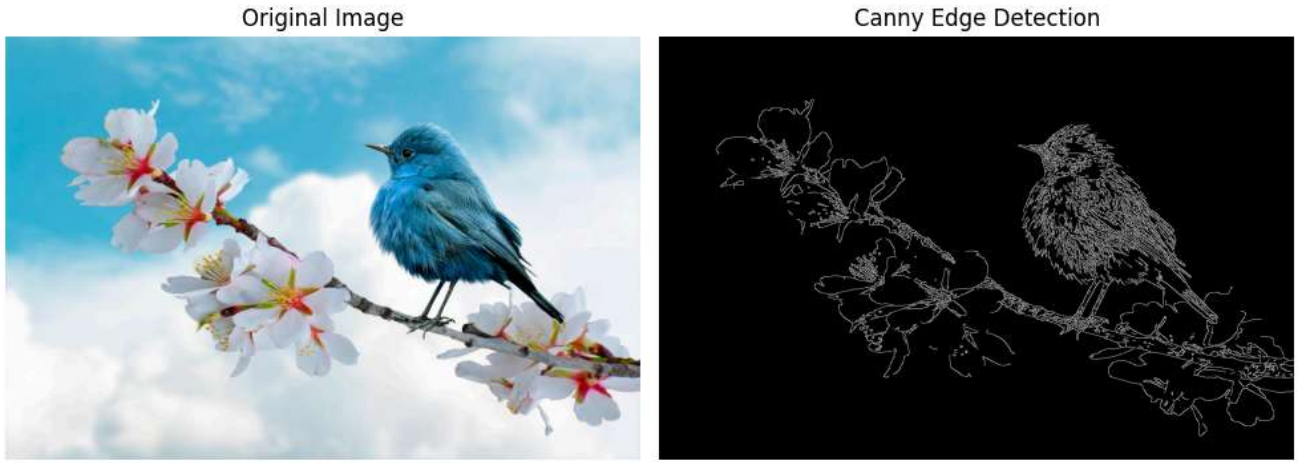
```
from google.colab import files
uploaded = files.upload()
```

Choose Files image.ppm
image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done
Saving image.ppm to image (1).ppm

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("image.ppm")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

plt.imshow(gray, cmap="gray")
plt.axis("off")
plt.show()
```

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("image.ppm")
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

blur = cv2.GaussianBlur(img_rgb, (31 , 31), 1.5)

plt.figure(figsize=(8, 4))

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

plt.subplot(1, 2, 2)
plt.imshow(blur)
plt.title("Gaussian Blurred Image")
plt.axis("off")

plt.tight_layout()
plt.show()
```


```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("image.ppm")

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

edges = cv2.Canny(gray, 100, 200)

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

plt.tight_layout()
plt.show()
```




```
from google.colab import files
uploaded = files.upload()
```

Choose Files image.ppm
image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done
Saving image.ppm to image (2).ppm

```
# Import libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
# Read the image
img = cv2.imread(list(uploaded.keys())[0])

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

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

# Apply dilation
dilated_img = cv2.dilate(gray_img, kernel, iterations=1)

# Display original and dilated images
plt.figure(figsize=(10,5))

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

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

plt.show()
```


```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("image.ppm", cv2.IMREAD_GRAYSCALE)


if img is None:
    raise FileNotFoundError("Image not found. Check the filename or path.")

kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))


eroded = cv2.erode(img, kernel, iterations=1)

plt.figure(figsize=(10, 5))

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

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

plt.tight_layout()
plt.show()
```


```
from google.colab import files
uploaded = files.upload()
```

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

```
Saving image.ppm to image (3).ppm
```

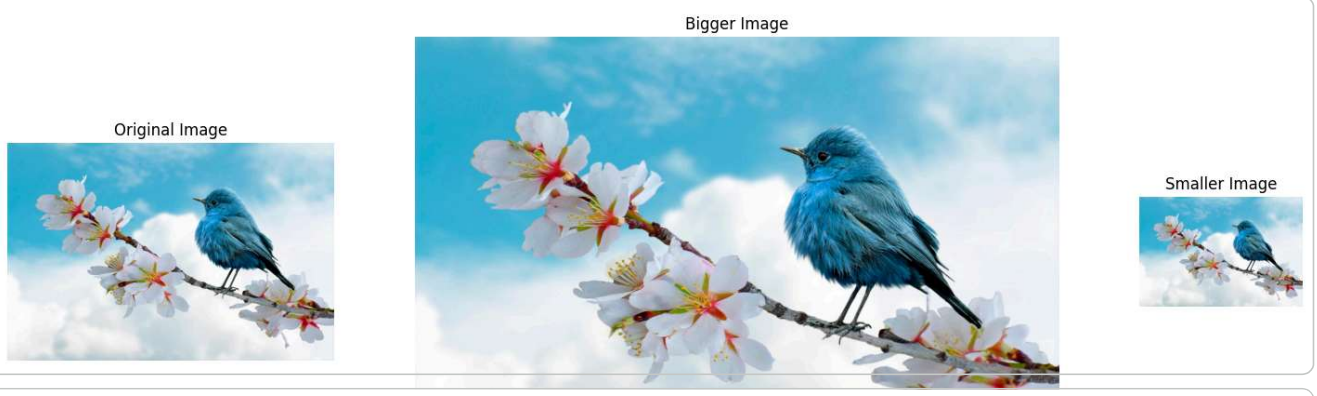
```
# Upload video file
from google.colab import files
uploaded = files.upload()

# Import libraries
import cv2
import time
from google.colab.patches import cv2_imshow

# Read the video
video_path = list(uploaded.keys())[0]
cap = cv2.VideoCapture(video_path)

# ----- SPEED CONTROL -----
# Slow motion : 0.1
# Normal      : 0.03
# Fast motion : 0.005
delay = 0.1   # change this value for slow / fast motion

# Read and display video frames
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    cv2_imshow(frame)
    time.sleep(delay)

cap.release()
```

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

```
Saving image.ppm to image (4).ppm
```

```
from google.colab import files
import cv2
import matplotlib.pyplot as plt
# Import libraries
import cv2
import matplotlib.pyplot as plt

# Read image
img = cv2.imread(list(uploaded.keys())[0])

# Scaling
bigger_img = cv2.resize(img, None, fx=2, fy=2, interpolation=cv2.INTER_LINEAR)
smaller_img = cv2.resize(img, None, fx=0.5, fy=0.5, interpolation=cv2.INTER_AREA)

# Horizontal display with different widths
fig, axs = plt.subplots(
    1, 3,
    figsize=(18, 6),
    gridspec_kw={'width_ratios': [1, 2, 0.5]}
)

axs[0].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
axs[0].set_title("Original Image")
axs[0].axis('off')

axs[1].imshow(cv2.cvtColor(bigger_img, cv2.COLOR_BGR2RGB))
axs[1].set_title("Bigger Image")
axs[1].axis('off')

axs[2].imshow(cv2.cvtColor(smaller_img, cv2.COLOR_BGR2RGB))
axs[2].set_title("Smaller Image")
axs[2].axis('off')

plt.show()
```




```
from google.colab import files
uploaded = files.upload()
from PIL import Image
import matplotlib.pyplot as plt

# Get uploaded image name
image_name = list(uploaded.keys())[0]

# Open image
img = Image.open(image_name)

# Rotate image
clockwise = img.rotate(-90, expand=True)
counter_clockwise = img.rotate(90, expand=True)

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

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

plt.subplot(1,3,2)
plt.title("Clockwise")
plt.imshow(clockwise)
plt.axis("off")

plt.subplot(1,3,3)
plt.title("Counter Clockwise")
plt.imshow(counter_clockwise)
plt.axis("off")

plt.show()
```

Choose Files image.ppm
image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done
Saving image.ppm to image (5).ppm

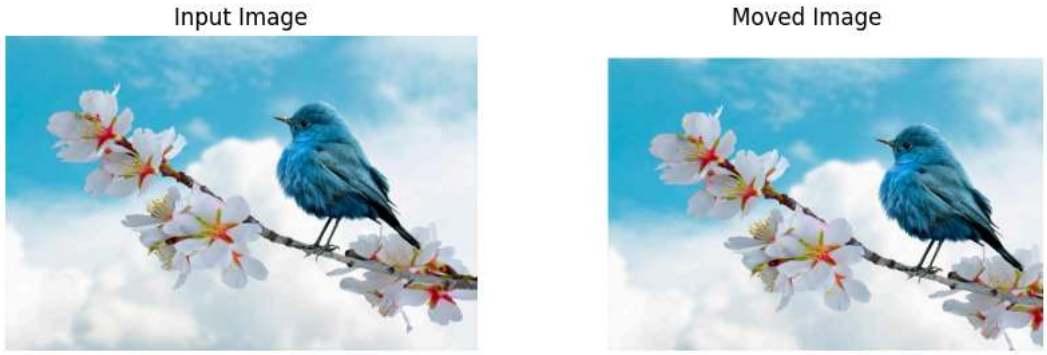

```
from google.colab import files
uploaded = files.upload()
```

```
from PIL import Image
import matplotlib.pyplot as plt

# Take uploaded image as input
img_name = list(uploaded.keys())[0]
img = Image.open(img_name)

# Image size
width, height = img.size

# Create blank background
output = Image.new("RGB", (width, height), (255, 255, 255))

# Translation values (change these)
tx = 100   # move right
ty = 60    # move down

# Move image
output.paste(img, (tx, ty))

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

plt.subplot(1,2,1)
plt.title("Input Image")
plt.imshow(img)
plt.axis("off")

plt.subplot(1,2,2)
plt.title("Moved Image")
plt.imshow(output)
plt.axis("off")

plt.show()
```

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

Saving image.ppm to image (6).ppm




```
from google.colab import files
uploaded = files.upload()
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Get uploaded image name
img_name = list(uploaded.keys())[0]

# Read image
img = cv2.imread(img_name)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Get image size
rows, cols, ch = img.shape

# Source points (original image)
src_points = np.float32([
    [0, 0],
    [cols - 1, 0],
    [0, rows - 1]
])
```

```
# Destination points (after transformation)
dst_points = np.float32([
    [50, 50],          # shifted
    [cols - 100, 80],  # skewed
    [80, rows - 100]
])

# Get affine transformation matrix
matrix = cv2.getAffineTransform(src_points, dst_points)

# Apply affine transformation
affine_img = cv2.warpAffine(img, matrix, (cols, rows))

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

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

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

plt.show()
```

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

```
Saving image.ppm to image (7).ppm
```


```
from google.colab import files
uploaded = files.upload()
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Get uploaded image name
img_name = list(uploaded.keys())[0]

# Read image
img = cv2.imread(img_name)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Image dimensions
rows, cols, ch = img.shape

# Source points (4 corners of the original image)
src_points = np.float32([
    [0, 0],
    [cols - 1, 0],
    [cols - 1, rows - 1],
    [0, rows - 1]
])

# Destination points (perspective view)
dst_points = np.float32([
    [50, 50],
    [cols - 100, 30],
    [cols - 50, rows - 50],
```

```
    [100, rows - 80]
])

# Get perspective transformation matrix
matrix = cv2.getPerspectiveTransform(src_points, dst_points)

# Apply perspective transformation
perspective_img = cv2.warpPerspective(img, matrix, (cols, rows))

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

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

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

plt.show()
```

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

```
Saving image.ppm to image (8).ppm
```

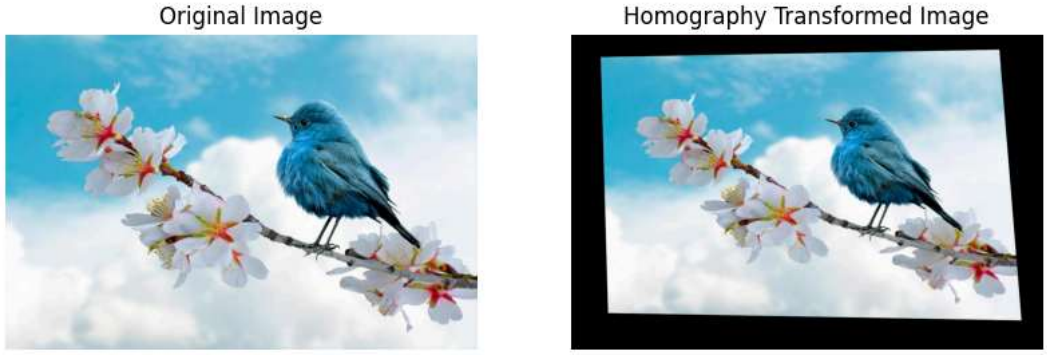

```
from google.colab import files
uploaded = files.upload()
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Get uploaded image name
img_name = list(uploaded.keys())[0]

# Read image
img = cv2.imread(img_name)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Image size
rows, cols, ch = img.shape

# Source points (4 points from original image)
src_points = np.float32([
    [0, 0],
    [cols - 1, 0],
    [cols - 1, rows - 1],
    [0, rows - 1]
])

# Destination points (mapped positions)
dst_points = np.float32([
    [80, 60],
    [cols - 120, 40],
    [cols - 60, rows - 80],
    [100, rows - 100]
])

# Compute Homography matrix
```

```
H, status = cv2.findHomography(src_points, dst_points)

# Apply Homography transformation
homography_img = cv2.warpPerspective(img, H, (cols, rows))

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

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

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

plt.show()
```

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

```
Saving image.ppm to image (9).ppm
```




```
from google.colab import files
uploaded = files.upload()
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Get uploaded image name
img_name = list(uploaded.keys())[0]

# Read image
img = cv2.imread(img_name)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
rows, cols, ch = img.shape

# Source points (original)
src_pts = np.array([
    [0, 0],
    [cols-1, 0],
    [cols-1, rows-1],
    [0, rows-1]
], dtype=np.float32)

# Destination points (mapped)
dst_pts = np.array([
    [80, 60],
    [cols-120, 40],
    [cols-60, rows-80],
    [100, rows-100]
], dtype=np.float32)

# Function to compute Homography using DLT
def compute_homography(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)
        return H / H[2,2]  # Normalize

# Compute Homography
H = compute_homography(src_pts, dst_pts)

# Apply transformation
dlt_img = cv2.warpPerspective(img, H, (cols, rows))

# Display
plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.title("Original Image")
plt.imshow(img)
plt.axis("off")

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

Choose Files image.ppm

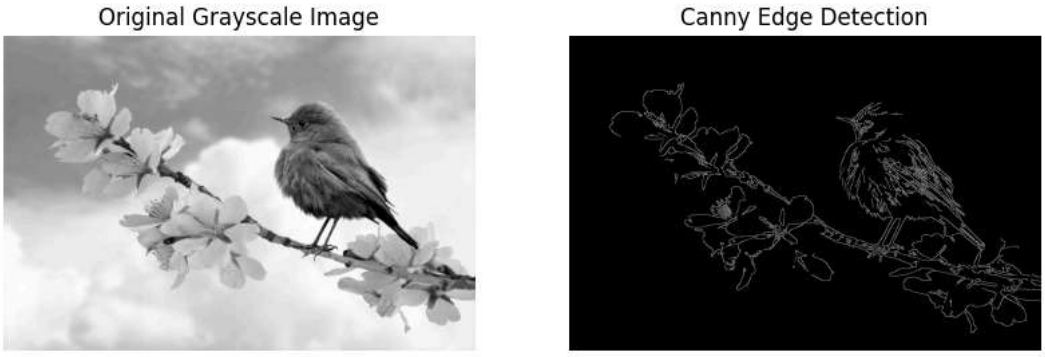
image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

```
Saving image.ppm to image (10).ppm
```


```
from google.colab import files
uploaded = files.upload()
import cv2
import matplotlib.pyplot as plt

# Get uploaded image name
img_name = list(uploaded.keys())[0]

# Read image in grayscale
img = cv2.imread(img_name, cv2.IMREAD_GRAYSCALE)

# Apply Gaussian blur (optional, improves edge detection)
blur = cv2.GaussianBlur(img, (5,5), 1.4)

# Apply Canny edge detection
edges = cv2.Canny(blur, threshold1=50, threshold2=150)

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

plt.subplot(1,2,1)
plt.title("Original Grayscale Image")
plt.imshow(img, cmap='gray')
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 image.ppm
image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done
Saving image.ppm to image (11).ppm




```
from google.colab import files
uploaded = files.upload()
import cv2
import matplotlib.pyplot as plt
import numpy as np

# Get uploaded image name
img_name = list(uploaded.keys())[0]

# Read image in grayscale
img = cv2.imread(img_name, cv2.IMREAD_GRAYSCALE)

# Apply Sobel operator along X axis
sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)

# Convert to 8-bit image for display
sobelx = cv2.convertScaleAbs(sobelx)

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

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

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

plt.show()
```

Choose Files image.ppm
image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done
Saving image.ppm to image (12).ppm


```
from google.colab import files
uploaded = files.upload()
```

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

# Get uploaded image name
img_name = list(uploaded.keys())[0]

# Read image in grayscale
img = cv2.imread(img_name, cv2.IMREAD_GRAYSCALE)

# Apply Sobel operator along Y axis
sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)

# Convert to 8-bit image for display
sobely = cv2.convertScaleAbs(sobely)

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

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

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

plt.show()
```

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

```
Saving image.ppm to image (13).ppm
```

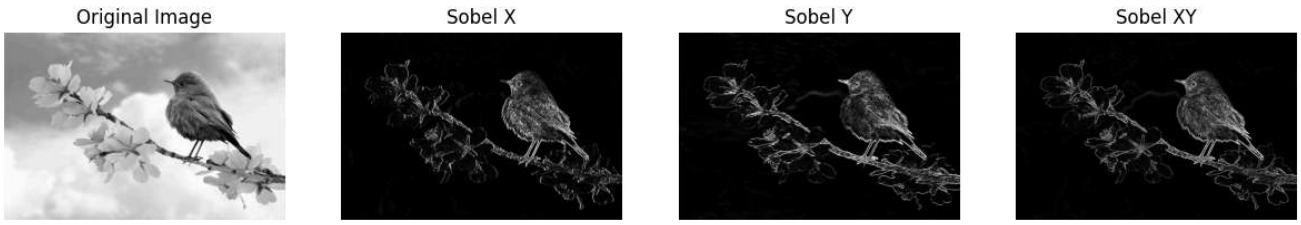

```
from google.colab import files
uploaded = files.upload()
import cv2
import matplotlib.pyplot as plt
import numpy as np

# Get uploaded image name
img_name = list(uploaded.keys())[0]

# Read image in grayscale
img = cv2.imread(img_name, cv2.IMREAD_GRAYSCALE)

# Sobel operator along X-axis
sobelx = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)
sobelx = cv2.convertScaleAbs(sobelx)

# Sobel operator along Y-axis
sobely = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)
sobely = cv2.convertScaleAbs(sobely)

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

# Display results
plt.figure(figsize=(15,4))
```

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

plt.subplot(1,4,2)
plt.title("Sobel X")
plt.imshow(sobelx, cmap='gray')
plt.axis("off")

plt.subplot(1,4,3)
plt.title("Sobel Y")
plt.imshow(sobely, cmap='gray')
plt.axis("off")

plt.subplot(1,4,4)
plt.title("Sobel XY")
plt.imshow(sobelxy, cmap='gray')
plt.axis("off")

plt.show()
```

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

```
Saving image.ppm to image (14).ppm
```



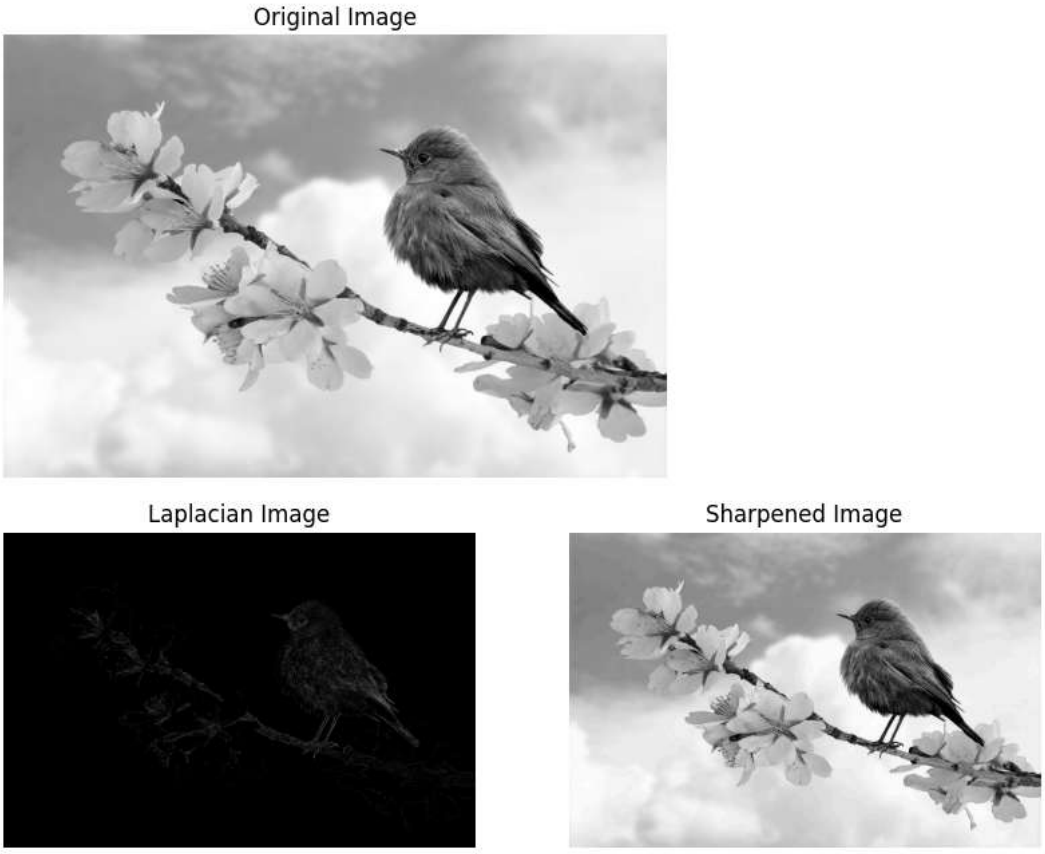

```
# Question 16: Laplacian Sharpening with negative center coefficient

# Step 1: Upload image
from google.colab import files
uploaded = files.upload()

# Step 2: Import libraries
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Step 3: Read image
image_path = list(uploaded.keys())[0]
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Step 4: Display original image
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis('off')

# Step 5: Define Laplacian mask (-4 center)
laplacian_mask = np.array([[0, 1, 0],
                           [1, -4, 1],
                           [0, 1, 0]])

# Step 6: Apply Laplacian filter
laplacian_img = cv2.filter2D(img, -1, laplacian_mask)

# Step 7: Sharpen image
sharpened_img = cv2.subtract(img, laplacian_img)

# Step 8: Display results
plt.figure(figsize=(10,4))

plt.subplot(1,2,1)
plt.imshow(laplacian_img, cmap='gray')
plt.title("Laplacian Image")
plt.axis('off')
```

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

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

```
Saving image.ppm to image (15).ppm
(np.float64(-0.5), np.float64(1279.5), np.float64(852.5), np.float64(-0.5))
```



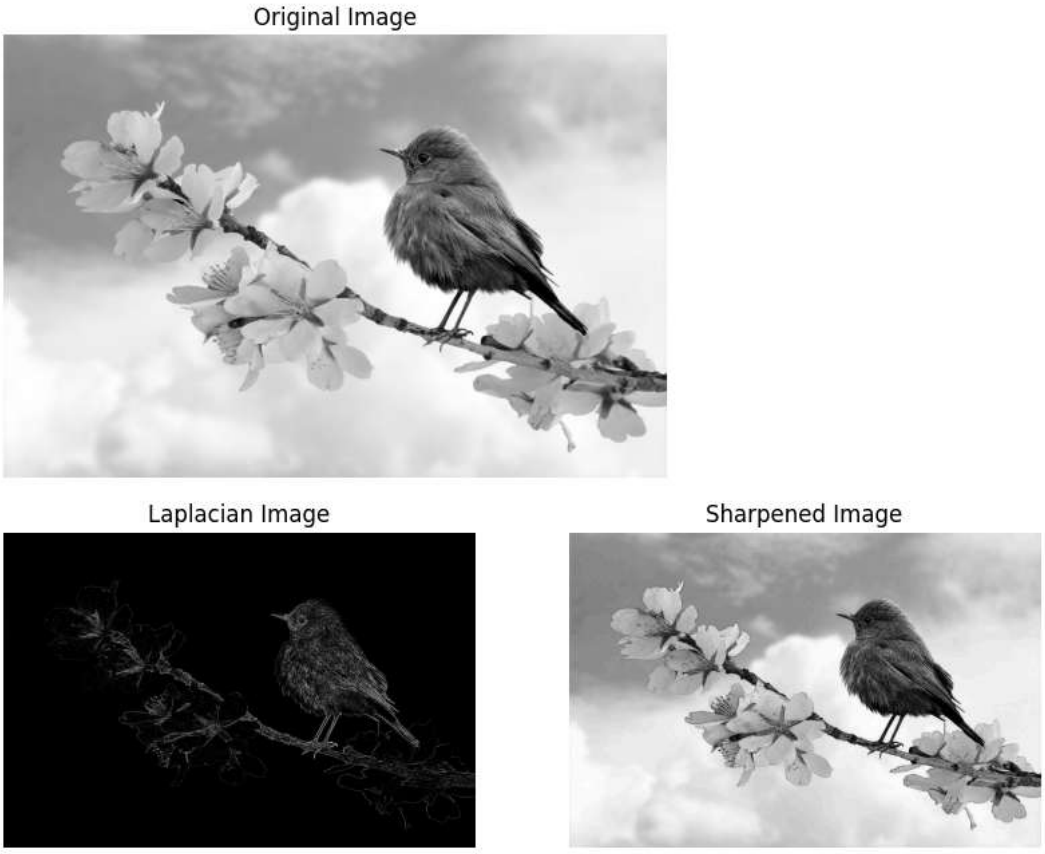

```
# Question 17: Laplacian Sharpening with diagonal neighbors

# Step 1: Upload image
from google.colab import files
uploaded = files.upload()

# Step 2: Import libraries
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Step 3: Read image
image_path = list(uploaded.keys())[0]
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Step 4: Display original image
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis('off')

# Step 5: Define Laplacian mask (-8 center)
laplacian_mask = np.array([[1, 1, 1],
                           [1, -8, 1],
                           [1, 1, 1]])

# Step 6: Apply Laplacian filter
laplacian_img = cv2.filter2D(img, -1, laplacian_mask)

# Step 7: Sharpen image
sharpened_img = cv2.subtract(img, laplacian_img)
```

```
# Step 8: Display results
plt.figure(figsize=(10,4))

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

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

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

```
Saving image.ppm to image (16).ppm
(np.float64(-0.5), np.float64(1279.5), np.float64(852.5), np.float64(-0.5))
```




```
# Question 18: Laplacian Sharpening with positive center coefficient

# Step 1: Upload image
from google.colab import files
uploaded = files.upload()

# Step 2: Import libraries
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Step 3: Read image
image_path = list(uploaded.keys())[0]
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Step 4: Display original image
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis('off')

# Step 5: Define Laplacian mask (+5 center)
laplacian_mask = np.array([[0, -1, 0],
                           [-1, 5, -1],
```

```
                              [0, -1, 0]])

# Step 6: Apply mask directly (sharpening)
sharpened_img = cv2.filter2D(img, -1, laplacian_mask)

# Step 7: Display sharpened image
plt.figure(figsize=(5,4))
plt.imshow(sharpened_img, cmap='gray')
plt.title("Sharpened Image (Positive Center Laplacian)")
plt.axis('off')
```

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

```
Saving image.ppm to image (17).ppm
(np.float64(-0.5), np.float64(1279.5), np.float64(852.5), np.float64(-0.5))
```


```
# Question 19: Image Sharpening using Unsharp Masking

# Step 1: Upload image
from google.colab import files
uploaded = files.upload()

# Step 2: Import libraries
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Step 3: Read image
image_path = list(uploaded.keys())[0]
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Step 4: Display original image
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis('off')

# Step 5: Blur the image
blurred_img = cv2.GaussianBlur(img, (5,5), 0)

# Step 6: Unsharp masking
sharpened_img = cv2.subtract(img, blurred_img)
```

```
# Step 7: 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(blurred_img, cmap='gray')
plt.title("Blurred Image")
plt.axis('off')

plt.subplot(1,3,3)
plt.imshow(sharpened_img, cmap='gray')
plt.title("Sharpened Image (Unsharp Masking)")
plt.axis('off')
```

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

```
Saving image.ppm to image (18).ppm
(np.float64(-0.5), np.float64(1279.5), np.float64(852.5), np.float64(-0.5))
```


```
# Question 20: Image Sharpening using High-Boost Mask

# Step 1: Upload image
from google.colab import files
uploaded = files.upload()

# Step 2: Import libraries
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Step 3: Read image
image_path = list(uploaded.keys())[0]
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Step 4: Display original image
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis('off')

# Step 5: Define High-Boost Laplacian mask (4-neighbor)
A = 2   # High-boost factor (A >= 1)
```

```
high_boost_mask = np.array([[0, -1, 0],
                            [-1, A+4, -1],
                            [0, -1, 0]])

# Step 6: Apply High-Boost filter
sharpened_img = cv2.filter2D(img, -1, high_boost_mask)

# Step 7: Display sharpened image
plt.figure(figsize=(5,4))
plt.imshow(sharpened_img, cmap='gray')
plt.title("High-Boost Sharpened Image")
plt.axis('off')
```

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

```
Saving image.ppm to image (19).ppm
(np.float64(-0.5), np.float64(1279.5), np.float64(852.5), np.float64(-0.5))
```

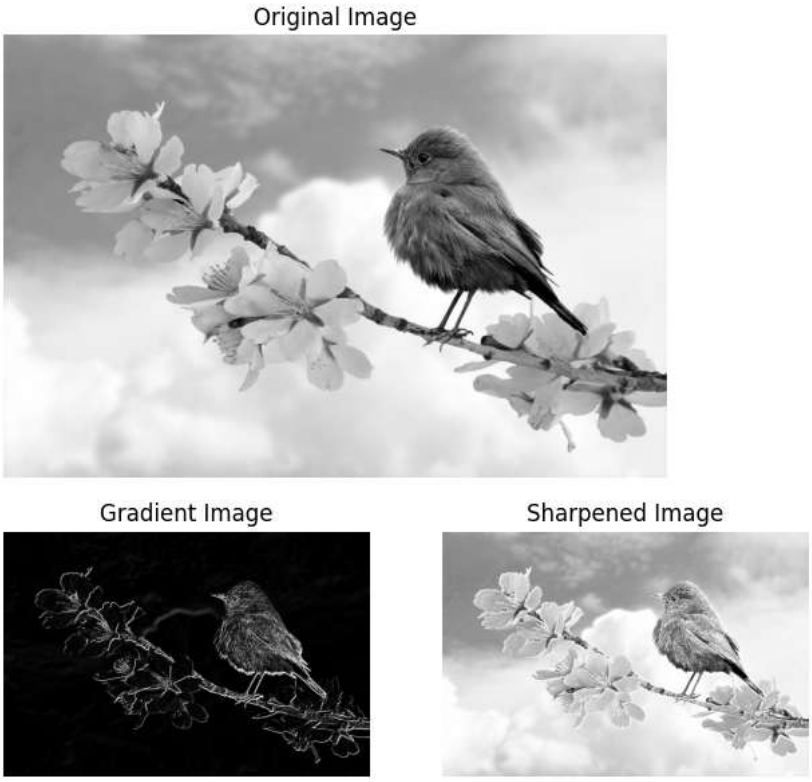

```
# Question 21: Image Sharpening using Gradient Masking

# Step 1: Upload image
from google.colab import files
uploaded = files.upload()

# Step 2: Import libraries
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Step 3: Read image
image_path = list(uploaded.keys())[0]
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Step 4: Display original image
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis('off')

# Step 5: Define gradient masks (Sobel)
Gx = np.array([[-1, 0, 1]
```

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

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

# Step 6: Apply gradient masks
grad_x = cv2.filter2D(img, -1, Gx)
grad_y = cv2.filter2D(img, -1, Gy)

# Step 7: Gradient magnitude
gradient = cv2.add(np.abs(grad_x), np.abs(grad_y))

# Step 8: Sharpen image
sharpened_img = cv2.add(img, gradient)

# Step 9: Display results
plt.figure(figsize=(12,4))

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

plt.subplot(1,3,2)
plt.imshow(sharpened_img, cmap='gray')
plt.title("Sharpened Image")
plt.axis('off')
```

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

```
Saving image.ppm to image (20).ppm
(np.float64(-0.5), np.float64(1279.5), np.float64(852.5), np.float64(-0.5))
```




```
# Question 22: Insert watermark into an image using OpenCV

# Step 1: Upload image
from google.colab import files
uploaded = files.upload()

# Step 2: Import libraries
import cv2
import numpy as np
```

```
from matplotlib import pyplot as plt

# Step 3: Read image
image_path = list(uploaded.keys())[0]
img = cv2.imread(image_path)

# Step 4: Add watermark text
watermarked_img = img.copy()
text = "WATERMARK"
position = (50, 50)

cv2.putText(watermarked_img, text, position,
            cv2.FONT_HERSHEY_SIMPLEX,
            1, (255, 255, 255), 2, cv2.LINE_AA)

# Step 5: Display image
plt.imshow(cv2.cvtColor(watermarked_img, cv2.COLOR_BGR2RGB))
plt.title("Watermarked Image")
plt.axis('off')
```

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

```
Saving image.ppm to image (21).ppm
(np.float64(-0.5), np.float64(1279.5), np.float64(852.5), np.float64(-0.5))
```


```
# Question 23: Boundary detection using convolution kernel

# Step 1: Upload image
from google.colab import files
uploaded = files.upload()

# Step 2: Import libraries
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Step 3: Read image
image_path = list(uploaded.keys())[0]
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# Step 4: Threshold image
_, binary = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

# Step 5: Define structuring element
kernel = np.ones((3,3), np.uint8)

# Step 6: Erosion
eroded = cv2.erode(binary, kernel, iterations=1)

# Step 7: Boundary extraction
boundary = cv2.subtract(binary, eroded)

# Step 8: Display results
plt.figure(figsize=(12,4))

plt.subplot(1,3,1)
plt.imshow(binary, cmap='gray')
```

```
plt.imshow(binary, cmap='gray')
plt.title("Binary Image")
plt.axis('off')

plt.subplot(1,3,2)
plt.imshow(eroded, cmap='gray')
plt.title("Eroded Image")
plt.axis('off')

plt.subplot(1,3,3)
plt.imshow(boundary, cmap='gray')
plt.title("Boundary Image")
plt.axis('off')
```

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

```
Saving image.ppm to image (22).ppm
(np.float64(-0.5), np.float64(1279.5), np.float64(852.5), np.float64(-0.5))
```


```
# Question 24: Morphological operation - Erosion

from google.colab import files
uploaded = files.upload()

import cv2
import numpy as np
from matplotlib import pyplot as plt

image_path = list(uploaded.keys())[0]
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

_, binary = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

kernel = np.ones((3,3), np.uint8)
eroded = cv2.erode(binary, kernel, iterations=1)

plt.figure(figsize=(10,4))

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

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

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

Saving image.ppm to image (23).ppm

(np.float64(-0.5), np.float64(1279.5), np.float64(852.5), np.float64(-0.5))

```
# Question 25: Morphological operation - Dilation

from google.colab import files
uploaded = files.upload()

import cv2
import numpy as np
from matplotlib import pyplot as plt

image_path = list(uploaded.keys())[0]
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

_, binary = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

kernel = np.ones((3,3), np.uint8)
dilated = cv2.dilate(binary, kernel, iterations=1)

plt.figure(figsize=(10,4))

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

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

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

Saving image.ppm to image (24).ppm

(np.float64(-0.5), np.float64(1279.5), np.float64(852.5), np.float64(-0.5))


```
# Question 26: Morphological operation - Opening

from google.colab import files
uploaded = files.upload()

import cv2
import numpy as np
from matplotlib import pyplot as plt

image_path = list(uploaded.keys())[0]
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

_, binary = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

kernel = np.ones((3,3), np.uint8)
opening = cv2.morphologyEx(binary, cv2.MORPH_OPEN, kernel)

plt.figure(figsize=(10,4))

plt.subplot(1,2,1)
plt.imshow(binary, cmap='gray')
plt.title("Binary Image")
plt.axis('off')
```

```
plt.subplot(1,2,2)
plt.imshow(opening, cmap='gray')
plt.title("Opening Result")
plt.axis('off')
```

Choose Files image.ppm
image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done
Saving image.ppm to image (25).ppm
(np.float64(-0.5), np.float64(1279.5), np.float64(852.5), np.float64(-0.5))


```
# Question 27: Morphological operation - Closing

from google.colab import files
uploaded = files.upload()

import cv2
import numpy as np
from matplotlib import pyplot as plt

image_path = list(uploaded.keys())[0]
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

_, binary = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

kernel = np.ones((3,3), np.uint8)
closing = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel)

plt.figure(figsize=(10,4))

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

plt.subplot(1,2,2)
plt.imshow(closing, cmap='gray')
plt.title("Closing Result")
plt.axis('off')
```

Choose Files image.ppm
image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done
Saving image.ppm to image (26).ppm
(np.float64(-0.5), np.float64(1279.5), np.float64(852.5), np.float64(-0.5))

```
# Question 28: Morphological operation - Morphological Gradient

from google.colab import files
uploaded = files.upload()

import cv2
import numpy as np
from matplotlib import pyplot as plt

image_path = list(uploaded.keys())[0]
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

_, binary = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

kernel = np.ones((3,3), np.uint8)
gradient = cv2.morphologyEx(binary, cv2.MORPH_GRADIENT, kernel)

plt.figure(figsize=(10,4))

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

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

Choose Files image.ppm

image.ppm(n/a) - 168056 bytes, last modified: 12/16/2025 - 100% done

```
Saving image.ppm to image (27).ppm
(np.float64(-0.5), np.float64(1279.5), np.float64(852.5), np.float64(-0.5))
```