GIFT©
UNIVERSITY

Sufyan Ahmad
201980013
Assignment 02
Computer Vision

Introduction of Edge Detection:

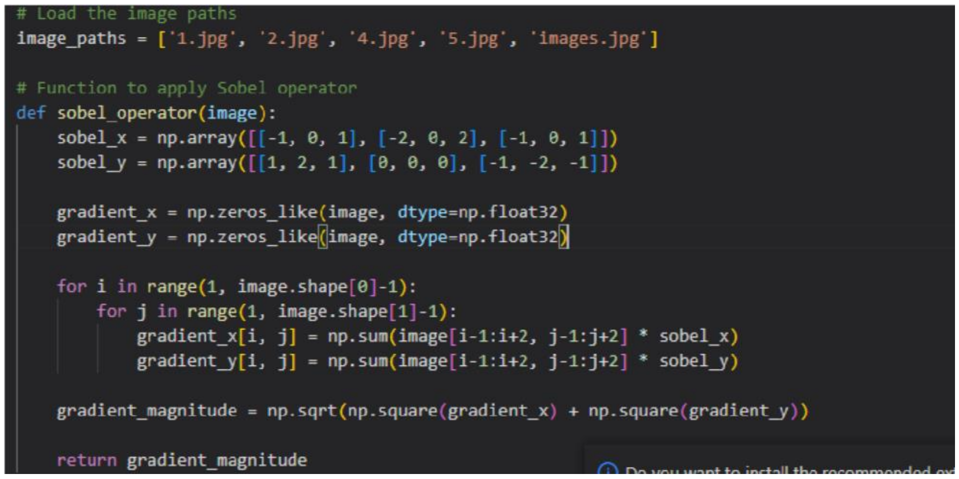
Edge detection is like finding lines where an image changes a lot. This can happen when colors, textures, or light suddenly shift. It's important for fixing pictures. It helps find where things start and end in an image.This is really useful for computer vision tasks like spotting objects, cutting images into parts, and getting important details. The main goal of edge detection tools is to highlight places in a picture where there are big changesin color or where things suddenly stop or start.

Edge Filter Description:

In computer vision, when we want to find the edges in a picture, we use special tools called filters or operators.These tools look at each tiny part of the picture and decide if there might be an edge there.

Sobel Operator:

The Sobel operator is a well-known trick we use to discover edges in images. It's like a detective looking at each little dot in the picture and checking if the colors are shifting. It uses two special patterns that it slides across the picture-one for changes that go up and down, and the other for changes that go from side to side.Then, it combines the results from these patterns to tell us how much the colors are changing.This is super helpful because it shows us where the edges are, whether they're going up and down or side to side.



```
# Load the image paths
image_paths = ['1.jpg', '2.jpg', '4.jpg', '5.jpg', 'images.jpg']

# Function to apply Sobel operator
def sobel_operator(image):
    sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_y = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])

    gradient_x = np.zeros_like(image, dtype=np.float32)
    gradient_y = np.zeros_like(image, dtype=np.float32)

    for i in range(1, image.shape[0]-1):
        for j in range(1, image.shape[1]-1):
            gradient_x[i, j] = np.sum(image[i-1:i+2, j-1:j+2] * sobel_x)
            gradient_y[i, j] = np.sum(image[i-1:i+2, j-1:j+2] * sobel_y)

    gradient_magnitude = np.sqrt(np.square(gradient_x) + np.square(gradient_y))

    return gradient_magnitude
```

Prewitt Operator:

Just like the Sobel operator, the Prewitt operator is another way to figure out edges in images. It also tries to measure how the colors change. It uses two special shapes, one for changes that go sideways and the other for changes that go up and down. It then adds up the measurements from these shapes to show how much the colors are changing.The Prewitt operator is easier to use on computers, but it might not work as well as the Sobel operator in some cases.

Laplacian:

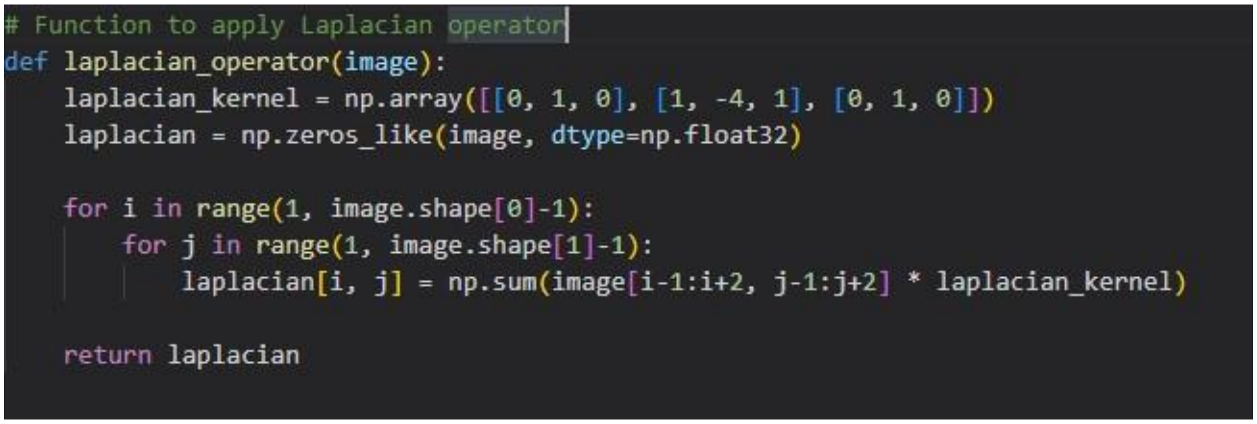
1-First, change the picture to black and white.

2-Then, use a special trick called the Laplacian operator on the black and white picture.

3-This trick helps us see where the colors change a lot in the picture.

4-After that, we can decide which parts of the picture are edges and which parts are not by looking at the results from the trick.

5-If you want, you can make the edges look even better by doing some extra steps, like removing noise or making the edges thinner.



```python
# Function to apply Laplacian operator
def laplacian_operator(image):
    laplacian_kernel = np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]])
    laplacian = np.zeros_like(image, dtype=np.float32)

    for i in range(1, image.shape[0]-1):
        for j in range(1, image.shape[1]-1):
            laplacian[i, j] = np.sum(image[i-1:i+2, j-1:j+2] * laplacian_kernel)

    return laplacian
```

Canny:

The Canny edge detector is a multi-stage edge detection algorithm that provides robust and accurate results. It involves the following steps:

Smoothing: At first, we make the picture a bit smoother using a special filter.This helps get rid of tiny specks and false edges that can happen because of small color changes.

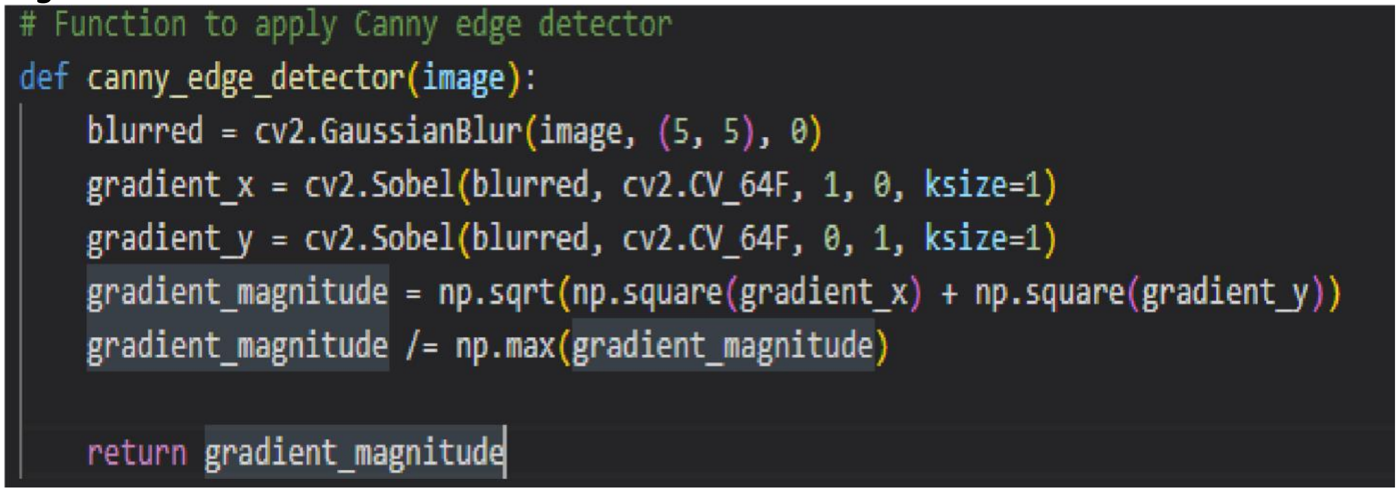
Gradient Calculation: Next, we look at how fast the colors are changing at each spot in the picture.This helps us find where the edges might be.

Non-maximum Suppression: Then, we make the edges look even clearer.We do this by keeping only the biggest color changes in each direction.

Thresholding:After that, we use two levels to decide if a spot is part of an edge or not.This helps us get rid of edges that aren't really important.

Hysteresis: Lastly, we try to connect the edges so they look like complete lines.We do this by looking at the strong edges and seeing if the weaker edges next to them are part of the same line.This final step aims to connect weak edges to strong edges by considering connectivity. If a weak edge pixel is connected to a strong edge pixel, it is considered part of the edge. This helps in completing fragmented edges.

Edge Detection Process:



```
# Function to apply Canny edge detector
def canny_edge_detector(image):
    blurred = cv2.GaussianBlur(image, (5, 5), 0)
    gradient_x = cv2.Sobel(blurred, cv2.CV_64F, 1, 0, ksize=1)
    gradient_y = cv2.Sobel(blurred, cv2.CV_64F, 0, 1, ksize=1)
    gradient_magnitude = np.sqrt(np.square(gradient_x) + np.square(gradient_y))
    gradient_magnitude /= np.max(gradient_magnitude)

    return gradient_magnitude
```

The process of edge detection involves the following steps:

Preprocessing: It is often beneficial to preprocess the image by reducing noise or enhancing relevant features. Techniques such as smoothing, denoising, or contrast adjustment can be applied depending on the specific requirements.

Filtering: Apply an edge detection filter or operator to the preprocessed image. This involves convolving the image with the chosen filter to obtain the gradient magnitude or edge likelihood at each pixel.

Thresholding: Based on the output of the filter, apply a threshold to categorize pixels as edge pixels or non-edge pixels.Pixels with gradient values above a certain threshold are considered as edge pixels.

Post-processing: Optional post-processing steps can be performed to refine the detected edges. Techniques like edge thinning, edge linking, or morphological operations can be employed to improve the edge map.

Comparison:

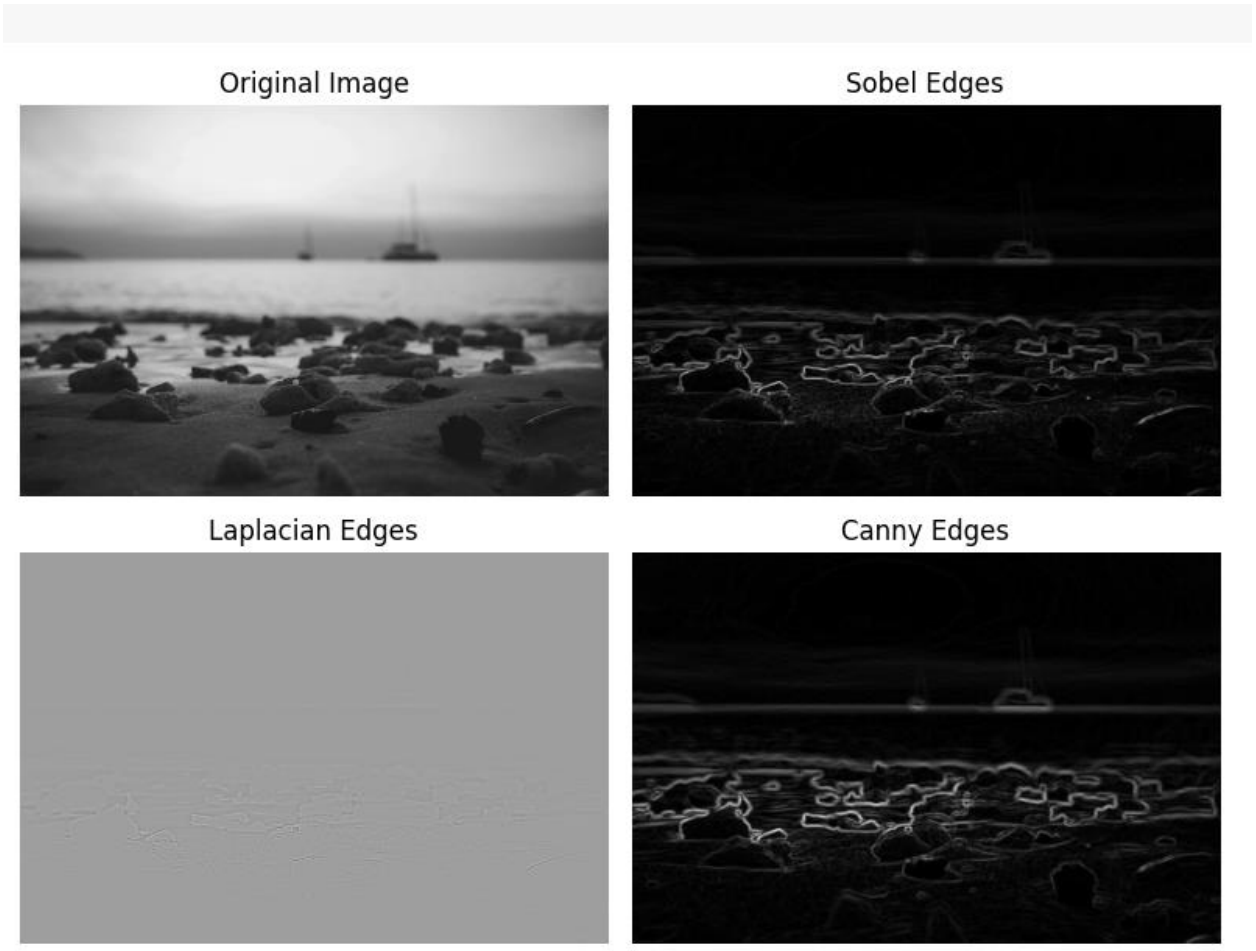
Techniques	Sobel	Laplacian	Canny
	Robust to noise	Emphasizes high-frequency edges	Optimal edge detection
	Computes gradient magnitude and direction	Detects edges regardless of orientation	Handles noise effectively
	Easy to implement	Simple implementation	Suppresses non-maximum edge
	Fast computation		hysteresis thresholding for better results
	Moderate	Moderate	Excellent

Overall best performance:

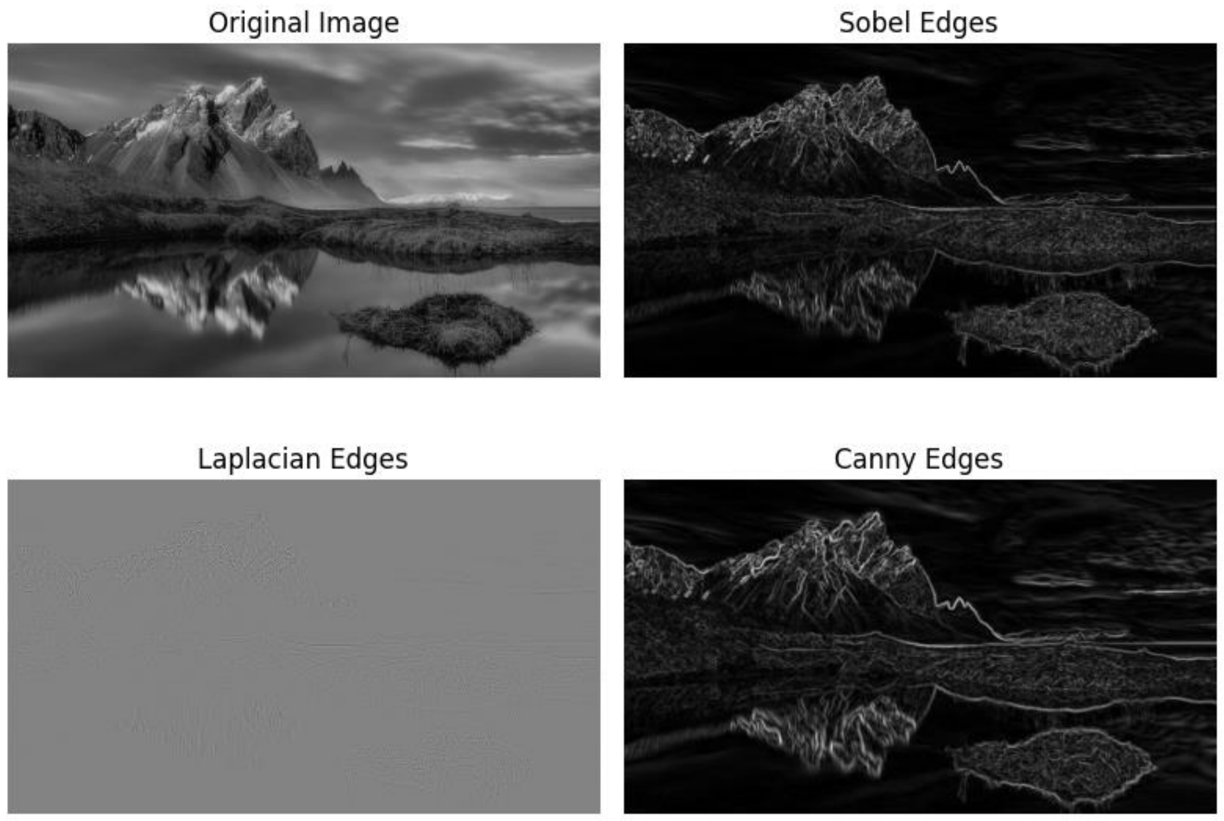
While **Canny** is often considered the most effective overall, Sobel and Laplacian have their strengths and may be more suitable in certain scenarios. Therefore, it is recommended to experiment with different techniques and evaluate their performance based on our specific use case.

Applied Detection techniques on different types of noise images:

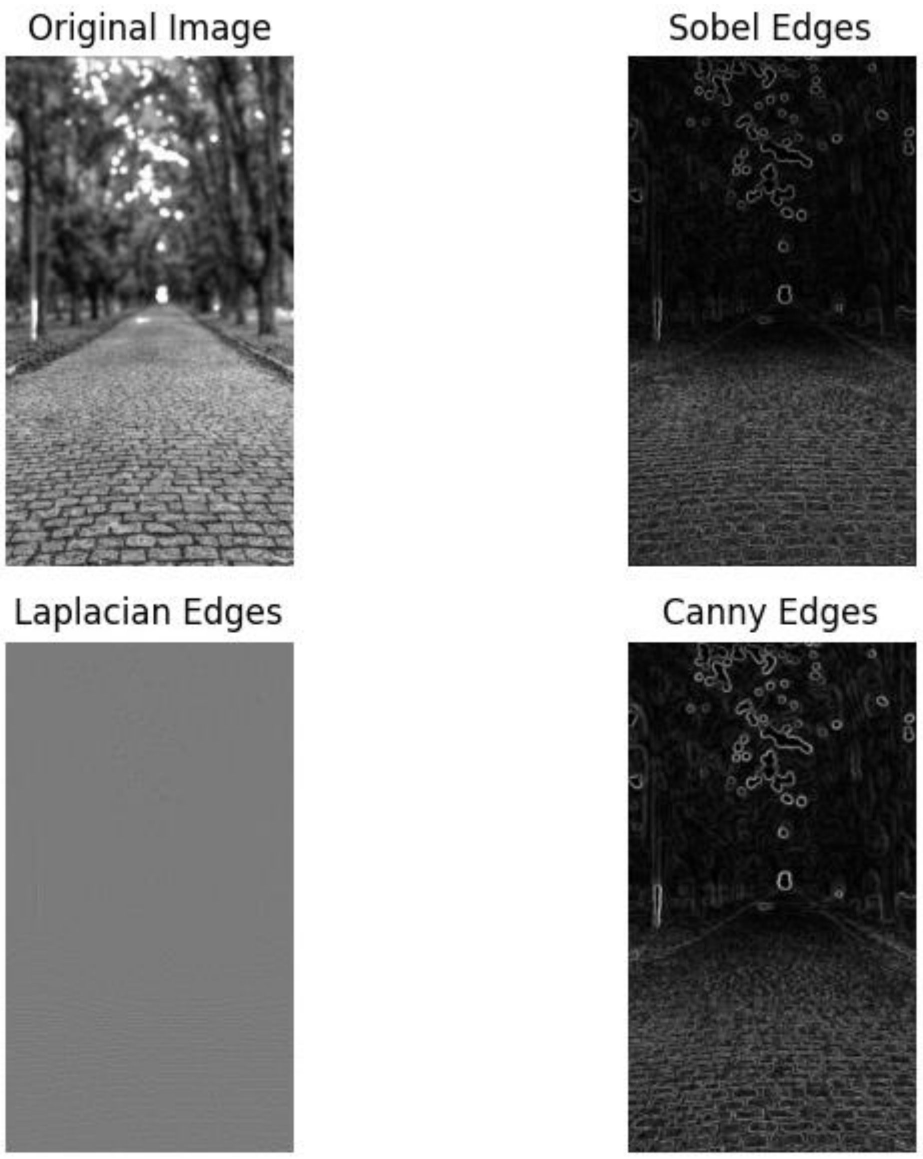
Beach image result:



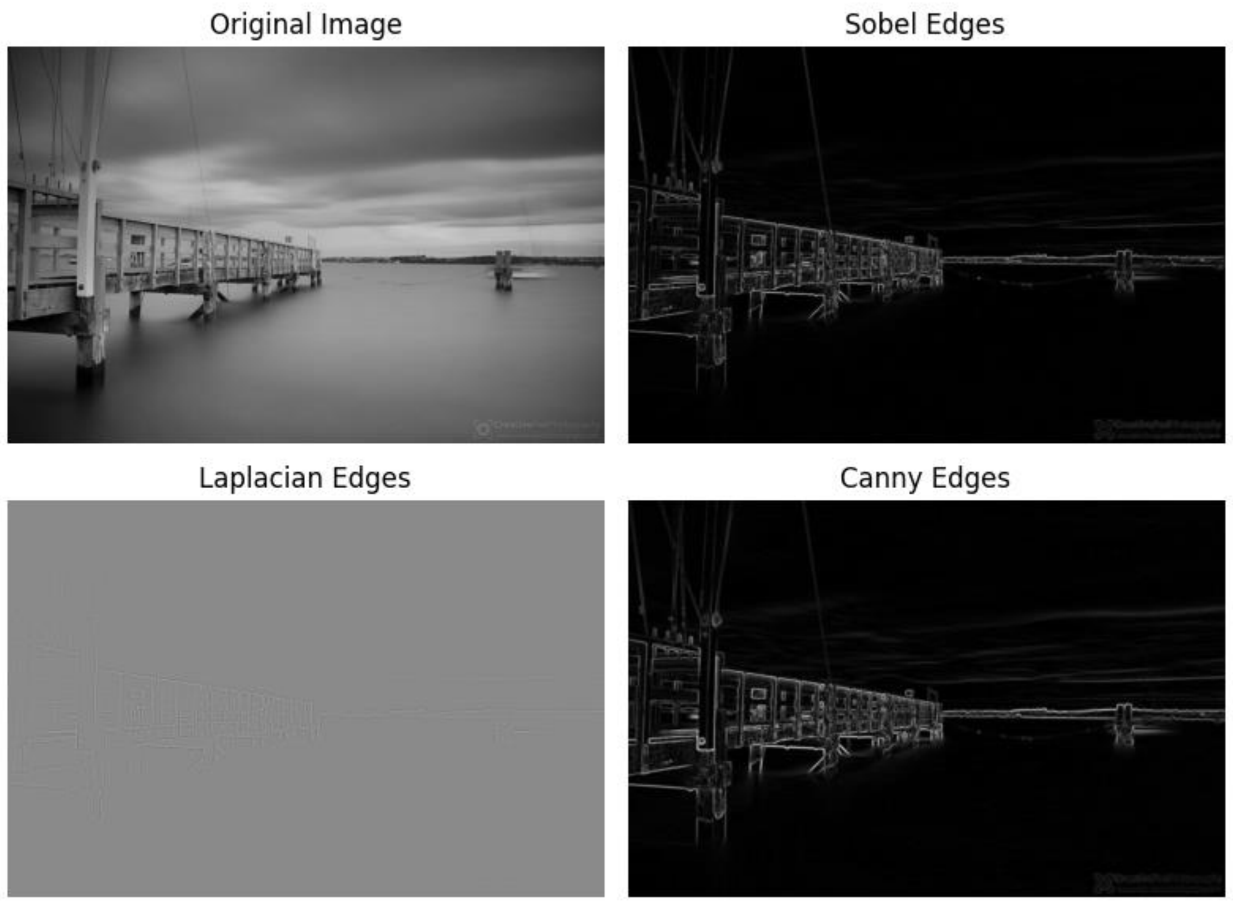
Lake image result:



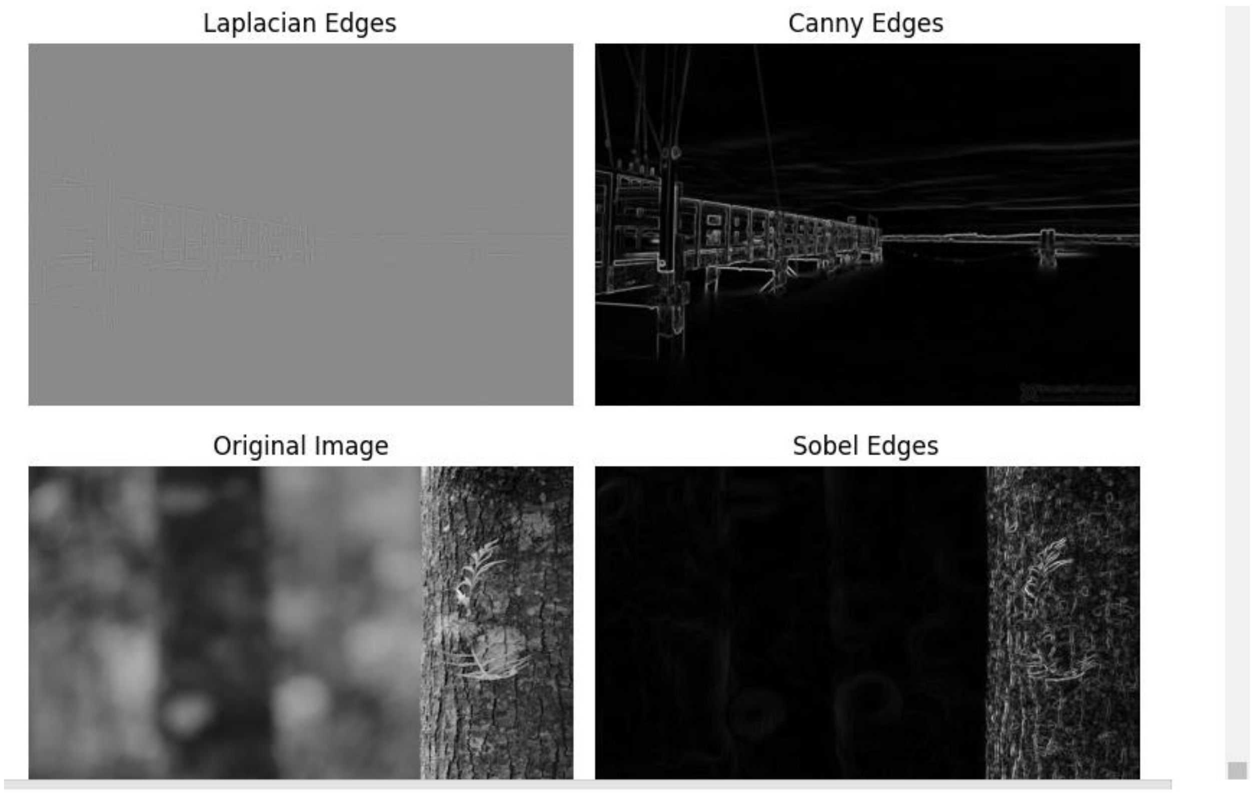
Park image result:



Sea image result:



Tree image result:


Laplacian Edges
Canny Edges
Original Image
Sobel Edges