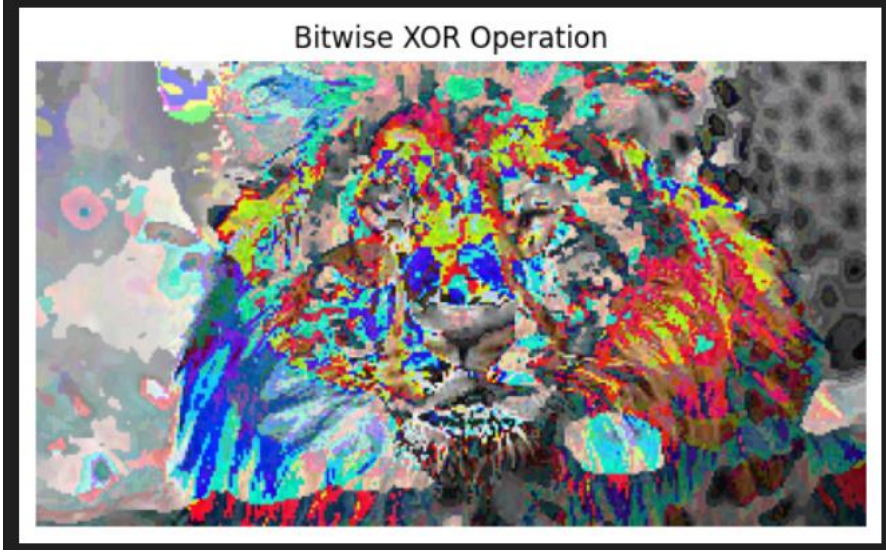
This code performs a **bitwise OR operation** on two images using OpenCV. It loads and resizes img2 to match img1, applies cv2.bitwise_or() to combine pixel values, converts the result to RGB, and displays it using Matplotlib.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
img1 = cv2.imread('image1.png')
img2 = cv2.imread('animal.png')
if img1 is None or img2 is None:
    print("Error: One or both images could not be loaded. Check file paths.")
else:
    img2 = cv2.resize(img2, (img1.shape[1], img1.shape[0]))
    dest_or = cv2.bitwise_or(img1, img2)
    dest_or_rgb = cv2.cvtColor(dest_or, cv2.COLOR_BGR2RGB)
    plt.imshow(dest_or_rgb)
    plt.axis("off")
    plt.title("Bitwise OR Operation")
    plt.show()
```



This code performs a **bitwise XOR operation** on two images using OpenCV. It loads and resizes img2 to match img1, applies cv2.bitwise_xor() to keep non-overlapping pixel values, converts the result to RGB, and displays it using Matplotlib.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
img1 = cv2.imread('image1.png')
img2 = cv2.imread('animal.png')
if img1 is None or img2 is None:
    print("Error: One or both images could not be loaded. Check file paths.")
else:
    img2 = cv2.resize(img2, (img1.shape[1], img1.shape[0]))
    dest_xor = cv2.bitwise_xor(img1, img2)
    dest_xor_rgb = cv2.cvtColor(dest_xor, cv2.COLOR_BGR2RGB)
    plt.imshow(dest_xor_rgb)
    plt.axis("off")
    plt.title("Bitwise XOR Operation")
    plt.show()
```



Bitwise Operations on Binary Images

This code performs a **bitwise NOT operation** on two images using OpenCV. It inverts pixel values of img1 and img2, converts them to RGB, and displays both results side by side using Matplotlib.

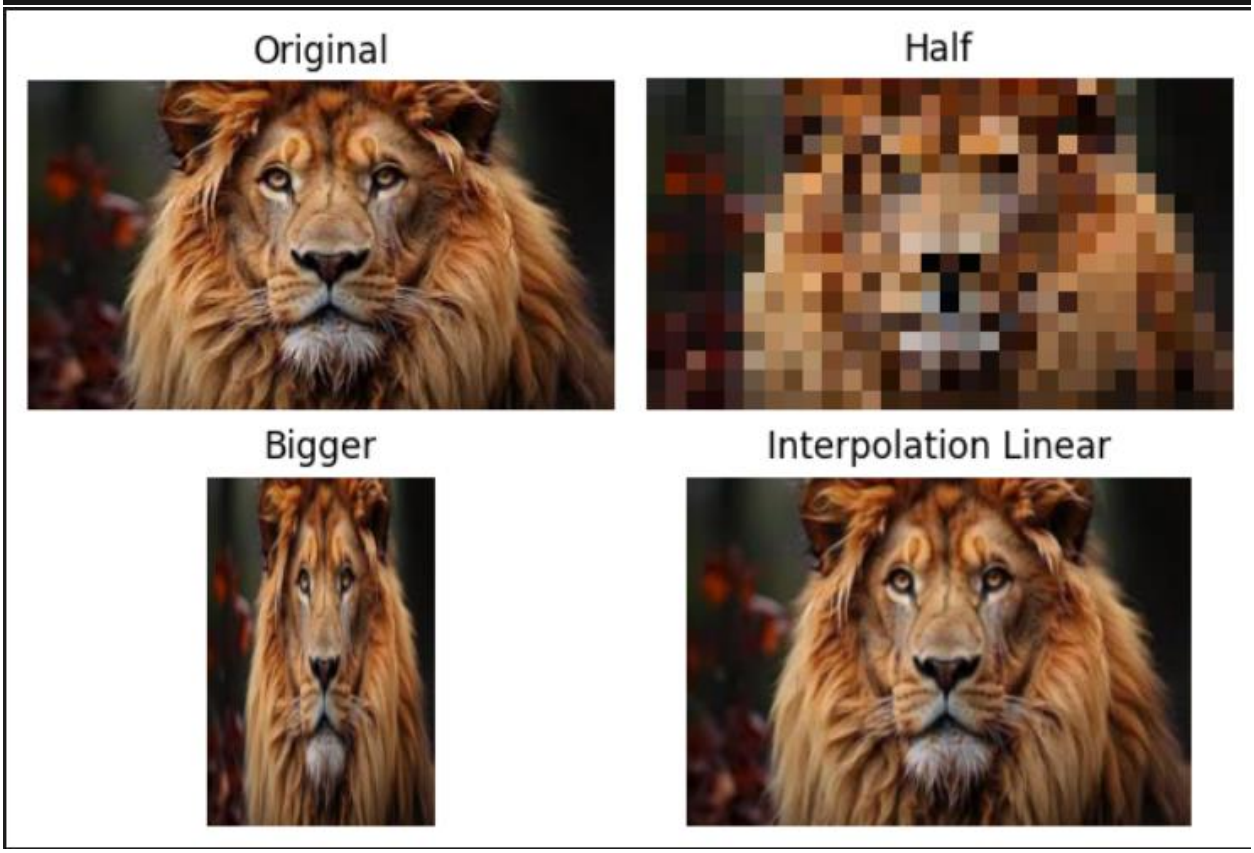
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
img1 = cv2.imread('image1.png')
img2 = cv2.imread('animal.png')
if img1 is None or img2 is None:
    print("Error: One or both images could not be loaded. Check file paths.")
else:
    dest_not1 = cv2.bitwise_not(img1)
    dest_not2 = cv2.bitwise_not(img2)
    fig, axes = plt.subplots(1, 2, figsize=(10, 5))
    axes[0].imshow(cv2.cvtColor(dest_not1, cv2.COLOR_BGR2RGB))
    axes[0].axis("off")
    axes[0].set_title("Bitwise NOT on Image 1")
    axes[1].imshow(cv2.cvtColor(dest_not2, cv2.COLOR_BGR2RGB))
    axes[1].axis("off")
    axes[1].set_title("Bitwise NOT on Image 2")
    plt.show()
```

Image Processing

Image Resizing

This code demonstrates **image resizing** using OpenCV. It loads an image, scales it to **half size**, **enlarges** it to specific dimensions, and applies **linear interpolation** for resizing. The results are displayed using Matplotlib in a 2×2 grid.

```python
import cv2
import matplotlib.pyplot as plt
image = cv2.imread("image1.png")
if image is None:
    print("Error: Could not load image. Check the file path and integrity.")
else:
    half = cv2.resize(image, (0, 0), fx=0.1, fy=0.1)
    bigger = cv2.resize(image, (1050, 1610))
    stretch_near = cv2.resize(image, (780, 540), interpolation=cv2.INTER_LINEAR)
    titles = ["Original", "Half", "Bigger", "Interpolation Linear"]
    images = [image, half, bigger, stretch_near]
    plt.figure(figsize=(6, 4))
    for i in range(4):
        plt.subplot(2, 2, i + 1)
        plt.title(titles[i])
        plt.imshow(cv2.cvtColor(images[i], cv2.COLOR_BGR2RGB))
        plt.axis("off")
    plt.tight_layout()
    plt.show()
```



Blurring an Image

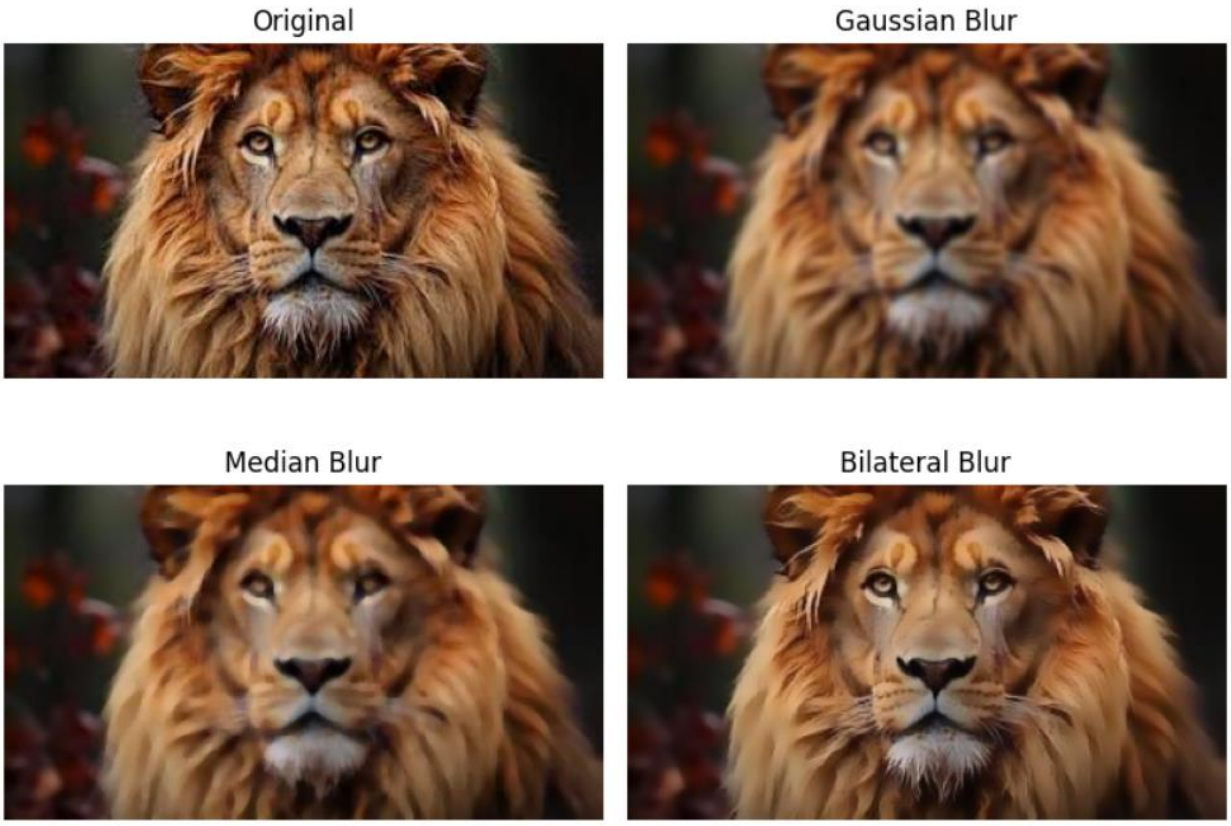
This code applies **three types of blurring** to an image using OpenCV:

1. **Gaussian Blur** – Uses a Gaussian kernel to smooth the image.
2. **Median Blur** – Replaces each pixel with the median of surrounding pixels, reducing noise.

3. **Bilateral Filter** – Preserves edges while smoothing.

The original and processed images are displayed using Matplotlib in a 2×2 grid.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image = cv2.imread('image1.png')
if image is None:
    print("Error: Could not load image. Check the file path and integrity.")
else:
    gaussian = cv2.GaussianBlur(image, (7, 7), 0)
    median = cv2.medianBlur(image, 5)
    bilateral = cv2.bilateralFilter(image, 9, 75, 75)
    titles = ["Original", "Gaussian Blur", "Median Blur", "Bilateral Blur"]
    images = [image, gaussian, median, bilateral]
    plt.figure(figsize=(8, 6))
    for i in range(4):
        plt.subplot(2, 2, i + 1)
        plt.title(titles[i])
        plt.imshow(cv2.cvtColor(images[i], cv2.COLOR_BGR2RGB))
        plt.axis("off")
    plt.tight_layout()
    plt.show()
```



Create Border around Images

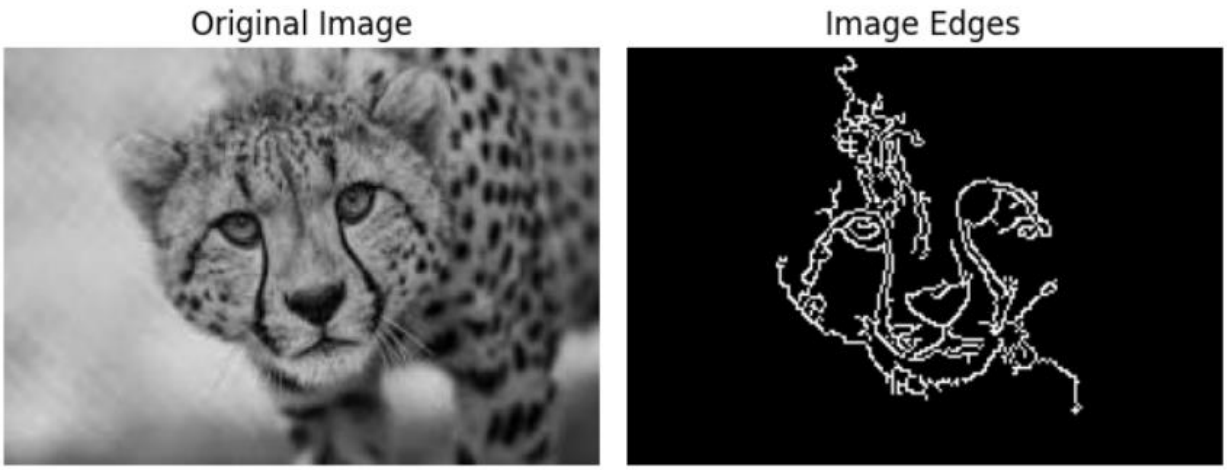
This code **resizes an image** to 25% of its original size and adds a **black border** using OpenCV. The processed image is then displayed using Matplotlib.

```python
import cv2
import matplotlib.pyplot as plt
path = "animal.png"
image = cv2.imread(path)
if image is None:
    print(f"Error: Could not load image from '{path}'. Check the file path and integrity.")
else:
    scale_percent = 25
    width = int(image.shape[1] * scale_percent / 100)
    height = int(image.shape[0] * scale_percent / 100)
    small_image = cv2.resize(image, (width, height))
    border_color = (0, 0, 0)
    border_size = 25
    image_with_border = cv2.copyMakeBorder(
        small_image, border_size, border_size, border_size, border_size,
        cv2.BORDER_CONSTANT, value=border_color
    )
    plt.figure(figsize=(6, 6))
    plt.imshow(cv2.cvtColor(image_with_border, cv2.COLOR_BGR2RGB))
    plt.axis("off")
    plt.show()
```

Scaling, Rotating, Shifting and Edge Detection

This code converts an image to **grayscale** and applies **Canny edge detection** using OpenCV. The original and edge-detected images are displayed side by side using Matplotlib.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
path = "animal.png"
img = cv2.imread(path)
if img is None:
    print("Error: Image not loaded. Check the file path.")
else:
    image_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    edges = cv2.Canny(gray_image, 100, 700)
    fig, axs = plt.subplots(1, 2, figsize=(7, 4))
    axs[0].imshow(image_rgb)
    axs[0].set_title("Original Image")
    axs[1].imshow(edges, cmap="gray")
    axs[1].set_title("Image Edges")
    for ax in axs:
        ax.axis("off")
    plt.tight_layout()
    plt.show()
```



Morphological Image Processing

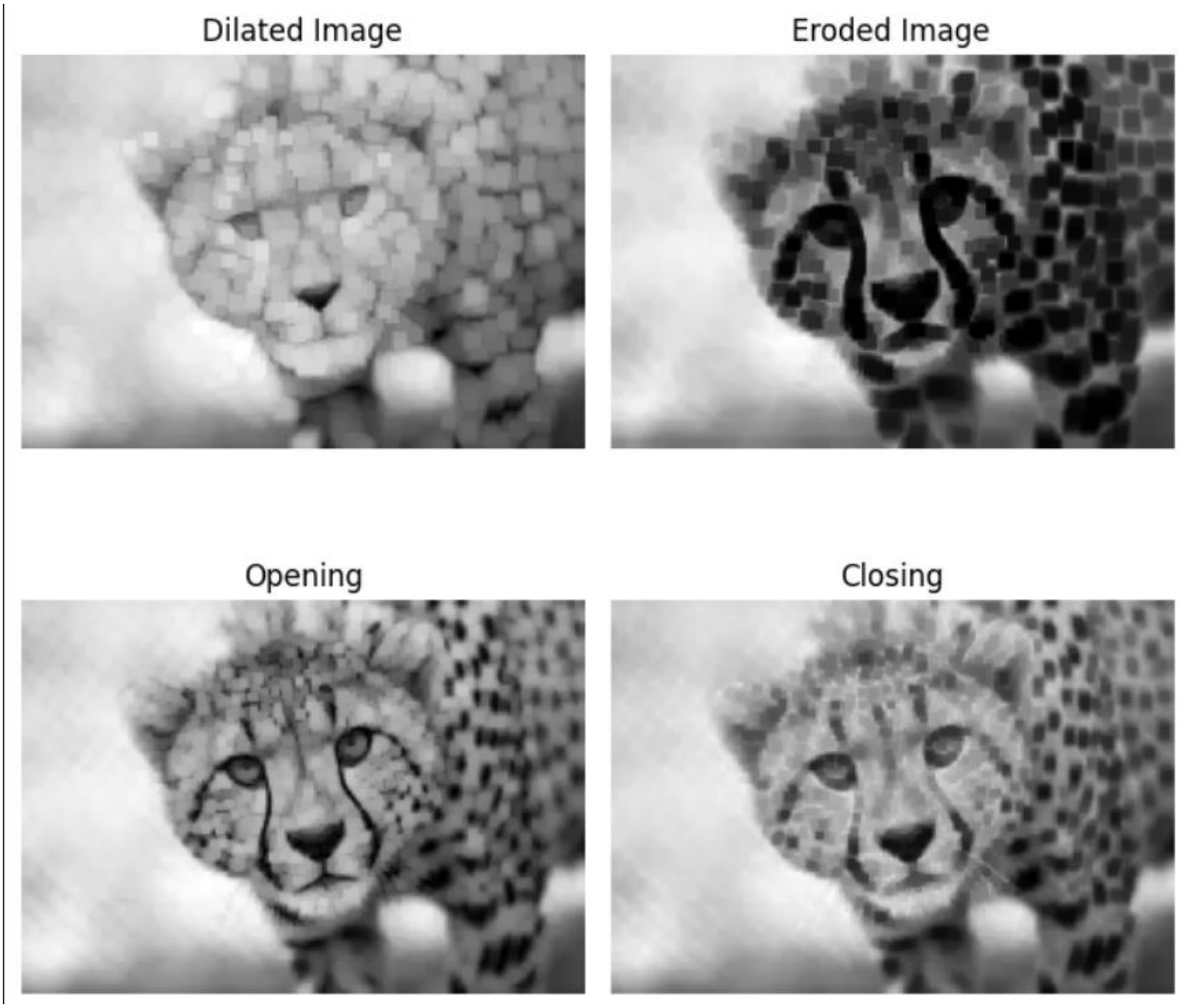
This code applies **morphological operations** on a grayscale image using OpenCV:

1. **Dilation** – Expands bright regions.

2. **Erosion** – Shrinks bright regions.
3. **Opening** – Removes small noise (erosion → dilation).
4. **Closing** – Fills small holes (dilation → erosion).

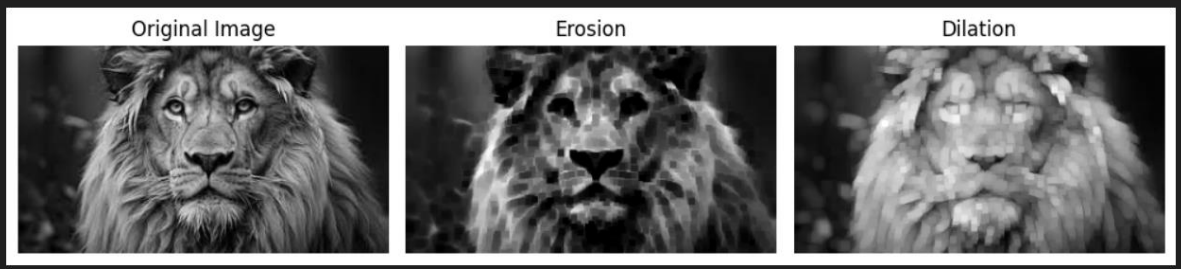
The results are displayed in a 2×2 grid using Matplotlib.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
path = "animal.png"
image = cv2.imread(path)
if image is None:
    print("Error: Image not loaded. Check the file path.")
else:
    image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    kernel = np.ones((3, 3), np.uint8)
    dilated = cv2.dilate(image_gray, kernel, iterations=2)
    eroded = cv2.erode(image_gray, kernel, iterations=2)
    opening = cv2.morphologyEx(image_gray, cv2.MORPH_OPEN, kernel)
    closing = cv2.morphologyEx(image_gray, cv2.MORPH_CLOSE, kernel)
    titles = ["Dilated Image", "Eroded Image", "Opening", "Closing"]
    images = [dilated, eroded, opening, closing]
    fig, axs = plt.subplots(2, 2, figsize=(7, 7))
    for i, ax in enumerate(axs.flat):
        ax.imshow(images[i], cmap="gray")
        ax.set_title(titles[i])
        ax.axis("off")
    plt.tight_layout()
    plt.show()
```



This code applies **erosion** and **dilation** on a grayscale image using OpenCV. **Erosion** shrinks bright areas, while **dilation** expands them. The original, eroded, and dilated images are displayed side by side using Matplotlib.

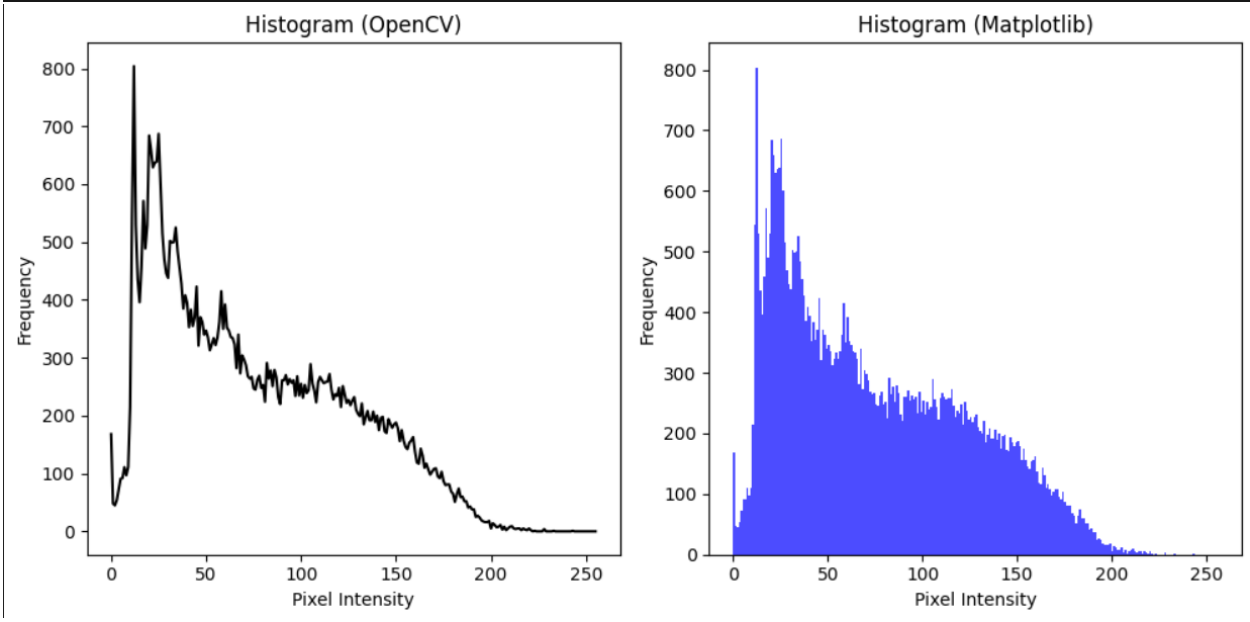
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread("image1.png", 0)
if img is None:
    print("Error: Image not loaded. Please check the file path and integrity.")
else:
    kernel = np.ones((5, 5), np.uint8)
    img_erosion = cv2.erode(img, kernel, iterations=1)
    img_dilation = cv2.dilate(img, kernel, iterations=1)
    titles = ["Original Image", "Erosion", "Dilation"]
    images = [img, img_erosion, img_dilation]
    plt.figure(figsize=(10, 4))
    for i in range(3):
        plt.subplot(1, 3, i + 1)
        plt.imshow(images[i], cmap="gray")
        plt.title(titles[i])
        plt.axis("off")
    plt.tight_layout()
    plt.show()
```



Analyze an image using Histogram

This code calculates and displays the **histogram of a grayscale image** using **OpenCV** and **Matplotlib**. The OpenCV method (cv2.calcHist) plots intensity distribution, while Matplotlib's plt.hist creates a histogram from pixel values. Both are shown side by side.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('image1.png', 0)
if img is None:
    print("Error: Image not loaded. Please check the file path and integrity.")
else:
    hist_cv = cv2.calcHist([img], [0], None, [256], [0, 256])
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.plot(hist_cv, color='black')
    plt.title('Histogram (OpenCV)')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')
    plt.subplot(1, 2, 2)
    plt.hist(img.ravel(), bins=256, range=[0, 256], color='blue', alpha=0.7)
    plt.title('Histogram (Matplotlib)')
    plt.xlabel('Pixel Intensity')
    plt.ylabel('Frequency')
    plt.tight_layout()
    plt.show()
```



Adaptive thresholding type on an image

This code applies **adaptive thresholding** to a grayscale image using OpenCV:

1. **Adaptive Mean Thresholding** – Uses the mean of neighboring pixels.
2. **Adaptive Gaussian Thresholding** – Uses a weighted sum of neighboring pixels.

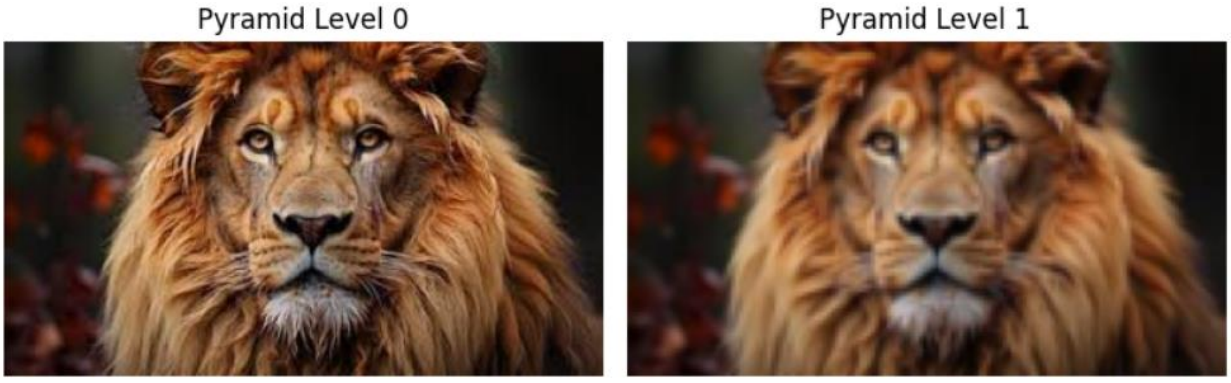
Both results are displayed side by side using Matplotlib.

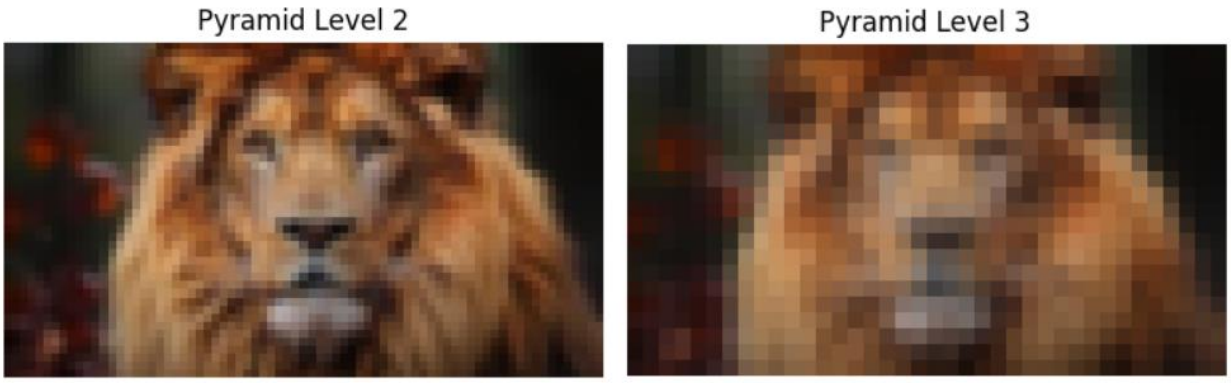
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
image1 = cv2.imread('image1.png', cv2.IMREAD_GRAYSCALE)
if image1 is None:
    print("Error: Image not loaded. Check the file path.")
else:
    thresh1 = cv2.adaptiveThreshold(image1, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
                                    cv2.THRESH_BINARY, 199, 5)
    thresh2 = cv2.adaptiveThreshold(image1, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                    cv2.THRESH_BINARY, 199, 5)
    titles = ["Adaptive Mean", "Adaptive Gaussian"]
    images = [thresh1, thresh2]
    plt.figure(figsize=(10, 5))
    for i in range(2):
        plt.subplot(1, 2, i + 1)
        plt.title(titles[i])
        plt.imshow(images[i], cmap="gray")
        plt.axis("off")
    plt.tight_layout()
    plt.show()
```

Image Pyramiding

This code generates an **image pyramid** using OpenCV's pyrDown(), which progressively reduces the image size. It creates **four pyramid levels** and displays them in a 2×2 grid using Matplotlib.

```python
import cv2
import matplotlib.pyplot as plt
img = cv2.imread('image1.png', cv2.IMREAD_COLOR)
if img is None:
    print("Error: Could not open or find the image. Please check the file path.")
else:
    layer = img.copy()
    images = [layer]
    for _ in range(3):
        layer = cv2.pyrDown(layer)
        images.append(layer.copy())
    plt.figure(figsize=(8, 8))
    for i in range(len(images)):
        plt.subplot(2, 2, i + 1)
        plt.imshow(cv2.cvtColor(images[i], cv2.COLOR_BGR2RGB))
        plt.axis("off")
        plt.title(f"Pyramid Level {i}")
    plt.tight_layout()
    plt.show()
```





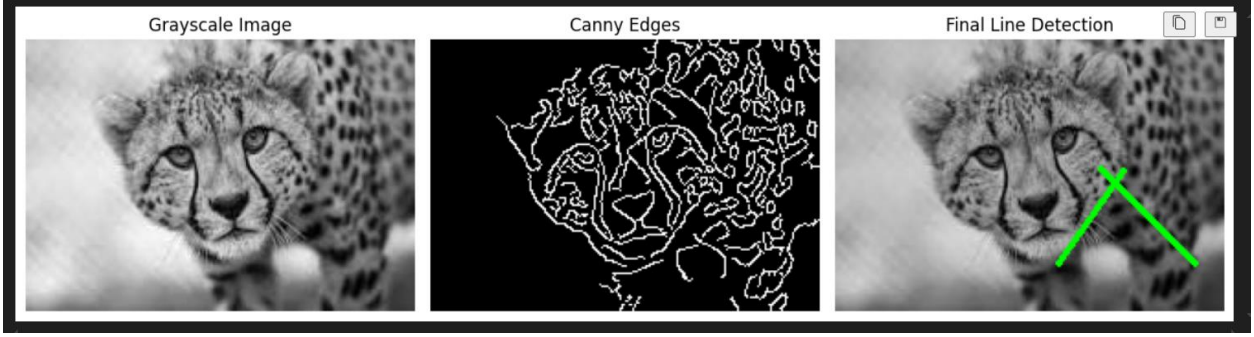
Feature Detection and Description

Line detection

This code detects **edges** using the **Canny algorithm** and identifies **straight lines** using **Hough Line Transform** in an image. The grayscale, edge-detected, and final line-detected images are displayed side by side using Matplotlib.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('animal.png', cv2.IMREAD_COLOR)
if img is None:
    print("Error: Could not load image. Please check the file path.")
else:
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 1.5)
    edges = cv2.Canny(blurred, 50, 150, apertureSize=3)
    lines = cv2.HoughLinesP(edges, 1, np.pi / 180, 50, minLineLength=50, maxLineGap=5)
    if lines is not None:
        print(f"Detected {len(lines)} lines")
        for line in lines:
            x1, y1, x2, y2 = line[0]
            cv2.line(img, (x1, y1), (x2, y2), (0, 255, 0), 2)
    else:
        print("No lines detected.")
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.figure(figsize=(12, 6))
    plt.subplot(1, 3, 1)
    plt.imshow(gray, cmap="gray")
    plt.title("Grayscale Image")
    plt.axis("off")
    plt.subplot(1, 3, 2)
    plt.imshow(edges, cmap="gray")
    plt.title("Canny Edges")
    plt.axis("off")
    plt.subplot(1, 3, 3)
    plt.imshow(img_rgb)
    plt.title("Final Line Detection")
    plt.axis("off")
    plt.tight_layout()
    plt.show()
```

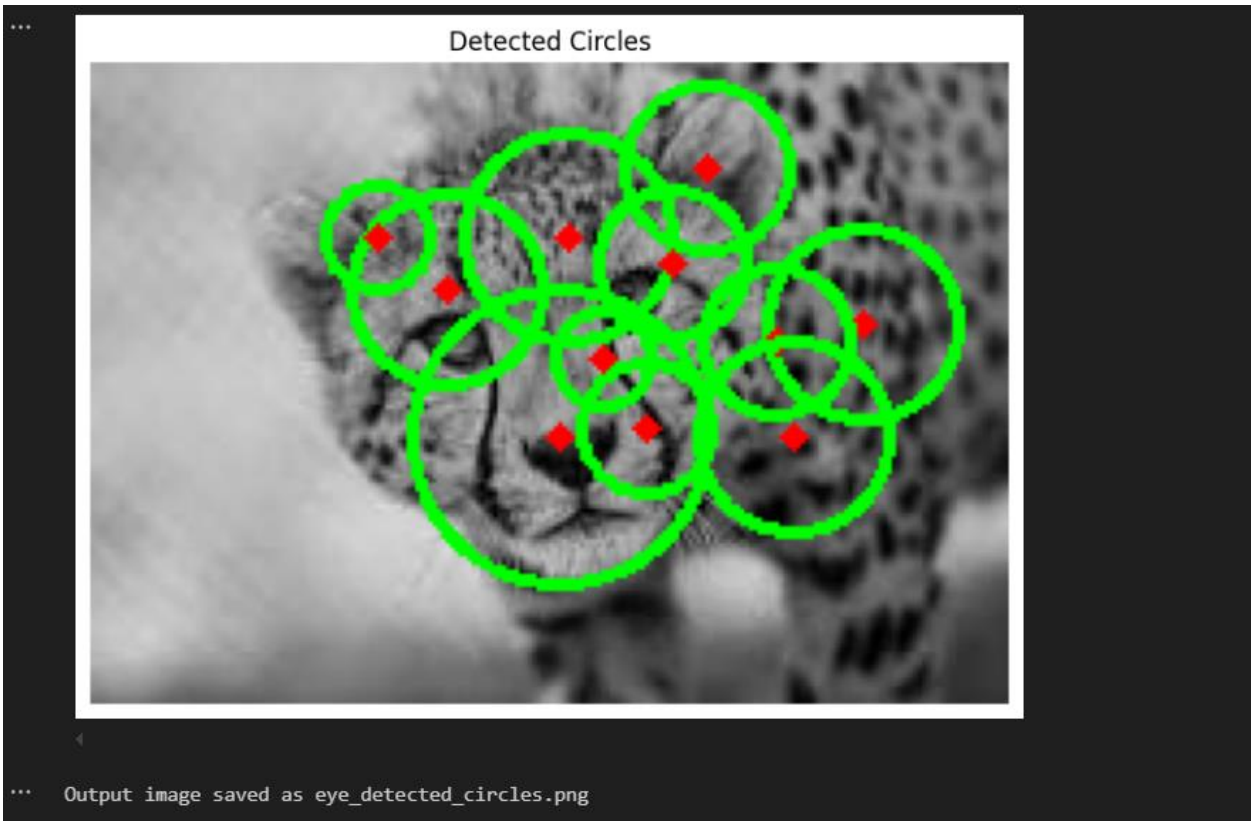
```
Detected 2 lines
```



Circle Detection

This code detects **circular objects** in an image using the **Hough Circle Transform**. It converts the image to grayscale, applies blurring, and detects circles based on edge gradients. Detected circles are highlighted in green, with centers marked in red. The result is displayed using Matplotlib and saved as an output image.

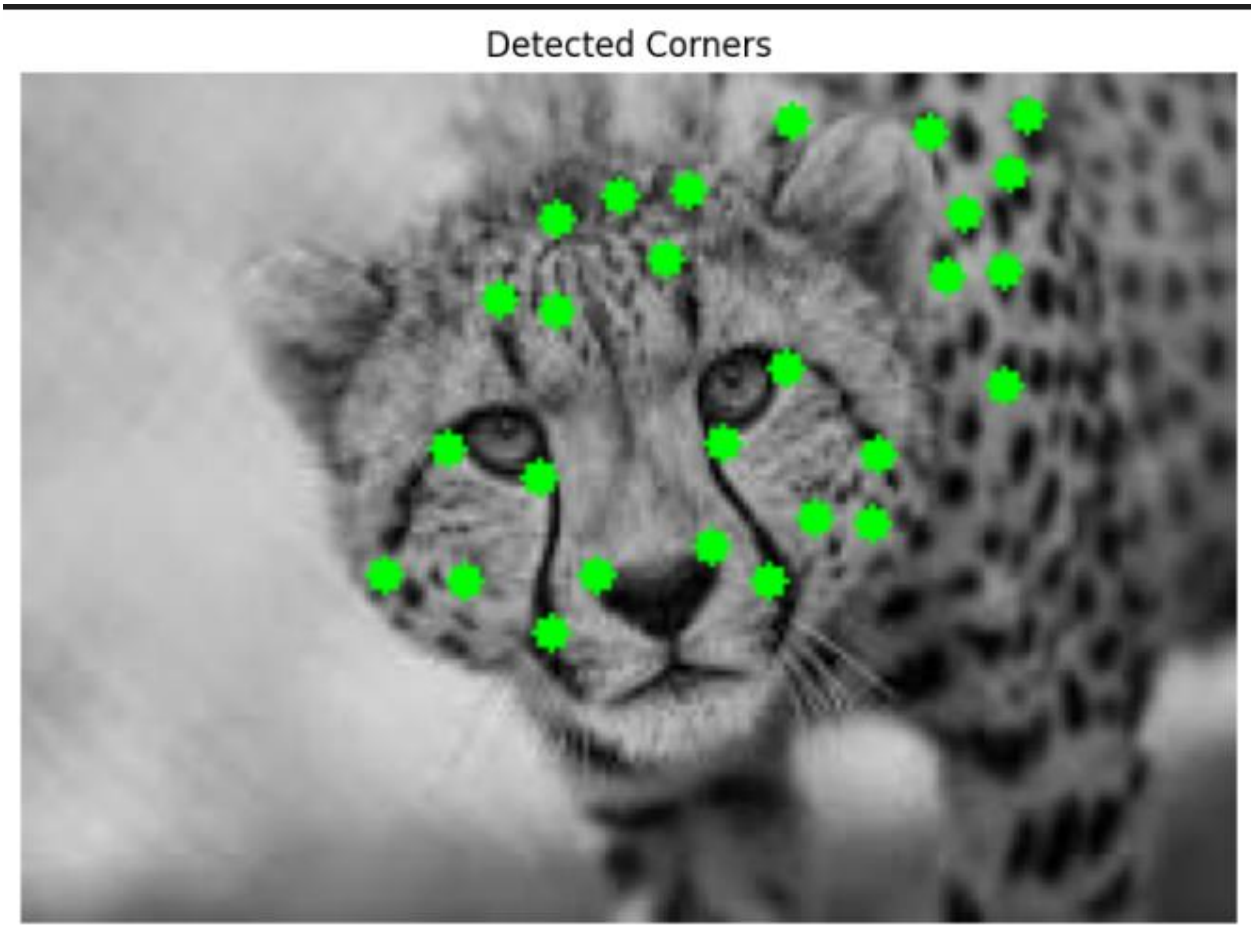
```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
image_path = 'animal.png'
img = cv2.imread(image_path, cv2.IMREAD_COLOR)
if img is None:
    print("Error: Could not load image. Please check the file path.")
else:
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gray_blurred = cv2.blur(gray, (3, 3))

    detected_circles = cv2.HoughCircles(
        gray_blurred,
        cv2.HOUGH_GRADIENT,
        dp=1,
        minDist=20,
        param1=50,
        param2=30,
        minRadius=1,
        maxRadius=40
    )
    if detected_circles is not None:
        detected_circles = np.uint16(np.around(detected_circles))
        for pt in detected_circles[0, :]:
            a, b, r = pt[0], pt[1], pt[2]
            cv2.circle(img, (a, b), r, (0, 255, 0), 2)
            cv2.circle(img, (a, b), 1, (0, 0, 255), 3)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.figure(figsize=(8, 6))
    plt.imshow(img_rgb)
    plt.axis("off")
    plt.title("Detected Circles")
    plt.show()
    output_path = 'eye_detected_circles.png'
    cv2.imwrite(output_path, img)
    print(f"Output image saved as {output_path}")
```



Detect corner of an image

This code detects **corners** in an image using the **Shi-Tomasi corner detection** method. It converts the image to grayscale, identifies key corners, and marks them with green circles. The result is displayed using Matplotlib.

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

image_path = 'animal.png'
img = cv2.imread(image_path)

if img is None:
    print("Error: Could not load image. Please check the file path.")
else:
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    corners = cv2.goodFeaturesToTrack(gray, maxCorners=27, qualityLevel=0.01, minDistance=10)

    if corners is not None:
        corners = np.int32(corners)
        for i in corners:
            x, y = i.ravel()
            cv2.circle(img, (x, y), 3, (0, 255, 0), -1)

    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    plt.figure(figsize=(8, 6))
    plt.imshow(img_rgb)
    plt.title("Detected Corners")
    plt.axis('off')
    plt.show()
```



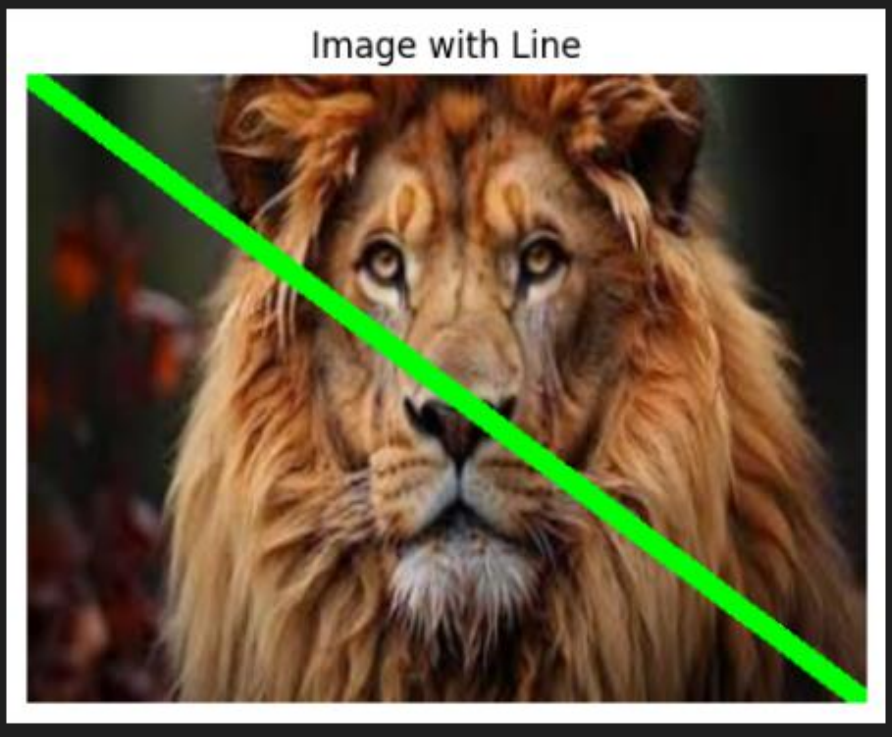
Drawing Functions

Draw a line

This code **resizes an image** to **400x300 pixels**, draws a **green diagonal line**, and displays the result using Matplotlib.

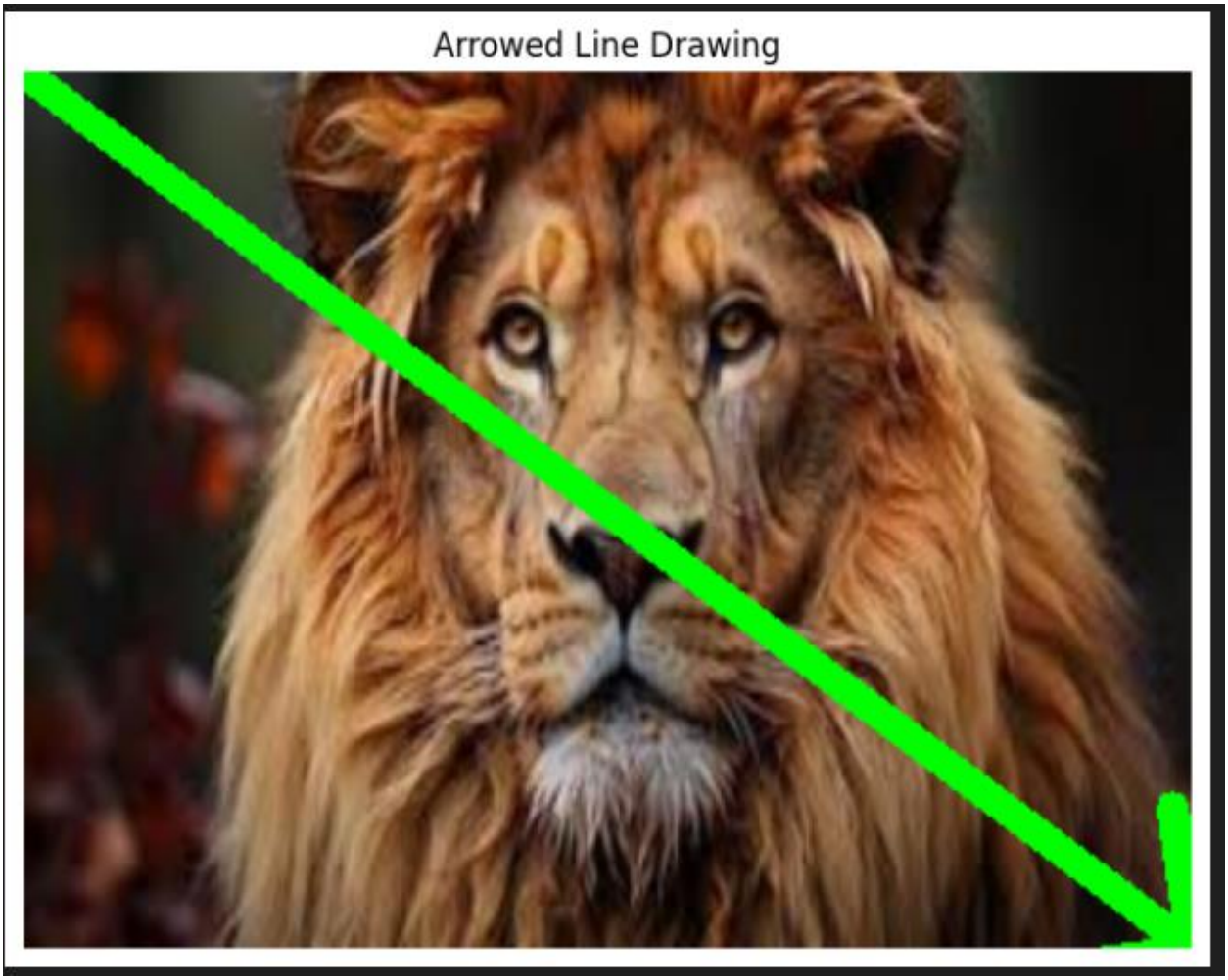
```python
import cv2
import matplotlib.pyplot as plt
image_path = 'image1.png'
image = cv2.imread(image_path)
if image is None:
    print("Error: Could not load image. Please check the file path.")
else:
    height, width, _ = image.shape
    print("Original image shape (height, width):", height, width)
    new_width = 400
    new_height = 300
    resized_image = cv2.resize(image, (new_width, new_height))
    print("Resized image shape (height, width):", new_height, new_width)
    start_point = (0, 0)
    end_point = (new_width, new_height)
    color = (0, 255, 0)  # Green color in BGR
    thickness = 9
    resized_image = cv2.line(resized_image, start_point, end_point, color, thickness)
    img_rgb = cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB)
    plt.figure(figsize=(6, 4))
    plt.imshow(img_rgb)
    plt.title("Image with Line")
    plt.axis('off')
    plt.show()
```

```
Original image shape (height, width): 168 300
Resized image shape (height, width): 300 400
```



This code **resizes an image** to **400x300 pixels**, draws a **green arrowed line** from the top-left to the bottom-right corner, and displays the result using Matplotlib.

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
image_path = 'image1.png'
image = cv2.imread(image_path)
if image is None:
    print("Error: Could not load image. Please check the file path.")
else:
    new_width = 400
    new_height = 300
    resized_image = cv2.resize(image, (new_width, new_height))
    start_point = (0, 0)
    end_point = (new_width, new_height)
    color = (0, 255, 0)
    thickness = 9
    resized_image = cv2.arrowedLine(resized_image, start_point, end_point, color, thickness)
    img_rgb = cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB)
    plt.figure(figsize=(8, 6))
    plt.imshow(img_rgb)
    plt.title("Arrowed Line Drawing")
    plt.axis('off')
    plt.show()
```


Arrowed Line Drawing