

## 1. Install OpenCV

```
!pip install opencv-python-headless
```

```
→ Requirement already satisfied: opencv-python-headless in /usr/local/lib/python3.10/dist-packages (4.10.0.84)
Requirement already satisfied: numpy>=1.21.2 in /usr/local/lib/python3.10/dist-packages (from opencv-python-headless) (1.26.4)
```

## 2. Import Libraries

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

def display_image(img, title="Image"):
    plt.figure(figsize=(8,8))

    plt.imshow(cv2.cvtColor(img,
cv2.COLOR_BGR2RGB))    plt.title(title)
    plt.axis('off')    plt.show()

def display_images(original, processed, title1, title2):
    plt.figure(figsize=(10,5))
    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(original,
cv2.COLOR_BGR2RGB))    plt.title(title1)
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(processed, cv2.COLOR_BGR2RGB))
    plt.title(title2)    plt.axis('off')
    plt.show()
```

## 3. Upload Image

```
from google.colab import drive
drive.mount('/content/drive')
```

```
→ Mounted at /content/drive
```

```
uploaded = '/content/drive/MyDrive/exer1_img.jpg'
image = cv2.imread(uploaded)
display_image(image, "Original Image")
```

Original Image



```
# from google.colab import files
```

```
# from io import BytesIO
# from PIL import Image
```

```
# #CONVERT TO OPENCV FORMAT
# image_path = next(iter(uploaded))
# image = Image.open(BytesIO(uploaded[image_path]))
# image = cv2.cvtColor(np.array(image), cv2.COLOR_RGB2BGR)

# display_image(image, "Original Image")
```

EXERCISE 1. Scaling and Rotation

```
def scale_image(image, scale_factor):    height, width =
image.shape[:2]    scale_img = cv2.resize(image,
(int(width*scale_factor),
int(height*scale_factor)),
interpolation=cv2.INTER_LINEAR)    return scale_img
```

```
scaled_image = scale_image(image, 0.5)
display_image(scaled_image, "Scaled Image")

def rotate_image(image, angle):
    height, width =
image.shape[:2]    center = (width//2, height//2)    matrix =
cv2.getRotationMatrix2D(center, angle, 1)    rotated_img =
cv2.warpAffine(image, matrix, (width, height))    return
rotated_img

rotated_image = rotate_image(image, 45)
display_image(rotated_image, "Rotated Image")
```



Scaled Image



Rotated Image



#### Exercise 2: Blurring Techniques

```
gaussian.blur = cv2.GaussianBlur(image, (15,15), 0)
display_image(gaussian.blur, "Gussian Blur")

median.blur = cv2.medianBlur(image, 15)
display_image(median.blur, "Median Blur")

bilateral.blur = cv2.bilateralFilter(image, 51, 15, 15)
display_image(bilateral.blur, "Bilateral Blur")
```



Gussain Blur



Median Blur



Bilateral Blur



3. Edge Detection using Canny

```
edge = cv2.Canny(image, 130, 130)
```

```
display_image(edge, "Edge Detection")
```



#### Exercise 4: Basic Image Processor (Interactive)

```
def process_image(img, action):
    if action == 'scale':
        return scale_image(img, 0.5)
    elif action == 'rotate':
        return rotate_image(img, 45)
    elif action == 'gaussian_blur':
        return cv2.GaussianBlur(img, (5, 5), 0)
    elif action == 'median_blur':
        return cv2.medianBlur(img, 5)
    elif action == 'canny':
        return cv2.Canny(img, 100, 200)
    else:
        return img
"""

process_image(): This function allows users to specify an image transformation (scaling, rotation, blurring, or edge detection). Depending on the action passed
"""
    action = input("Enter action (scale, rotate, gaussian blur, median blur, canny): ")
    processed_image = process_image(image, action)
    display_images(image, processed_image, "Original Image", f"Processed Image ({action})")
    """
This allows users to enter their desired transformation interactively (via the input() function). It processes the image and displays both the original and transformed versions side by side.
"""


```

→ Enter action (scale, rotate, gaussian blur, median blur, canny): canny

Original Image



Processed Image (canny)



#### Exercise 5: Comparison of Filtering Techniques

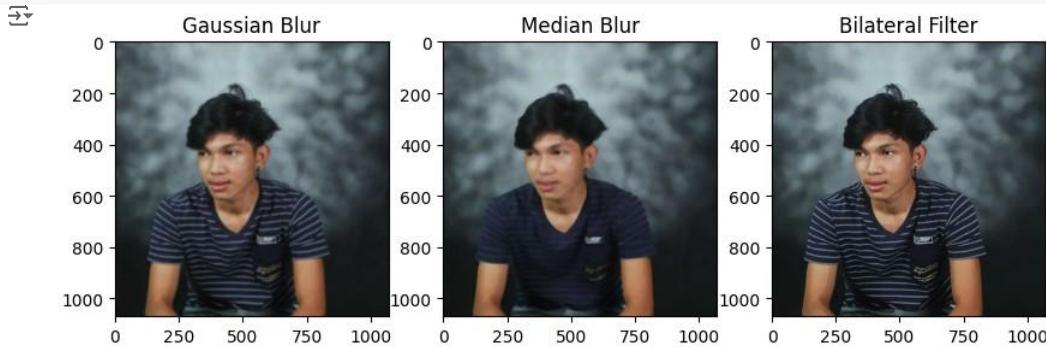
```
# Applying Gaussian, Median, and Bilateral filters
gaussian_blur = cv2.GaussianBlur(image, (15,15), 0)
median_blur = cv2.medianBlur(image, 15)
bilateral_filter = cv2.bilateralFilter(image, 51, 15,
15)
"""

cv2.bilateralFilter(): This filter smooths the image while keeping edges sharp, unlike Gaussian or median filters. It's useful for reducing noise while preserving details.
"""
# Display the results for comparison
plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(gaussian_blur,
cv2.COLOR_BGR2RGB))
plt.title("Gaussian Blur")
plt.subplot(1, 3, 2)
plt.imshow(cv2.cvtColor(median_blur, cv2.COLOR_BGR2RGB))
plt.title("Median Blur")
plt.subplot(1, 3, 3)
```

```

plt.imshow(cv2.cvtColor(bilateral_filter,
cv2.COLOR_BGR2RGB)) plt.title("Bilateral Filter")
plt.show() """
Explanation: This displays the images processed by different filtering techniques (Gaussian,
Median, and Bilateral) side by side for comparison.
"""

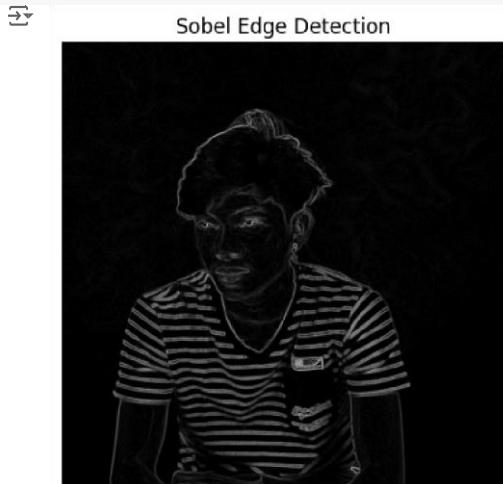
```



```

# Sobel Edge Detection def
sobel_edge_detection(img): # Convert to
grayscale gray = cv2.cvtColor(img,
cv2.COLOR_BGR2GRAY) # Sobel edge detection
in the x direction
sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=7)
# Sobel edge detection in the y direction
sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=7)
# Combine the two gradients
sobel_combined = cv2.magnitude(sobelx, sobely)
return sobel_combined
# Apply Sobel edge detection to the uploaded
image sobel_edges = sobel_edge_detection(image)
plt.imshow(sobel_edges, cmap='gray')
plt.title("Sobel Edge Detection")
plt.axis('off') plt.show()

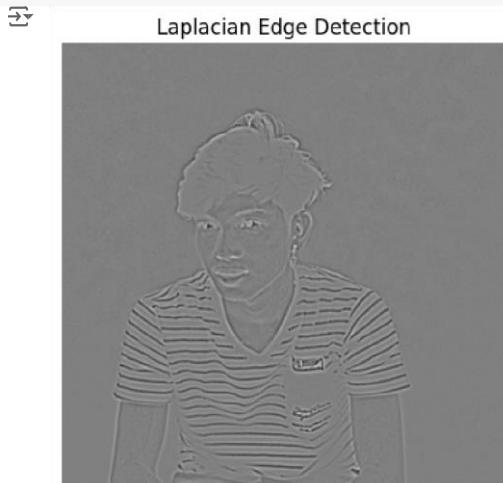
```



```

# Laplacian Edge Detection def
laplacian_edge_detection(img):
    # Convert to grayscale gray =
cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Apply Laplacian operator laplacian =
cv2.Laplacian(gray, cv2.CV_64F) return
laplacian
# Apply Laplacian edge detection to the uploaded image
laplacian_edges =
laplacian_edge_detection(gaussian_blur)
plt.imshow(laplacian_edges, cmap='gray')
plt.title("Laplacian Edge Detection") plt.axis('off')
plt.show()

```



```

# Prewitt Edge Detection def
prewitt_edge_detection(img):
    # Convert to grayscale gray = cv2.cvtColor(img,
cv2.COLOR_BGR2GRAY) # Prewitt operator kernels for x and y
directions kernelx = np.array([[1, 0, -1], [1, 0, -1], [1, 0, -1]], dtype=int)

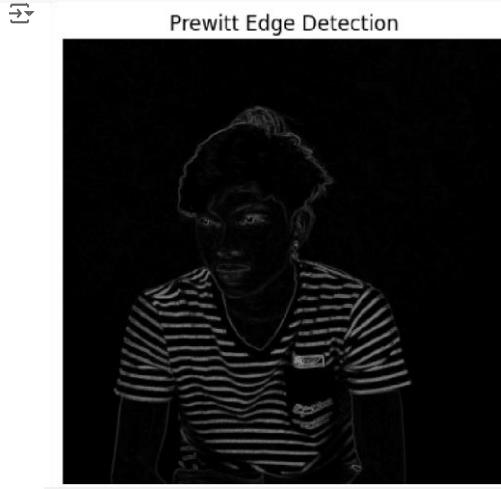
```

```

kernelx = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]], dtype=int)
# Applying the Prewitt operator  prewittx =
cv2.filter2D(gray, cv2.CV_64F, kernelx)  prewitty
= cv2.filter2D(gray, cv2.CV_64F, kernely)
# Combine the x and y gradients by converting to floating
point  prewitt_combined = cv2.magnitude(prewittx, prewitty)
return prewitt_combined

# Apply Prewitt edge detection to the uploaded
image prewitt_edges =
prewitt_edge_detection(image)
plt.imshow(prewitt_edges, cmap='gray')
plt.title("Prewitt Edge Detection")
plt.axis('off') plt.show()

```



```

# Bilateral Filter def
bilateral_blur(img):
    bilateral = cv2.bilateralFilter(img, 9, 75, 75)
return bilateral

# Apply Bilateral filter to the uploaded image
bilateral_blurred = bilateral_blur(image)
plt.imshow(cv2.cvtColor(bilateral_blurred,
cv2.COLOR_BGR2RGB)) plt.title("Bilateral Filter")
plt.axis('off') plt.show()

```



```

# Box Filter def
box.blur(img):
    box = cv2.boxFilter(img, -1, (5, 5))
return box

# Apply Box filter to the uploaded image
box_blurred = box.blur(image)
plt.imshow(cv2.cvtColor(box_blurred,
cv2.COLOR_BGR2RGB)) plt.title("Box Filter")
plt.axis('off') plt.show()

```



```
# Motion Blur def
motion_blur(img):
    # Create motion blur kernel (size 15x15)
    kernel_size = 15    kernel =
    np.zeros((kernel_size, kernel_size))

    kernel[int((kernel_size - 1) / 2), :] = np.ones(kernel_size)
    kernel = kernel / kernel_size
    # Apply motion blur    motion_blurred =
    cv2.filter2D(img, -1, kernel)    return
motion_blurred
# Apply Motion blur to the uploaded image motion_blurred
= motion_blur(image)
plt.imshow(cv2.cvtColor(motion_blurred,
cv2.COLOR_BGR2RGB)) plt.title("Motion Blur")
plt.axis('off') plt.show()
```



Motion Blur



```
# Unsharp Masking (Sharpening)
def unsharp_mask(img):
    # Create a Gaussian blur version of the image
blurred = cv2.GaussianBlur(img, (9, 9), 10.0)
    # Sharpen by adding the difference between the original and the blurred
image    sharpened = cv2.addWeighted(img, 1.5, blurred, -0.5, 0)    return
sharpened
# Apply Unsharp Masking to the uploaded image
sharpened_image = unsharp_mask(image)
plt.imshow(cv2.cvtColor(sharpened_image,
cv2.COLOR_BGR2RGB)) plt.title("Unsharp Mask (Sharpening)")
plt.axis('off') plt.show()
```



Unsharp Mask (Sharpening)



```

# Update process_image function to include new blurring
techniques def process_image(img, action):
    if action == 'scale':
        return scale_image(img, 0.5)
    elif action == 'rotate':
        return rotate_image(img, 45)
    elif action == 'gaussian_blur':
        return cv2.GaussianBlur(img, (5, 5), 0)
    elif action == 'median_blur':
        return cv2.medianBlur(img, 5)
    elif action == 'canny':
        return cv2.Canny(img, 100, 200)
    elif action == 'sobel':
        return sobel_edge_detection(img).astype(np.uint8)
    elif action == 'laplacian':
        return laplacian_edge_detection(img).astype(np.uint8)
    elif action == 'prewitt':
        return prewitt_edge_detection(img).astype(np.uint8)
    elif action == 'bilateral_blur':
        return bilateral_blur(img)
    elif action == 'box_blur':
        return box_blur(img)
    elif action == 'motion_blur':
        return motion_blur(img)
    elif action == 'unsharp_mask':
        return unsharp_mask(img)
    else:
        return img
# Add new blurring options for interactive processing
action = input("Enter action (scale, rotate, gaussian_blur, median_blur, canny, sobel, laplacian, prewitt, bilateral_blur, box_blur, motion_blur,unsharp_mas processed_image = process_image(image, action) display_images(image, processed_image, "Original Image", f"Processed Image ({action})")")

```



```

import matplotlib.pyplot as plt
import cv2

# Display Original Image plt.figure(figsize=(5, 5))
plt.suptitle("Original Image", fontweight='bold',
fontsize=14) plt.imshow(cv2.cvtColor(image,
cv2.COLOR_BGR2RGB)) plt.axis('off') plt.show()

# Display Blurring Techniques plt.figure(figsize=(10, 10))
plt.suptitle("Blurring Techniques", fontweight='bold',
fontsize=14)

plt.subplot(2, 3, 1)
plt.imshow(cv2.cvtColor(gaussian_blur,
cv2.COLOR_BGR2RGB)) plt.title("Gaussian Blur",
fontsize=10) plt.axis('off')

plt.subplot(2, 3, 2)
plt.imshow(cv2.cvtColor(median.blur,
cv2.COLOR_BGR2RGB)) plt.title("Median Blur",
fontsize=10) plt.axis('off')

plt.subplot(2, 3, 3)
plt.imshow(cv2.cvtColor(bilateral_filter,
cv2.COLOR_BGR2RGB)) plt.title("Bilateral Filter",
fontsize=10) plt.axis('off')

plt.subplot(2, 3, 4)
plt.imshow(cv2.cvtColor(motion_blurred,
cv2.COLOR_BGR2RGB)) plt.title("Motion Blur", fontsize=10)
plt.axis('off')

plt.subplot(2, 3, 5)
plt.imshow(cv2.cvtColor(box_blurred,
cv2.COLOR_BGR2RGB)) plt.title("Box Filter",
fontsize=10) plt.axis('off')

plt.subplot(2, 3, 6)
plt.imshow(cv2.cvtColor(sharpened_image,
cv2.COLOR_BGR2RGB)) plt.title("Unsharp Mask (Sharpening)",
fontsize=10) plt.axis('off')

plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()

# Display Edge Detection Techniques plt.figure(figsize=(10, 10))
plt.suptitle("Edge Detection Techniques", fontweight='bold',
fontsize=14)

plt.subplot(3, 2, 1) plt.imshow(prewitt.edges,
cmap='gray') plt.title("Prewitt Edge
Detection", fontsize=10) plt.axis('off')

plt.subplot(3, 2, 2) plt.imshow(laplacian.edges,
cmap='gray') plt.title("Laplacian Edge
Detection", fontsize=10) plt.axis('off')

plt.subplot(3, 2, 3) plt.imshow(sobel.edges,
cmap='gray') plt.title("Sobel Edge
Detection", fontsize=10) plt.axis('off')

plt.subplot(3, 2, 4) plt.imshow(canny.edges,
cmap='gray') plt.title("Canny Edge
Detection", fontsize=10) plt.axis('off')

plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()

```



**Original Image**



**Blurring Techniques**

Gaussian Blur



Median Blur



Bilateral Filter



Motion Blur



Box Filter



Unsharp Mask (Sharpening)



**Edge Detection Techniques**

Prewitt Edge Detection



Laplacian Edge Detection



## BLURRING TECHNIQUES

| <i>Filter</i>           | <i>Blurring</i> | <i>Noise Reduction</i> | <i>Edge Preservation</i> | <i>Sharpening</i> | <i>Artistic Effect</i> | <i>Facial Enhancement</i> |
|-------------------------|-----------------|------------------------|--------------------------|-------------------|------------------------|---------------------------|
| <i>Gaussian Blur</i>    | ✓               | ✓                      | ✗                        | ✗                 | ✗                      | ✗                         |
| <i>Median Blur</i>      | ✓               | ✓                      | ✓                        | ✗                 | ✗                      | ✗                         |
| <i>Bilateral Filter</i> | ✓               | ✓                      | ✓                        | ✗                 | ✗                      | ✓                         |
| <i>Motion Blur</i>      | ✓               | ✗                      | ✗                        | ✗                 | ✓                      | ✗                         |
| <i>Box Filter</i>       | ✓               | ✓                      | ✗                        | ✗                 | ✗                      | ✗                         |
| <i>Unsharp Mask</i>     | ✗               | ✗                      | ✗                        | ✓                 | ✓                      | ✓                         |

## EDGE DETECTION

| <i>Edge Detection Technique</i> | <i>Sensitivity to Noise</i> | <i>Edge Thinnness</i> | <i>Edge Continuity</i> | <i>Computational Efficiency</i> |
|---------------------------------|-----------------------------|-----------------------|------------------------|---------------------------------|
| <i>Sobel</i>                    | Moderate                    | Moderate              | Moderate               | High                            |
| <i>Laplacian</i>                | High                        | Moderate              | Moderate               | High                            |
| <i>Prewitt</i>                  | Moderate                    | Moderate              | Moderate               | High                            |
| <i>Canny</i>                    | Low                         | High                  | High                   | Moderate                        |