```
# Step 1: Import required libraries
import cv2
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path)

# Step 5: Convert image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 6: Display original and grayscale images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Grayscale Image")
plt.imshow(gray_image, cmap="gray")
plt.axis("off")

plt.show()
```

Choose Files aishu photo.avif

aishu photo.avif(image/avif) - 13376 bytes, last modified: 12/16/2025 - 100% done

```
Saving aishu photo.avif to aishu photo.avif
```


```
# Step 1: Import required libraries
import cv2
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path)

# Step 5: Apply Gaussian Blur
# Kernel size must be odd numbers (e.g., 5x5)
blur_image = cv2.GaussianBlur(image, (5, 5), 0)

# Step 6: Display original and blurred images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Image")
```

```
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Gaussian Blurred Image")
plt.imshow(cv2.cvtColor(blur_image, cv2.COLOR_BGR2RGB))
plt.axis("off")

plt.show()
```

Choose Files aishu photo.avif

aishu photo.avif(image/avif) - 13376 bytes, last modified: 12/16/2025 - 100% done

```
Saving aishu photo.avif to aishu photo (1).avif
```


```
# Step 1: Import required libraries
import cv2
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path)

# Step 5: Convert image to grayscale (Canny requires grayscale)
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 6: Apply Canny edge detection
# Threshold values can be adjusted
edges = cv2.Canny(gray_image, threshold1=100, threshold2=200)

# Step 7: Display original and edge-detected images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Canny Edge Detection (Outline)")
plt.imshow(edges, cmap="gray")
plt.axis("off")

plt.show()
```

Choose Files aishu photo.avif

aishu photo.avif(image/avif) - 13376 bytes, last modified: 12/16/2025 - 100% done

```
Saving aishu photo.avif to aishu photo (2).avif
```


```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path)

# Step 5: Convert image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 6: Create a kernel (structuring element)
kernel = np.ones((5, 5), np.uint8)

# Step 7: Apply dilation
dilated_image = cv2.dilate(gray_image, kernel, iterations=1)

# Step 8: Display original and dilated images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Grayscale Image")
plt.imshow(gray_image, cmap="gray")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Dilated Image")
plt.imshow(dilated_image, cmap="gray")
plt.axis("off")

plt.show()
```

Choose Files aishu photo.avif

aishu photo.avif(image/avif) - 13376 bytes, last modified: 12/16/2025 - 100% done

```
Saving aishu photo.avif to aishu photo (3).avif
```

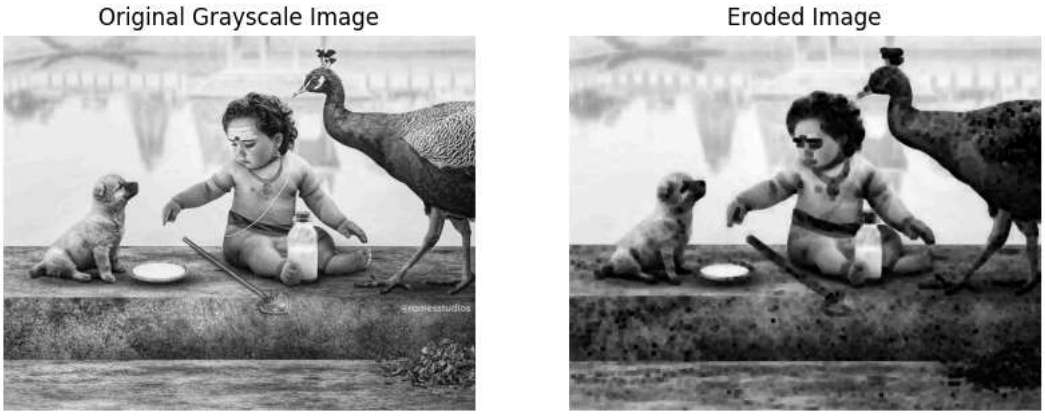

```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
```

```
# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path)

# Step 5: Convert image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 6: Create a kernel (structuring element)
kernel = np.ones((5, 5), np.uint8)

# Step 7: Apply erosion
eroded_image = cv2.erode(gray_image, kernel, iterations=1)

# Step 8: Display original and eroded images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Grayscale Image")
plt.imshow(gray_image, cmap="gray")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Eroded Image")
plt.imshow(eroded_image, cmap="gray")
plt.axis("off")

plt.show()
```

Choose Files murugaa.jpeg

murugaa.jpeg(image/jpeg) - 90744 bytes, last modified: 12/18/2025 - 100% done

```
Saving murugaa.jpeg to murugaa.jpeg
```



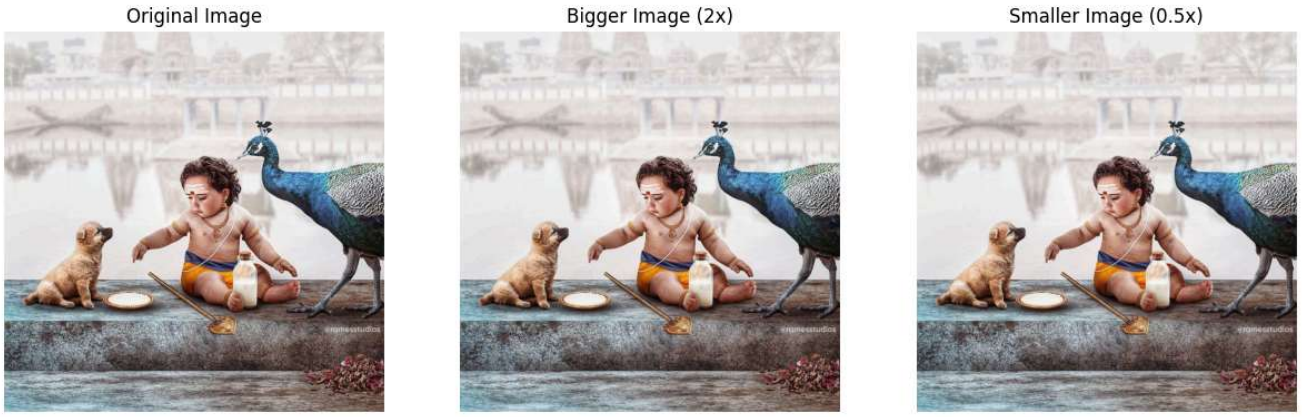

```
# Step 1: Import required libraries
import cv2
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path)

# Step 5: Convert BGR to RGB for display
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Step 6: Get original image dimensions
height, width, _ = image.shape

# Step 7: Scale image to bigger size (2x)
bigger_image = cv2.resize(
    image_rgb,
    (width * 2, height * 2),
    interpolation=cv2.INTER_LINEAR
)
```

```
)

# Step 8: Scale image to smaller size (0.5x)
smaller_image = cv2.resize(
    image_rgb,
    (width // 2, height // 2),
    interpolation=cv2.INTER_AREA
)

# Step 9: Display images
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.title("Original Image")
plt.imshow(image_rgb)
plt.axis("off")

plt.subplot(1, 3, 2)
plt.title("Bigger Image (2x)")
plt.imshow(bigger_image)
plt.axis("off")

plt.subplot(1, 3, 3)
plt.title("Smaller Image (0.5x)")
plt.imshow(smaller_image)
plt.axis("off")

plt.show()
```

Choose Files No Photo D…allpaper.jpg

No Photo Description Available Lord Murugan Wallpaper.jpg(image/jpeg) - 227563 bytes, last modified: 12/18/2025 - 100% done

```
Saving No Photo Description Available Lord Murugan Wallpaper.jpg to No Photo Description Available Lord Murugan Wallpaper.jpg
```



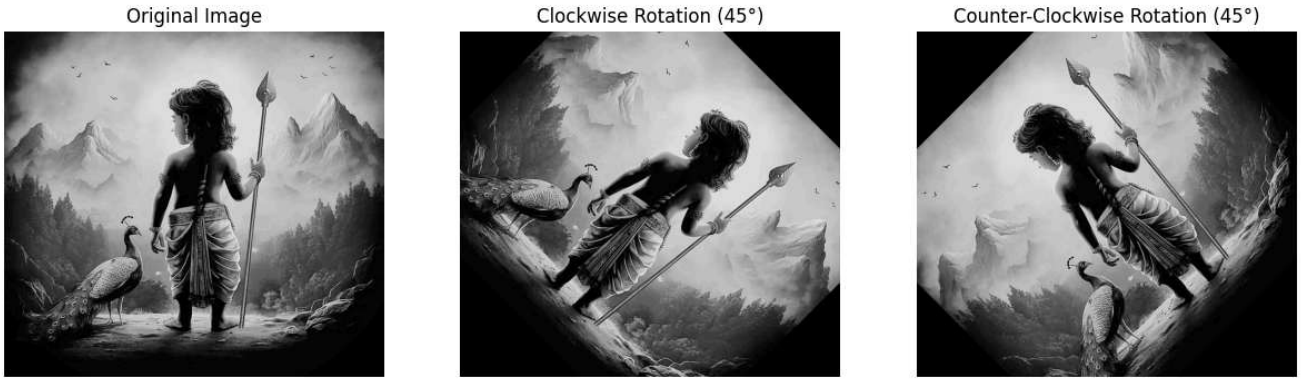

```
# Step 1: Import required libraries
import cv2
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path)

# Step 5: Convert BGR to RGB for display
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Step 6: Get image center
(h, w) = image_rgb.shape[:2]
center = (w // 2, h // 2)

# Step 7: Rotate image clockwise (-45 degrees)
clockwise_matrix = cv2.getRotationMatrix2D(center, -45, 1.0)
```

```
clockwise_image = cv2.warpAffine(image_rgb, clockwise_matrix, (w, h))

# Step 8: Rotate image counter-clockwise (+45 degrees)
counter_matrix = cv2.getRotationMatrix2D(center, 45, 1.0)
counter_image = cv2.warpAffine(image_rgb, counter_matrix, (w, h))

# Step 9: Display images
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.title("Original Image")
plt.imshow(image_rgb)
plt.axis("off")

plt.subplot(1, 3, 2)
plt.title("Clockwise Rotation (45°)")
plt.imshow(clockwise_image)
plt.axis("off")

plt.subplot(1, 3, 3)
plt.title("Counter-Clockwise Rotation (45°)")
plt.imshow(counter_image)
plt.axis("off")

plt.show()
```

Choose Files IMG_1156.JPG

IMG_1156.JPG(image/jpeg) - 1280019 bytes, last modified: 12/18/2025 - 100% done

```
Saving IMG_1156.JPG to IMG_1156.JPG
```



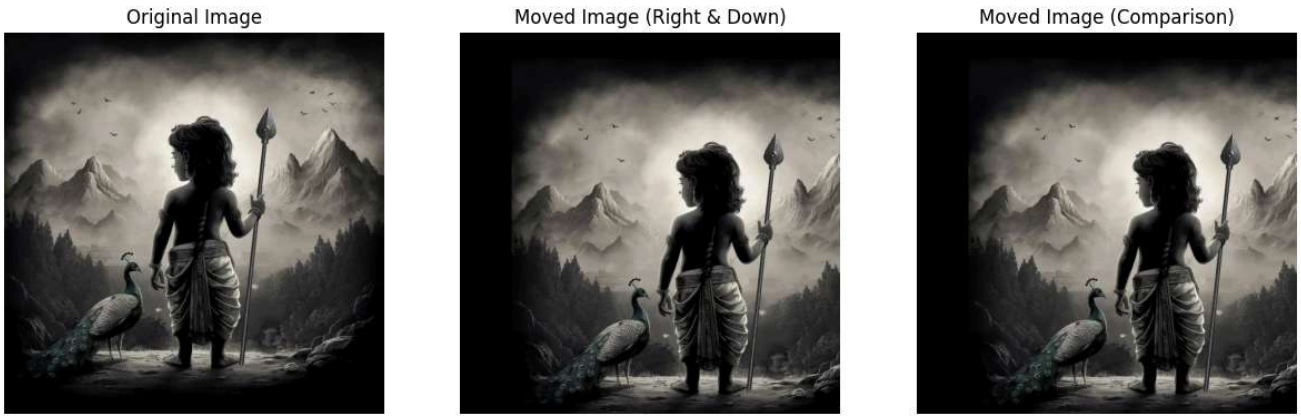

```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path)

# Step 5: Convert BGR to RGB for display
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Step 6: Define translation values
# tx = movement in x-direction (right)
# ty = movement in y-direction (down)
tx = 100
ty = 50

# Step 7: Create translation matrix
translation_matrix = np.float32([[1, 0, tx],
                                 [0, 1, ty]])
```

```
# Step 8: Apply translation
height, width = image_rgb.shape[:2]
moved_image = cv2.warpAffine(image_rgb, translation_matrix, (width, height))

# Step 9: Display images
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.title("Original Image")
plt.imshow(image_rgb)
plt.axis("off")

plt.subplot(1, 3, 2)
plt.title("Moved Image (Right & Down)")
plt.imshow(moved_image)
plt.axis("off")

plt.subplot(1, 3, 3)
plt.title("Moved Image (Comparison)")
plt.imshow(moved_image)
plt.axis("off")

plt.show()
```

Choose Files IMG_1154.JPG

IMG_1154.JPG(image/jpeg) - 76051 bytes, last modified: 12/18/2025 - 100% done

```
Saving IMG_1154.JPG to IMG_1154.JPG
```




```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path)

# Step 5: Convert BGR to RGB for display
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Step 6: Get image size
rows, cols = image_rgb.shape[:2]

# Step 7: Define source points (3 points)
src_points = np.float32([
    [0, 0],
    [cols - 1, 0],
    [0, rows - 1]
])
```

```
])

# Step 8: Define destination points (shifted / transformed)
dst_points = np.float32([
    [50, 50],
    [cols - 100, 50],
    [50, rows - 100]
])

# Step 9: Compute affine transformation matrix
affine_matrix = cv2.getAffineTransform(src_points, dst_points)

# Step 10: Apply affine transformation
affine_image = cv2.warpAffine(image_rgb, affine_matrix, (cols, rows))

# Step 11: Display images
plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image_rgb)
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Affine Transformed Image")
plt.imshow(affine_image)
plt.axis("off")

plt.show()
```

Choose Files aishu photo.avif

aishu photo.avif(image/avif) - 13376 bytes, last modified: 12/16/2025 - 100% done

```
Saving aishu photo.avif to aishu photo (4).avif
```


```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path)

# Step 5: Convert BGR to RGB for display
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Step 6: Get image size
rows, cols = image_rgb.shape[:2]

# Step 7: Define source points (4 points)
```

```
# Step 7: Define source points (4 points)
src_points = np.float32([
    [0, 0],
    [cols - 1, 0],
    [0, rows - 1],
    [cols - 1, rows - 1]
])

# Step 8: Define destination points (perspective effect)
dst_points = np.float32([
    [100, 50],
    [cols - 100, 0],
    [0, rows - 100],
    [cols - 50, rows - 50]
])

# Step 9: Compute perspective transformation matrix
perspective_matrix = cv2.getPerspectiveTransform(src_points, dst_points)

# Step 10: Apply perspective transformation
perspective_image = cv2.warpPerspective(image_rgb, perspective_matrix, (cols, rows))

# Step 11: Display images
plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image_rgb)
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Perspective Transformed Image")
plt.imshow(perspective_image)
plt.axis("off")

plt.show()
```

```
Choose Files  Snapchat-2…2505498.jpg
Snapchat-2112505498.jpg(image/jpeg) - 257295 bytes, last modified: 6/22/2023 - 100% done
Saving Snapchat-2112505498.jpg to Snapchat-2112505498.jpg
```


```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
```

```
image = cv2.imread(image_path)

# Step 5: Convert BGR to RGB for display
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Step 6: Get image dimensions
h, w = image_rgb.shape[:2]

# Step 7: Define source points (4 points on original image)
src_points = np.float32([
    [0, 0],
    [w - 1, 0],
    [0, h - 1],
    [w - 1, h - 1]
])

# Step 8: Define destination points (mapped points)
dst_points = np.float32([
    [100, 50],
    [w - 150, 80],
    [80, h - 120],
    [w - 100, h - 50]
])

# Step 9: Compute Homography matrix
H, status = cv2.findHomography(src_points, dst_points)

# Step 10: Apply Homography transformation
homography_image = cv2.warpPerspective(image_rgb, H, (w, h))

# Step 11: Display results
plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image_rgb)
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Homography Transformed Image")
plt.imshow(homography_image)
plt.axis("off")

plt.show()
```

Choose Files 20210502_060102.jpg

20210502_060102.jpg(image/jpeg) - 8287769 bytes, last modified: 5/2/2021 - 100% done

```
Saving 20210502_060102.jpg to 20210502_060102.jpg
```


```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]
```

```
# Step 4: Read the image
image = cv2.imread(image_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Step 5: Get image dimensions
h, w = image_rgb.shape[:2]

# Step 6: Define source points (4 points)
src_pts = np.array([
    [0, 0],
    [w - 1, 0],
    [0, h - 1],
    [w - 1, h - 1]
], dtype=np.float32)

# Step 7: Define destination points
dst_pts = np.array([
    [100, 50],
    [w - 150, 80],
    [80, h - 120],
    [w - 100, h - 50]
], dtype=np.float32)

# Step 8: Implement Direct Linear Transformation (DLT)
def compute_homography_DLT(src, dst):
    A = []
    for i in range(len(src)):
        x, y = src[i][0], src[i][1]
        u, v = dst[i][0], dst[i][1]

        A.append([-x, -y, -1, 0, 0, 0, x*u, y*u, u])
        A.append([0, 0, 0, -x, -y, -1, x*v, y*v, v])

    A = np.array(A)

    # Solve Ah = 0 using SVD
    U, S, Vt = np.linalg.svd(A)
    H = Vt[-1].reshape(3, 3)

    # Normalize
    return H / H[2, 2]

# Step 9: Compute homography using DLT
H_dlt = compute_homography_DLT(src_pts, dst_pts)

# Step 10: Apply homography transformation
dlt_image = cv2.warpPerspective(image_rgb, H_dlt, (w, h))

# Step 11: Display results
plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image_rgb)
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("DLT Transformed Image")
plt.imshow(dlt_image)
plt.axis("off")

plt.show()
```

Choose Files Screenshot… 122050.png

Screenshot 2025-12-22 122050.png(image/png) - 167380 bytes, last modified: 12/22/2025 - 100% done

```
Saving Screenshot 2025-12-22 122050.png to Screenshot 2025-12-22 122050.png
```

Original Image

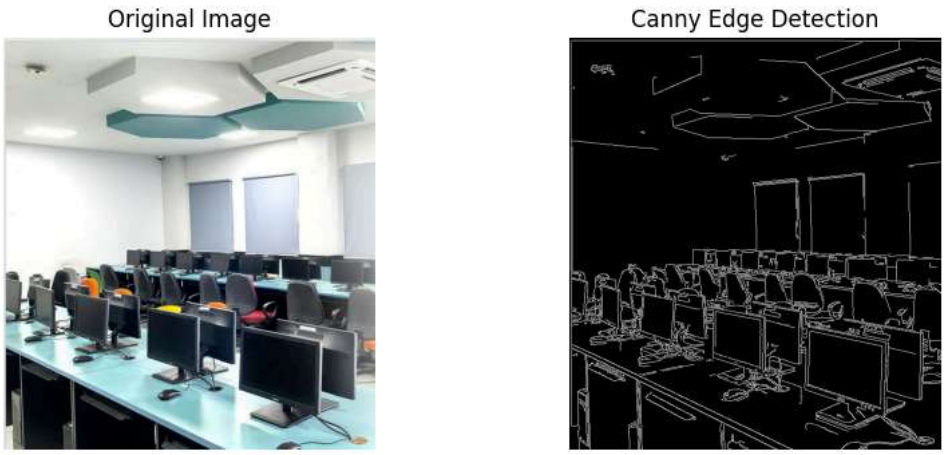
DLT Transformed Image

```
# Step 1: Import required libraries
import cv2
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path)

# Step 5: Convert image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 6: Apply Canny Edge Detection
# Threshold values can be adjusted
edges = cv2.Canny(gray_image, threshold1=100, threshold2=200)

# Step 7: Display original and edge-detected images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Canny Edge Detection")
plt.imshow(edges, cmap="gray")
plt.axis("off")

plt.show()
```

Choose Files Screenshot… 131237.png

Screenshot 2025-10-06 131237.png(image/png) - 484995 bytes, last modified: 10/6/2025 - 100% done

```
Saving Screenshot 2025-10-06 131237.png to Screenshot 2025-10-06 131237.png
```

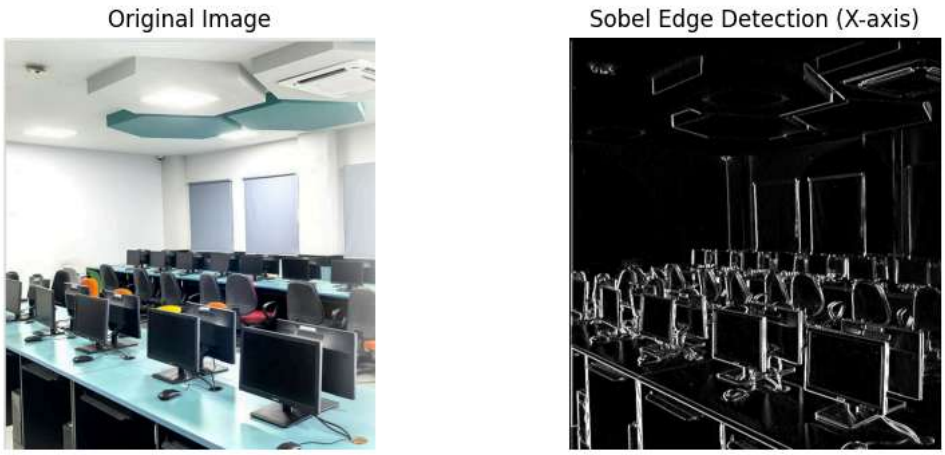



```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path)

# Step 5: Convert image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 6: Apply Sobel operator along X-axis
# dx=1, dy=0 → X direction
sobel_x = cv2.Sobel(
    gray_image,
    ddepth=cv2.CV_64F,
    dx=1,
    dy=0,
    ksize=3
)

# Step 7: Convert to absolute values and uint8
sobel_x_abs = cv2.convertScaleAbs(sobel_x)

# Step 8: Display original and Sobel X-edge images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Sobel Edge Detection (X-axis)")
plt.imshow(sobel_x_abs, cmap="gray")
plt.axis("off")

plt.show()
```

Choose Files Screenshot… 131237.png

Screenshot 2025-10-06 131237.png(image/png) - 484995 bytes, last modified: 10/6/2025 - 100% done

```
Saving Screenshot 2025-10-06 131237.png to Screenshot 2025-10-06 131237 (1).png
```

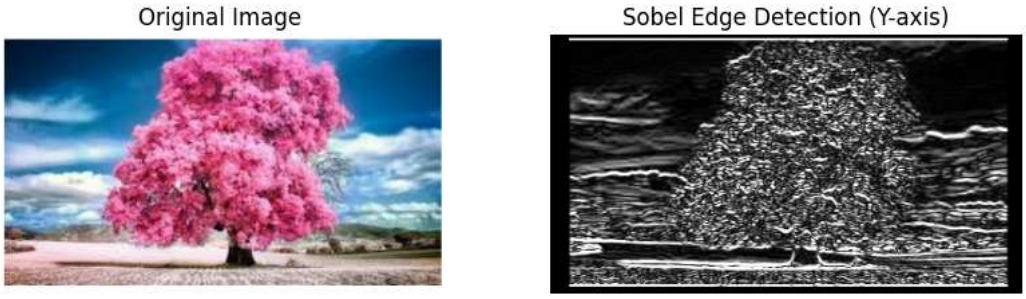



```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path)

# Step 5: Convert image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 6: Apply Sobel operator along Y-axis
# dx=0, dy=1 → Y direction
sobel_y = cv2.Sobel(
    gray_image,
    ddepth=cv2.CV_64F,
    dx=0,
    dy=1,
    ksize=3
)

# Step 7: Convert to absolute values and uint8
sobel_y_abs = cv2.convertScaleAbs(sobel_y)

# Step 8: Display original and Sobel Y-edge images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Sobel Edge Detection (Y-axis)")
plt.imshow(sobel_y_abs, cmap="gray")
plt.axis("off")

plt.show()
```

Choose Files Screenshot… 122050.png

Screenshot 2025-12-22 122050.png(image/png) - 167380 bytes, last modified: 12/22/2025 - 100% done

```
Saving Screenshot 2025-12-22 122050.png to Screenshot 2025-12-22 122050 (1).png
```




```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path)

# Step 5: Convert image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 6: Apply Sobel operator along X-axis
sobel_x = cv2.Sobel(gray_image, ddepth=cv2.CV_64F, dx=1, dy=0, ksize=3)

# Step 7: Apply Sobel operator along Y-axis
sobel_y = cv2.Sobel(gray_image, ddepth=cv2.CV_64F, dx=0, dy=1, ksize=3)

# Step 8: Compute gradient magnitude
sobel_xy = cv2.magnitude(sobel_x, sobel_y)

# Step 9: Convert to uint8 for display
sobel_xy = cv2.convertScaleAbs(sobel_xy)

# Step 10: Display original and Sobel XY images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Sobel Edge Detection (X & Y)")
plt.imshow(sobel_xy, cmap="gray")
plt.axis("off")

plt.show()
```

Choose Files Snapchat-2…2505498.jpg

Snapchat-2112505498.jpg(image/jpeg) - 257295 bytes, last modified: 6/22/2023 - 100% done

```
Saving Snapchat-2112505498.jpg to Snapchat-2112505498 (1).jpg
```


```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Step 5: Convert image to grayscale (optional)
gray_image = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2GRAY)

# Step 6: Define Laplacian sharpening kernel with negative center coefficient
laplacian_kernel = np.array([
    [0, -1, 0],
    [-1, 5, -1],  # Center coefficient is negative in subtraction sense; here 5 sharpens
    [0, -1, 0]
], dtype=np.float32)

# Step 7: Apply sharpening using filter2D
sharpened_image = cv2.filter2D(gray_image, -1, laplacian_kernel)

# Step 8: Display original and sharpened images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Grayscale Image")
plt.imshow(gray_image, cmap="gray")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Sharpened Image (Laplacian)")
plt.imshow(sharpened_image, cmap="gray")
plt.axis("off")

plt.show()
```

Choose Files Screenshot… 122050.png

Screenshot 2025-12-22 122050.png(image/png) - 167380 bytes, last modified: 12/22/2025 - 100% done

```
Saving Screenshot 2025-12-22 122050.png to Screenshot 2025-12-22 122050 (2).png
```


```
# Laplacian Image Sharpening with Negative Center Coefficient (Google Colab)

import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Upload image
uploaded = files.upload()

# Read image in grayscale
image_name = list(uploaded.keys())[0]
img = cv2.imread(image_name, cv2.IMREAD_GRAYSCALE)
if img is None:
    raise ValueError("Image not loaded properly")

# Laplacian mask (negative center)
laplacian_kernel = np.array([[0,  1, 0],
                             [1, -4, 1],
                             [0,  1, 0]])

# Apply Laplacian filter
laplacian = cv2.filter2D(img, cv2.CV_64F, laplacian_kernel)

# Sharpen image (negative center ⇒ subtract Laplacian)
sharpened = img - laplacian
sharpened = np.clip(sharpened, 0, 255).astype(np.uint8)

# Display results
plt.figure(figsize=(12,4))

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

plt.subplot(1,3,2)
plt.imshow(laplacian, cmap='gray')
plt.title("Laplacian")
plt.axis('off')

plt.subplot(1,3,3)
plt.imshow(sharpened, cmap='gray')
plt.title("Sharpened Image")
plt.axis('off')

plt.show()
```

Choose Files Screenshot… 131237.png

Screenshot 2025-10-06 131237.png(image/png) - 484995 bytes, last modified: 10/6/2025 - 100% done

```
Saving Screenshot 2025-10-06 131237.png to Screenshot 2025-10-06 131237 (2).png
```


```
# Laplacian Image Sharpening with Diagonal Neighbors (Negative Center = -8)
# Google Colab – Single Complete Code

import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Upload image
uploaded = files.upload()

# Read image in grayscale
image_name = list(uploaded.keys())[0]
img = cv2.imread(image_name, cv2.IMREAD_GRAYSCALE)
if img is None:
    raise ValueError("Image not loaded properly")

# Laplacian mask with diagonal neighbors
laplacian_kernel = np.array([[1,  1,  1],
                             [1, -8,  1],
                             [1,  1,  1]])

# Apply Laplacian filter
laplacian = cv2.filter2D(img, cv2.CV_64F, laplacian_kernel)

# Sharpen image (negative center ⇒ subtract Laplacian)
sharpened = img - laplacian
sharpened = np.clip(sharpened, 0, 255).astype(np.uint8)

# Display results
plt.figure(figsize=(12,4))

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

plt.subplot(1,3,2)
plt.imshow(laplacian, cmap='gray')
plt.title("Laplacian (Diagonal Neighbors)")
plt.axis('off')

plt.subplot(1,3,3)
plt.imshow(sharpened, cmap='gray')
plt.title("Sharpened Image")
plt.axis('off')

plt.show()
```

Choose Files 20210126_071908.jpg

20210126_071908.jpg(image/jpeg) - 1827756 bytes, last modified: 1/26/2021 - 100% done

```
Saving 20210126_071908.jpg to 20210126_071908.jpg
```


```
# Laplacian Image Sharpening with Positive Center Coefficient
# Google Colab – Single Complete Code

import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Upload image
uploaded = files.upload()

# Read image in grayscale
image_name = list(uploaded.keys())[0]
img = cv2.imread(image_name, cv2.IMREAD_GRAYSCALE)
if img is None:
    raise ValueError("Image not loaded properly")

# Laplacian mask with positive center coefficient
laplacian_kernel = np.array([[ 0, -1,  0],
                             [-1,  5, -1],
                             [ 0, -1,  0]])

# Apply sharpening using Laplacian mask (positive center ⇒ direct convolution)
sharpened = cv2.filter2D(img, -1, laplacian_kernel)

# Display results
plt.figure(figsize=(8,4))

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

plt.subplot(1,2,2)
plt.imshow(sharpened, cmap='gray')
plt.title("Sharpened Image")
plt.axis('off')

plt.show()
```

Choose Files aishu photo.avif

aishu photo.avif(image/avif) - 13376 bytes, last modified: 12/16/2025 - 100% done

```
Saving aishu photo.avif to aishu photo (5).avif
```


```
# Unsharp Masking Image Sharpening
# Google Colab – Single Complete Code

import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Upload image
uploaded = files.upload()

# Read image in grayscale
image_name = list(uploaded.keys())[0]
img = cv2.imread(image_name, cv2.IMREAD_GRAYSCALE)
if img is None:
    raise ValueError("Image not loaded properly")

# Create blurred image using Gaussian filter
blurred = cv2.GaussianBlur(img, (5, 5), 0)

# Unsharp masking
# mask = original - blurred
mask = img - blurred

# Sharpened image = original + mask
sharpened = img + mask
sharpened = np.clip(sharpened, 0, 255).astype(np.uint8)

# Display results
plt.figure(figsize=(12,4))

plt.subplot(1,4,1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis('off')

plt.subplot(1,4,2)
plt.imshow(blurred, cmap='gray')
plt.title("Blurred Image")
plt.axis('off')

plt.subplot(1,4,3)
plt.imshow(mask, cmap='gray')
plt.title("Unsharp Mask")
plt.axis('off')

plt.subplot(1,4,4)
plt.imshow(sharpened, cmap='gray')
plt.title("Sharpened Image")
plt.axis('off')

plt.show()
```

Choose Files aishu photo.avif

aishu photo.avif(image/avif) - 13376 bytes, last modified: 12/16/2025 - 100% done

```
Saving aishu photo.avif to aishu photo (6).avif
```


```
# High-Boost Image Sharpening
# Google Colab – Single Complete Code

import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Upload image
uploaded = files.upload()

# Read image in grayscale
image_name = list(uploaded.keys())[0]
img = cv2.imread(image_name, cv2.IMREAD_GRAYSCALE)
if img is None:
    raise ValueError("Image not loaded properly")

# High-boost parameter (A ≥ 1)
A = 1.5   # you can change this value (A=1 gives standard Laplacian sharpening)

# High-boost mask (4-neighbor Laplacian version)
high_boost_kernel = np.array([[ 0, -1,  0],
                              [-1, A + 4, -1],
                              [ 0, -1,  0]])

# Apply high-boost sharpening
sharpened = cv2.filter2D(img, -1, high_boost_kernel)

# Display results
plt.figure(figsize=(8,4))

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

plt.subplot(1,2,2)
plt.imshow(sharpened, cmap='gray')
plt.title("High-Boost Sharpened Image")
plt.axis('off')

plt.show()
```

Choose Files aishu photo.avif

aishu photo.avif(image/avif) - 13376 bytes, last modified: 12/16/2025 - 100% done

```
Saving aishu photo.avif to aishu photo (7).avif
```

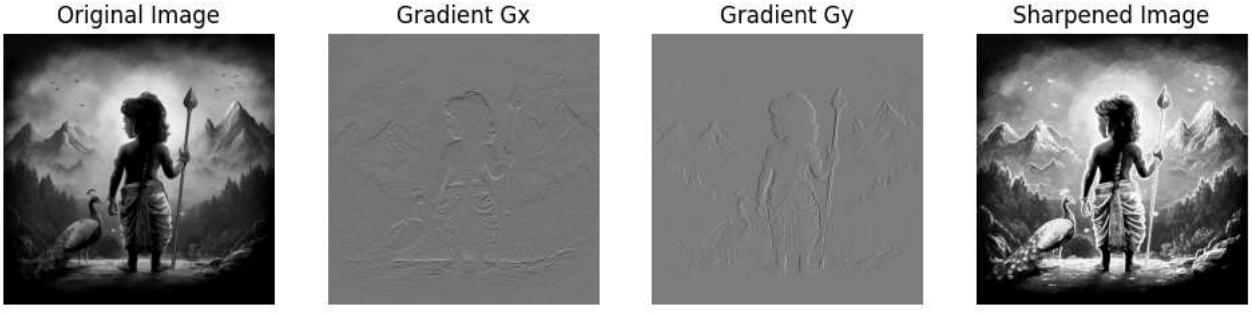

```
# Gradient Masking Image Sharpening (Sobel Operator)
# Google Colab – Single Complete Code
```

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Upload image
uploaded = files.upload()

# Read image in grayscale
image_name = list(uploaded.keys())[0]
img = cv2.imread(image_name, cv2.IMREAD_GRAYSCALE)
if img is None:
    raise ValueError("Image not loaded properly")

# Gradient masks (as shown in the image)
Gx_kernel = np.array([[-1, -2, -1],
                      [ 0,  0,  0],
                      [ 1,  2,  1]])

Gy_kernel = np.array([[-1,  0,  1],
                      [-2,  0,  2],
                      [-1,  0,  1]])

# Apply gradient masks
Gx = cv2.filter2D(img, cv2.CV_64F, Gx_kernel)
Gy = cv2.filter2D(img, cv2.CV_64F, Gy_kernel)

# Gradient magnitude
gradient_magnitude = np.sqrt(Gx**2 + Gy**2)

# Sharpened image = original + gradient magnitude
sharpened = img + gradient_magnitude
sharpened = np.clip(sharpened, 0, 255).astype(np.uint8)

# Display results
plt.figure(figsize=(12,4))

plt.subplot(1,4,1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis('off')

plt.subplot(1,4,2)
plt.imshow(Gx, cmap='gray')
plt.title("Gradient Gx")
plt.axis('off')

plt.subplot(1,4,3)
plt.imshow(Gy, cmap='gray')
plt.title("Gradient Gy")
plt.axis('off')

plt.subplot(1,4,4)
plt.imshow(sharpened, cmap='gray')
plt.title("Sharpened Image")
plt.axis('off')

plt.show()
```

Choose Files IMG_1154.JPG

IMG_1154.JPG(image/jpeg) - 76051 bytes, last modified: 12/18/2025 - 100% done

```
Saving IMG_1154.JPG to IMG_1154 (1).JPG
```

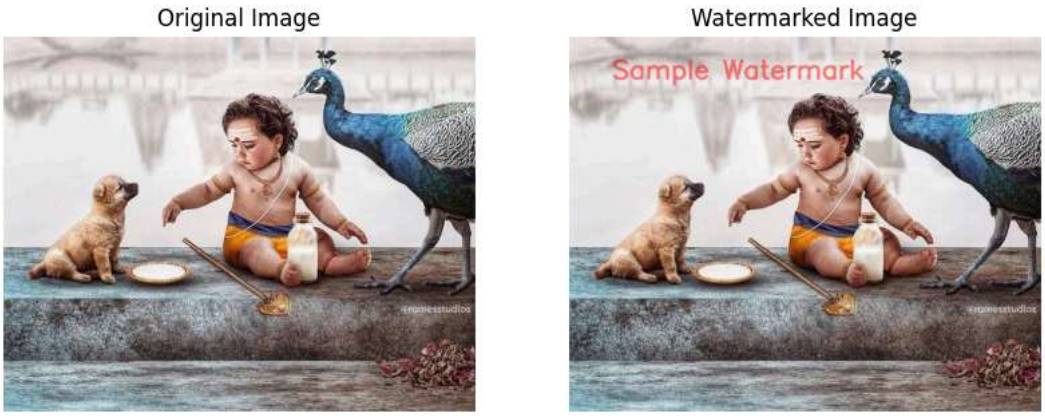


```
# Step 1: Import required libraries
import cv2
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Step 5: Define watermark text and properties
watermark_text = "Sample Watermark"
position = (50, 50)           # (x, y) position of text
font = cv2.FONT_HERSHEY_SIMPLEX
font_scale = 1
color = (255, 0, 0)           # Red color (BGR in OpenCV, but we display in RGB)
thickness = 2
opacity = 0.5                 # Transparency factor

# Step 6: Create a copy for overlay
overlay = image_rgb.copy()

# Step 7: Put the text on overlay
cv2.putText(overlay, watermark_text, position, font, font_scale, color, thickness, cv2.LINE_AA)

# Step 8: Blend the overlay with original image
watermarked_image = cv2.addWeighted(overlay, opacity, image_rgb, 1 - opacity, 0)

# Step 9: Display original and watermarked images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Image")
plt.imshow(image_rgb)
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Watermarked Image")
plt.imshow(watermarked_image)
plt.axis("off")

plt.show()
```

Choose Files murugaa.jpeg

murugaa.jpeg(image/jpeg) - 90744 bytes, last modified: 12/18/2025 - 100% done

```
Saving murugaa.jpeg to murugaa (1).jpeg
```




```
# Step 1: Import required libraries
import cv2
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload two images
```

```
print("Upload the base image (background):")
uploaded_base = files.upload()

print("Upload the image to crop and paste (foreground):")
uploaded_fg = files.upload()

# Step 3: Read images
base_path = list(uploaded_base.keys())[0]
fg_path = list(uploaded_fg.keys())[0]

base_image = cv2.imread(base_path)
fg_image = cv2.imread(fg_path)

# Convert BGR to RGB for display
base_rgb = cv2.cvtColor(base_image, cv2.COLOR_BGR2RGB)
fg_rgb = cv2.cvtColor(fg_image, cv2.COLOR_BGR2RGB)

# Step 4: Crop a region from foreground image
# Example crop coordinates: y1:y2, x1:x2
y1, y2, x1, x2 = 50, 200, 50, 200
cropped_fg = fg_rgb[y1:y2, x1:x2]

# Step 5: Define position to paste on base image
paste_y, paste_x = 100, 150
h, w = cropped_fg.shape[:2]

# Step 6: Paste the cropped region into base image
base_rgb[paste_y:paste_y+h, paste_x:paste_x+w] = cropped_fg

# Step 7: Display results
plt.figure(figsize=(12, 6))

plt.subplot(1, 3, 1)
plt.title("Base Image")
plt.imshow(base_rgb)
plt.axis("off")

plt.subplot(1, 3, 2)
plt.title("Cropped Foreground")
plt.imshow(cropped_fg)
plt.axis("off")

plt.subplot(1, 3, 3)
plt.title("Final Image (Pasted)")
plt.imshow(base_rgb)
plt.axis("off")

plt.show()
```

```
Upload the base image (background):
```
Choose Files aishu photo.avif

aishu photo.avif(image/avif) - 13376 bytes, last modified: 12/16/2025 - 100% done

```
Saving aishu photo.avif to aishu photo (8).avif
Upload the image to crop and paste (foreground):
```
Choose Files aishu photo.avif

aishu photo.avif(image/avif) - 13376 bytes, last modified: 12/16/2025 - 100% done

```
Saving aishu photo.avif to aishu photo (9).avif
```

```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload the image
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Step 5: Convert image to grayscale
gray_image = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2GRAY)

# Step 6: Define a boundary detection kernel (Laplacian style)
# This kernel highlights edges by detecting intensity changes
boundary_kernel = np.array([
    [-1, -1, -1],
    [-1,  8, -1],
    [-1, -1, -1]
], dtype=np.float32)

# Step 7: Apply convolution using filter2D
boundary_image = cv2.filter2D(gray_image, -1, boundary_kernel)

# Step 8: Display original and boundary images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Grayscale Image")
plt.imshow(gray_image, cmap="gray")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Boundary Image")
plt.imshow(boundary_image, cmap="gray")
plt.axis("off")

plt.show()
```

Choose Files aishu photo.avif

aishu photo.avif(image/avif) - 13376 bytes, last modified: 12/16/2025 - 100% done

```
Saving aishu photo.avif to aishu photo (10).avif
```


```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload an image (preferably binary or grayscale)
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]
```

```
# Step 4: Read the image
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Optional: Threshold the image to make it binary
_, binary_image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)

# Step 5: Define a kernel (structuring element)
kernel = np.ones((5, 5), np.uint8)  # 5x5 square kernel

# Step 6: Apply Erosion
eroded_image = cv2.erode(binary_image, kernel, iterations=1)

# Step 7: Display original and eroded images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Binary Image")
plt.imshow(binary_image, cmap="gray")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Eroded Image")
plt.imshow(eroded_image, cmap="gray")
plt.axis("off")

plt.show()
```

Choose Files aishu photo.avif

aishu photo.avif(image/avif) - 13376 bytes, last modified: 12/16/2025 - 100% done

Saving aishu photo.avif to aishu photo (11).avif

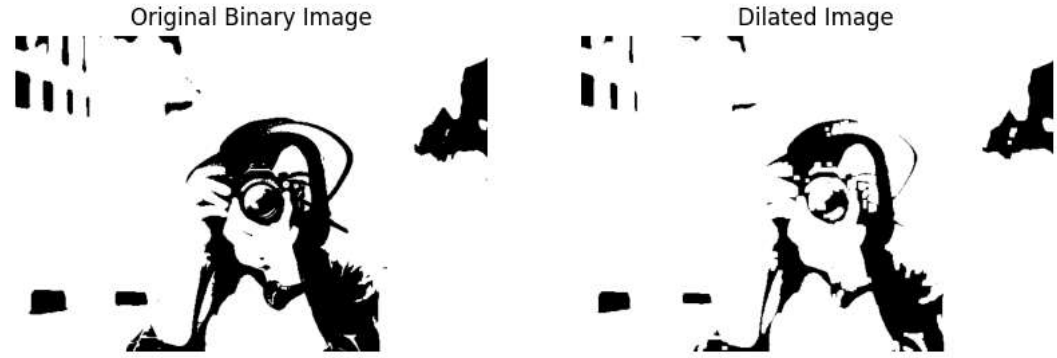

```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload an image (preferably binary or grayscale)
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Optional: Threshold the image to make it binary
_, binary_image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)

# Step 5: Define a kernel (structuring element)
kernel = np.ones((5, 5), np.uint8)  # 5x5 square kernel

# Step 6: Apply Dilation
dilated_image = cv2.dilate(binary_image, kernel, iterations=1)

# Step 7: Display original and dilated images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Binary Image")
plt.imshow(binary_image, cmap="gray")
```

```
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Dilated Image")
plt.imshow(dilated_image, cmap="gray")
plt.axis("off")

plt.show()
```

Choose Files aishu photo.avif

aishu photo.avif(image/avif) - 13376 bytes, last modified: 12/16/2025 - 100% done

```
Saving aishu photo.avif to aishu photo (12).avif
```

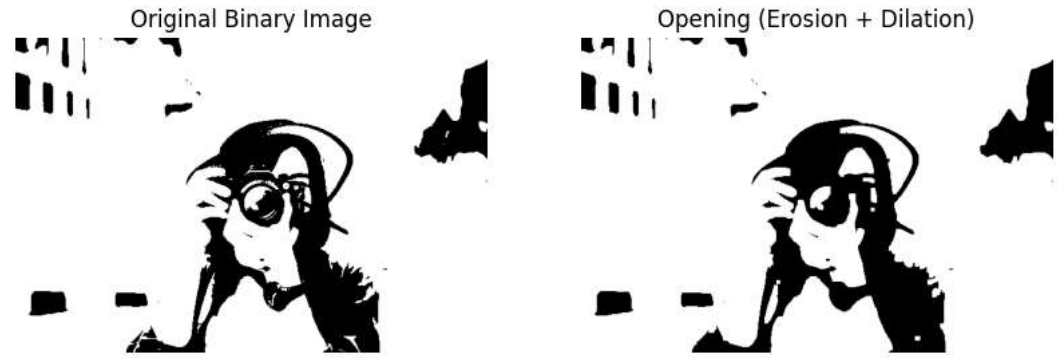



```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload an image (preferably binary or grayscale)
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Optional: Threshold the image to make it binary
_, binary_image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)

# Step 5: Define a kernel (structuring element)
kernel = np.ones((5, 5), np.uint8)  # 5x5 square kernel

# Step 6: Apply Opening (Erosion followed by Dilation)
opening_image = cv2.morphologyEx(binary_image, cv2.MORPH_OPEN, kernel)

# Step 7: Display original and opening images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Binary Image")
plt.imshow(binary_image, cmap="gray")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Opening (Erosion + Dilation)")
plt.imshow(opening_image, cmap="gray")
plt.axis("off")

plt.show()
```

Choose Files aishu photo.avif

aishu photo.avif(image/avif) - 13376 bytes, last modified: 12/16/2025 - 100% done

```
Saving aishu photo.avif to aishu photo (13).avif
```

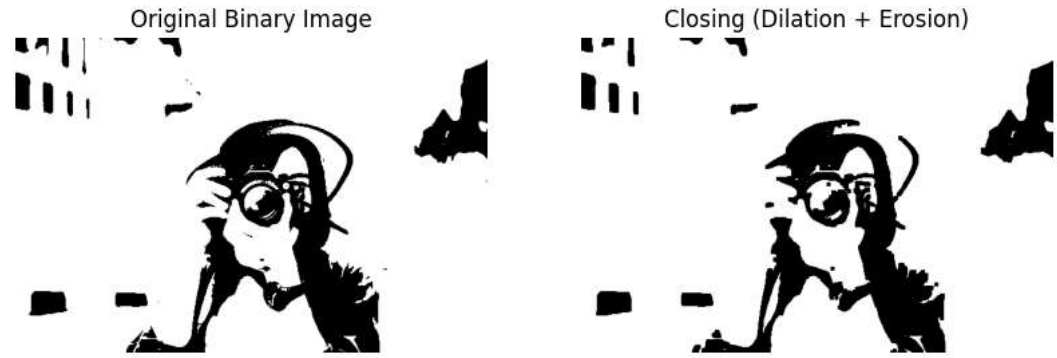



```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload an image (preferably binary or grayscale)
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Optional: Threshold the image to make it binary
_, binary_image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)

# Step 5: Define a kernel (structuring element)
kernel = np.ones((5, 5), np.uint8)  # 5x5 square kernel

# Step 6: Apply Closing (Dilation followed by Erosion)
closing_image = cv2.morphologyEx(binary_image, cv2.MORPH_CLOSE, kernel)

# Step 7: Display original and closing images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Binary Image")
plt.imshow(binary_image, cmap="gray")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Closing (Dilation + Erosion)")
plt.imshow(closing_image, cmap="gray")
plt.axis("off")

plt.show()
```

Choose Files aishu photo.avif

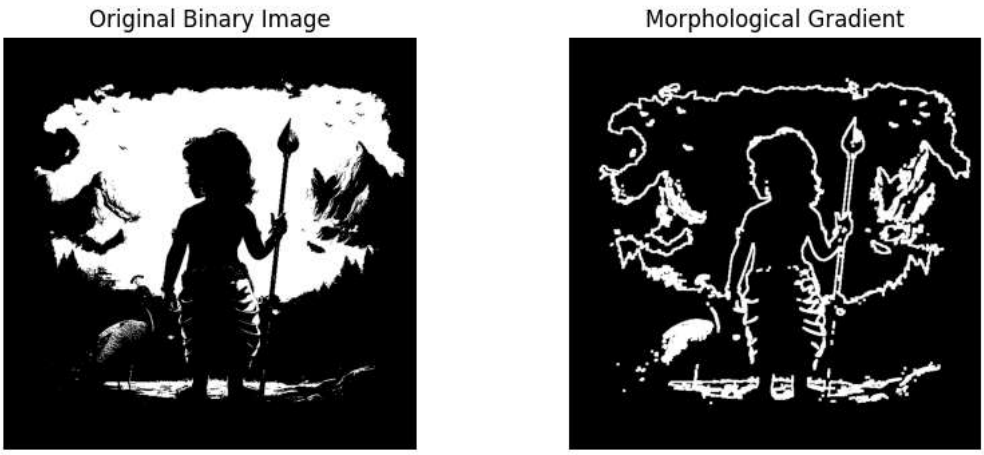
aishu photo.avif(image/avif) - 13376 bytes, last modified: 12/16/2025 - 100% done

```
Saving aishu photo.avif to aishu photo (14).avif
```




```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload an image (preferably binary or grayscale)
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Optional: Threshold the image to make it binary
_, binary_image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)

# Step 5: Define a kernel (structuring element)
kernel = np.ones((5, 5), np.uint8)  # 5x5 square kernel

# Step 6: Apply Morphological Gradient
morph_gradient = cv2.morphologyEx(binary_image, cv2.MORPH_GRADIENT, kernel)

# Step 7: Display original and gradient images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Binary Image")
plt.imshow(binary_image, cmap="gray")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Morphological Gradient")
plt.imshow(morph_gradient, cmap="gray")
plt.axis("off")

plt.show()
```

```
Choose Files IMG_1154.JPG
IMG_1154.JPG(image/jpeg) - 76051 bytes, last modified: 12/18/2025 - 100% done
Saving IMG_1154.JPG to IMG_1154 (2).JPG
```

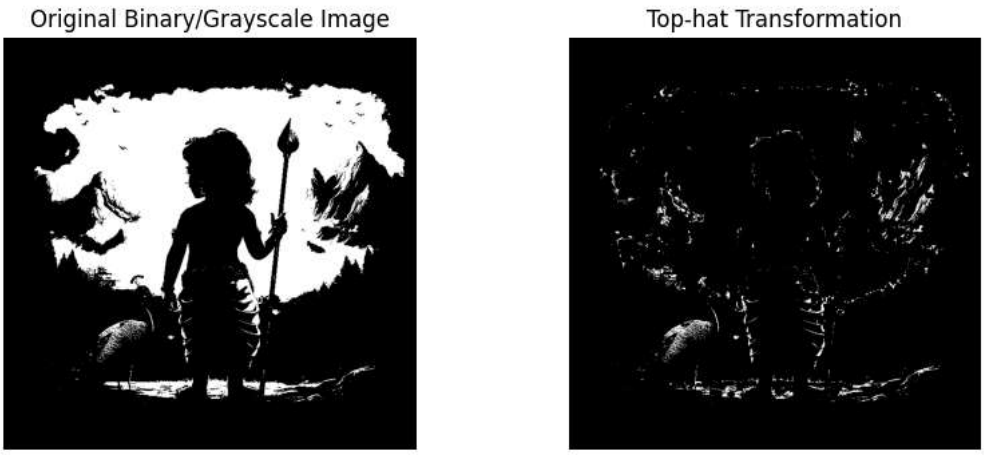



```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload an image (preferably grayscale)
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Optional: Thresholding (if needed)
_, binary_image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)

# Step 5: Define a kernel (structuring element)
kernel = np.ones((5, 5), np.uint8)  # 5x5 square kernel

# Step 6: Apply Top-hat transformation
top_hat = cv2.morphologyEx(binary_image, cv2.MORPH_TOPHAT, kernel)

# Step 7: Display original and Top-hat images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Binary/Grayscale Image")
plt.imshow(binary_image, cmap="gray")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Top-hat Transformation")
plt.imshow(top_hat, cmap="gray")
plt.axis("off")

plt.show()
```

Choose Files IMG_1154.JPG

IMG_1154.JPG(image/jpeg) - 76051 bytes, last modified: 12/18/2025 - 100% done

```
Saving IMG_1154.JPG to IMG_1154 (3).JPG
```

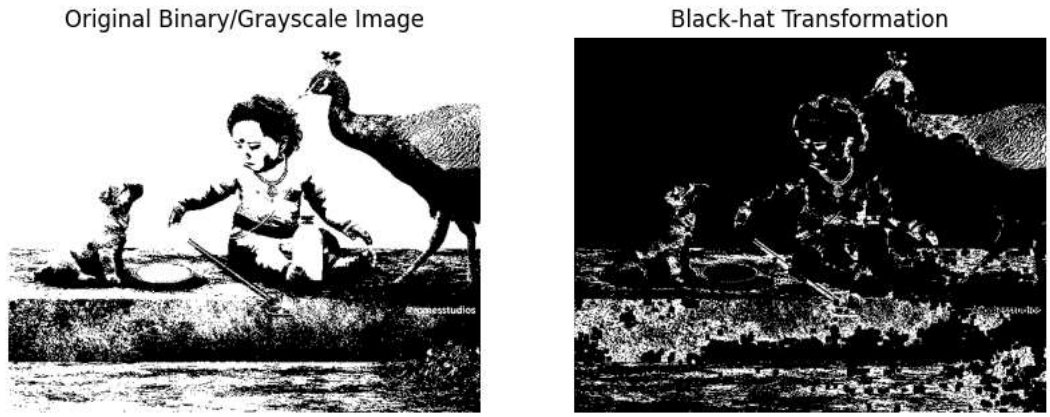



```
# Step 1: Import required libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Step 2: Upload an image (preferably grayscale)
uploaded = files.upload()

# Step 3: Get uploaded image path
image_path = list(uploaded.keys())[0]

# Step 4: Read the image
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Optional: Thresholding (if needed)
_, binary_image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)

# Step 5: Define a kernel (structuring element)
kernel = np.ones((5, 5), np.uint8)  # 5x5 square kernel

# Step 6: Apply Black-hat transformation
black_hat = cv2.morphologyEx(binary_image, cv2.MORPH_BLACKHAT, kernel)

# Step 7: Display original and Black-hat images
plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)
plt.title("Original Binary/Grayscale Image")
plt.imshow(binary_image, cmap="gray")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.title("Black-hat Transformation")
plt.imshow(black_hat, cmap="gray")
plt.axis("off")

plt.show()
```

Choose Files murugaa.jpeg

murugaa.jpeg(image/jpeg) - 90744 bytes, last modified: 12/18/2025 - 100% done

```
Saving murugaa.jpeg to murugaa (2).jpeg
```




```
# Watch Recognition using General Object Recognition (OpenCV)
# Google Colab – Single Complete Code

import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

# Upload image
uploaded = files.upload()

# Read image
image_name = list(uploaded.keys())[0]
img = cv2.imread(image_name)
if img is None:
    raise ValueError("Image not loaded properly")

# Convert to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Blur to reduce noise
blur = cv2.GaussianBlur(gray, (9, 9), 1.5)

# Detect circular objects using Hough Circle Transform (watch dial)
circles = cv2.HoughCircles(
    blur,
    cv2.HOUGH_GRADIENT,
    dp=1.2,
    minDist=100,
    param1=100,
    param2=30,
    minRadius=30,
    maxRadius=200
)

# Draw detected watch
output = img.copy()
if circles is not None:
    circles = np.uint16(np.around(circles))
    for (x, y, r) in circles[0]:
        cv2.circle(output, (x, y), r, (0, 255, 0), 3)
        cv2.circle(output, (x, y), 2, (0, 0, 255), 3)
        cv2.putText(output, "Watch Detected", (x-50, y-r-10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0,255,0), 2)
else:
    print("No watch detected")

# Display results
plt.figure(figsize=(10,4))

plt.subplot(1,2,1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title("Original Image")
plt.axis('off')
```

```
plt.subplot(1,2,2)
plt.imshow(cv2.cvtColor(output, cv2.COLOR_BGR2RGB))
plt.title("Recognised Watch")
plt.axis('off')

plt.show()
```

Choose Files Screenshot... 123453.png

Screenshot 2025-12-23 123453.png(image/png) - 128740 bytes, last modified: 12/23/2025 - 100% done

```
Saving Screenshot 2025-12-23 123453.png to Screenshot 2025-12-23 123453 (1).png
/tmp/ipython-input-440158167.py:43: RuntimeWarning: overflow encountered in scalar subtract
  cv2.putText(output, "Watch Detected", (x-50, y-r-10),
```