Project 2

Computer Vision - CS 6643

Out: Oct 09, 2023
Due: Nov 06, 2023 (deadline: 11:59 PM)

A1) Convolution and Derivative Filters

A2) Cross Correlation and Template Matching

A3) Creative Part:

Nidhushan Kanagaraja (N10852881)

A1) Convolution and Derivative Filters

1.1 Introduction

Convolution is used to apply filters to an image. These filters are represented in matrices, which slide over the image and its weighted sum is calculated at each position. It is used for tasks like blurring, edge detection, sharpening and more.

Derivative filters are a type of convolution filter used for approximating derivatives, which represent the rate of change or slope of a function. We use derivative filters to detect edges and highlight regions where pixel intensity changes rapidly.

1.2 Edge Detection Filtering

We have 3 images(1 personal) for which we will try to detect the edges. The original images are given below:

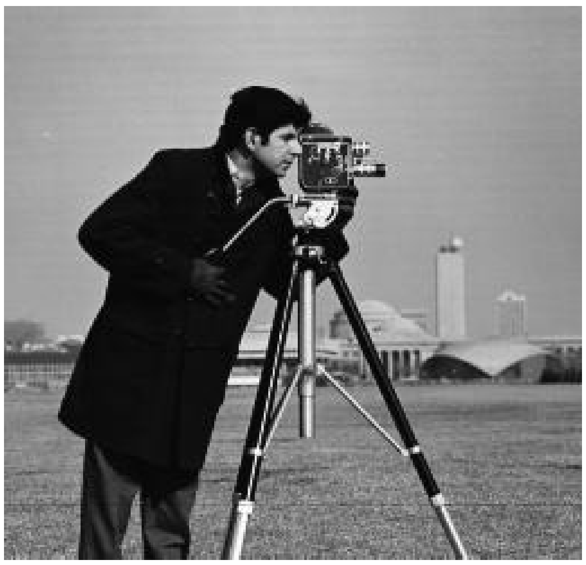


Fig. 1.1 Cameraman - Original Image

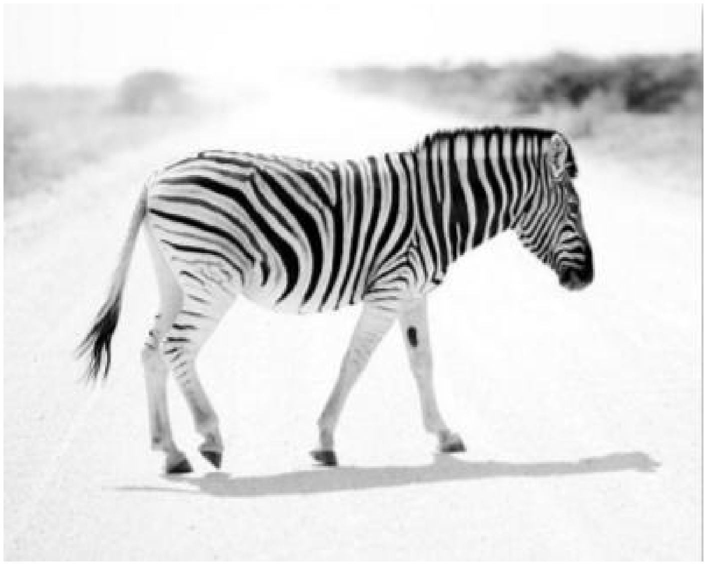


Fig. 1.2 Zebra - Original Image

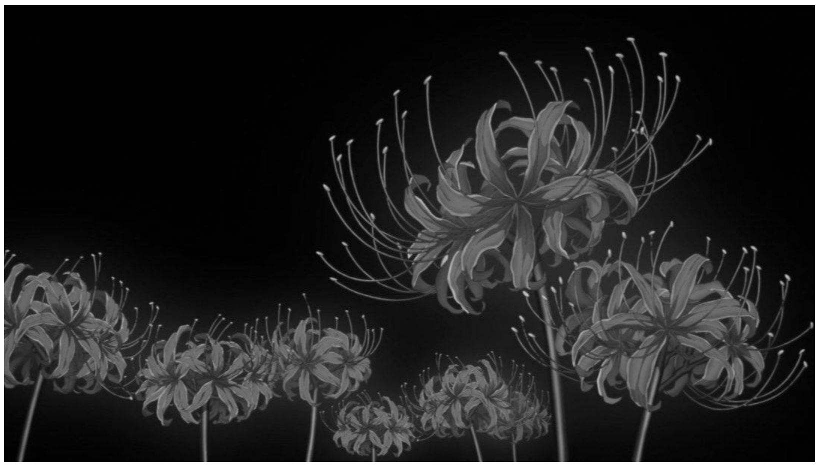


Fig. 1.3 Spider Lily - Original Image

For detecting the edges we start by denoising the image with a Gaussian filter. Then we compute the corresponding derivative image by using a derivative filter. Finally we compute the gradient magnitude image and create the binary edge images, using a best threshold.

1.3 Denoising the image with Gaussian Filter

We start by denoising the image with a 1D Gaussian filter first in the vertical and then horizontal direction. For creating this Gaussian Filter we use the following formula:

$$g_\sigma(u) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(u)^2}{2\sigma^2}\right)$$

For discrete Gauss Filters, u is an integer in the range:

$$u \in [-z, \dots, z],\ where\ usually\ z = \lceil 3\sigma \rceil.$$

We use a Gaussian Filter of size 3:

Gx = [[0.32710442   0.34579116   0.32710442]]

Gy = [[0.32710442]
[0.34579116]
[0.32710442]]

After denoising the image using the Gx filter we get the following results:

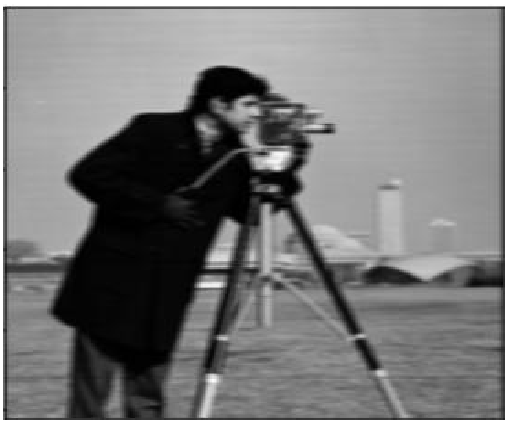


Fig. 1.4 Cameraman after denoising using Gx

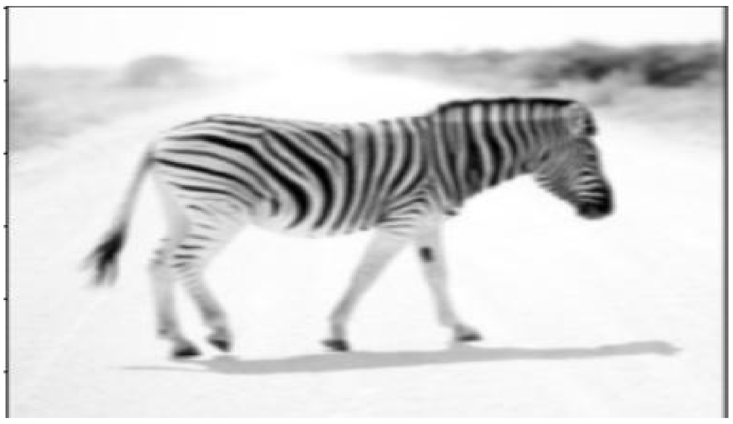


Fig. 1.5 Zebra after denoising using Gx

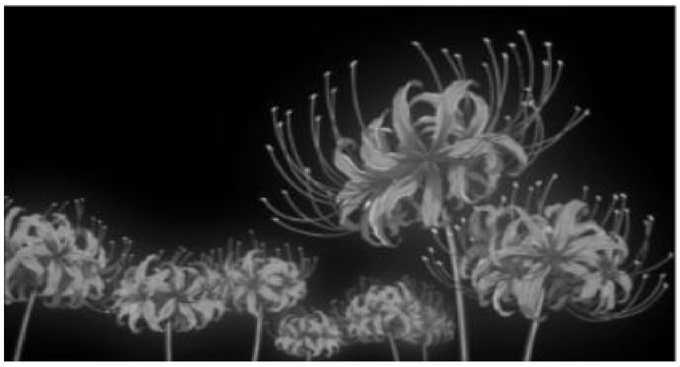


Fig. 1.6 Spider Lily after denoising using Gx

Now we again denoise using the Gy filter on the Gx filtered images to get the following results:

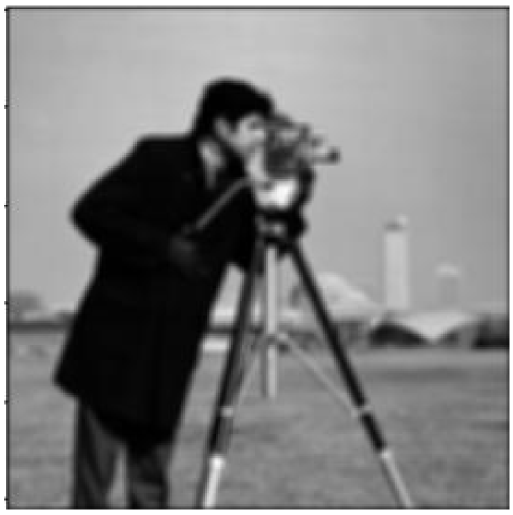


Fig. 1.7 Cameraman after denoising using Gy Filter

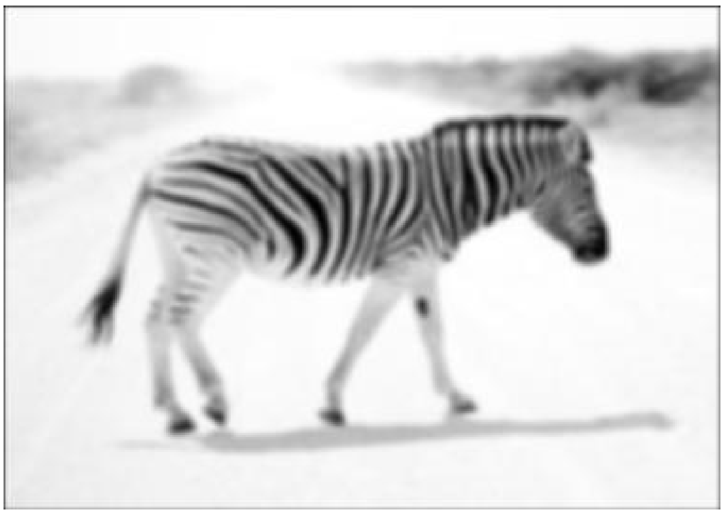


Fig. 1.8 Zebra after denoising using Gy Filter

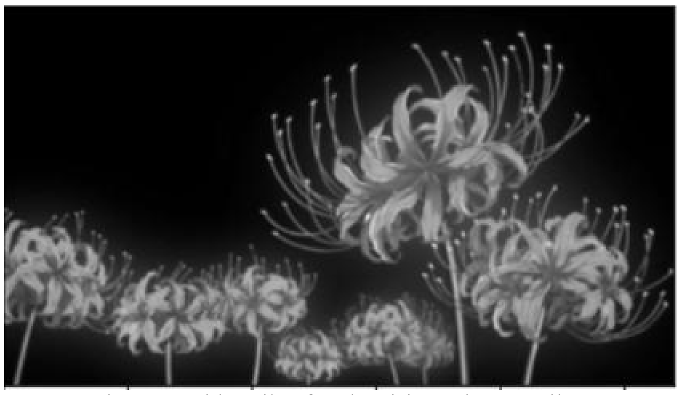


Fig. 1.9 Spider Lily after denoising using Gy Filter

1.4 Computing Derivative Image

We use a derivative filter to compute the Derivative Image. Here we hard code the derivative filter as follows:

Dx = [-1 0 1] and Dy = [-1 0 1]^T

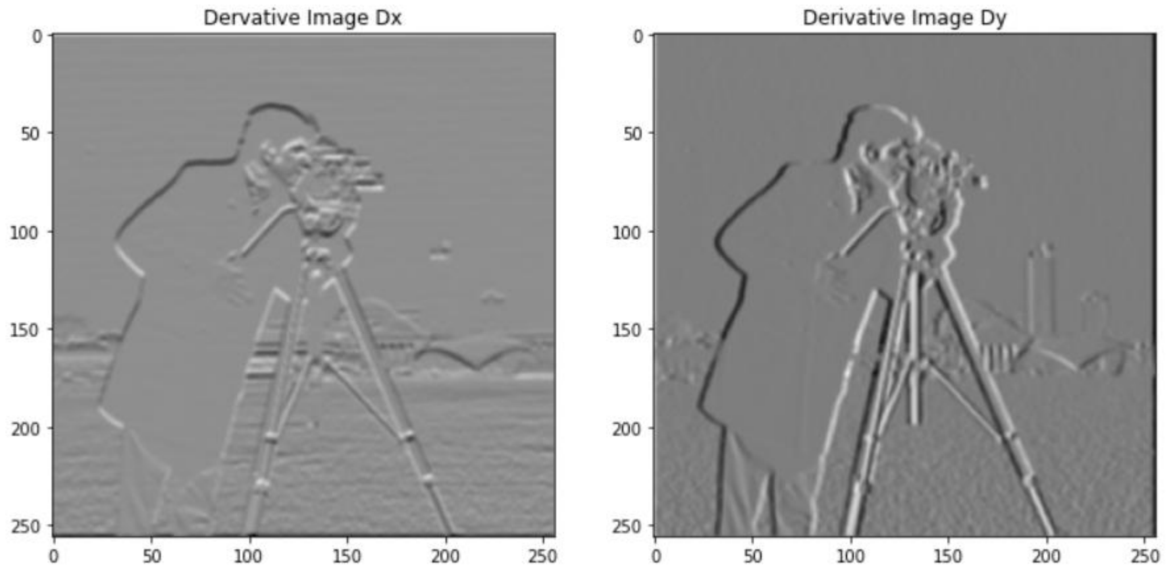


Fig. 1.10 Cameraman image after applying derivative filter Dx and Dy respectively

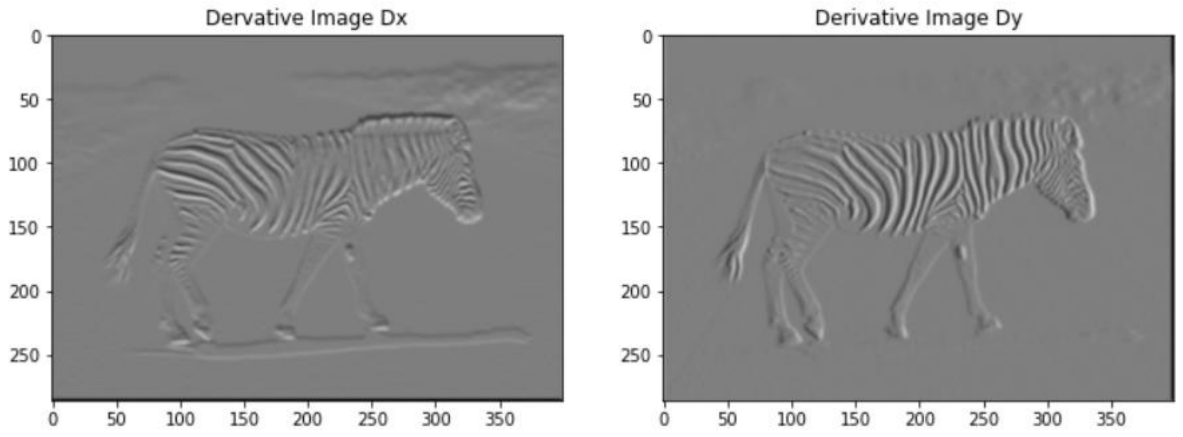


Fig. 1.11 Zebra image after applying derivative filter Dx and Dy respectively

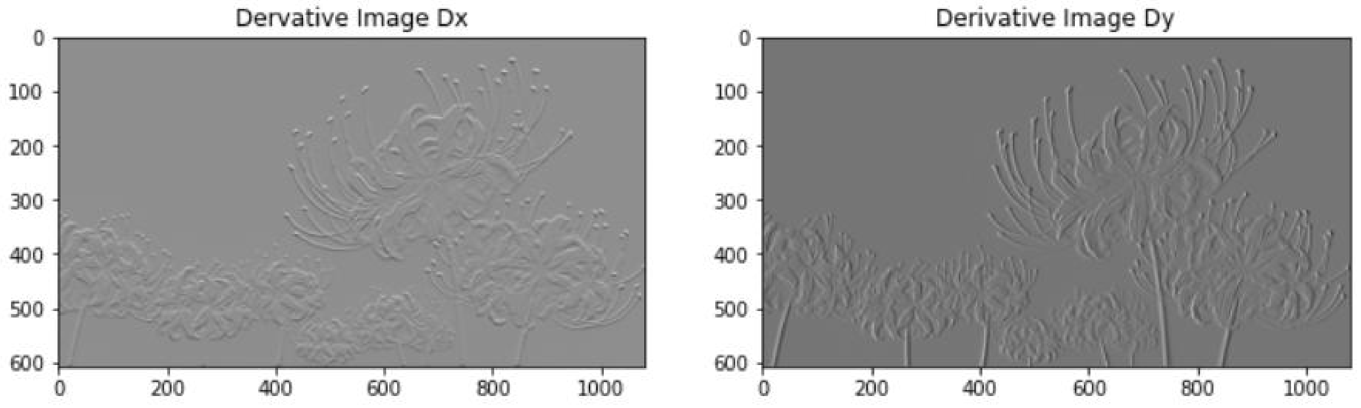


Fig 1.12 Spider Lily Image after applying derivative filter Dx and Dy respectively

Here, we have applied the derivative filters, Dx and Dy to obtain the derivative images of all 3 images.

1.5 Gradient Magnitude Image

The gradient magnitude image is computed in the following way

$\sqrt{(image\_dx)^2 + (image\_dy)^2}$

Where image_dx and image_dy are the images after applying the Dx and Dy derivative filters. The resulting image after computing the gradient magnitude is:

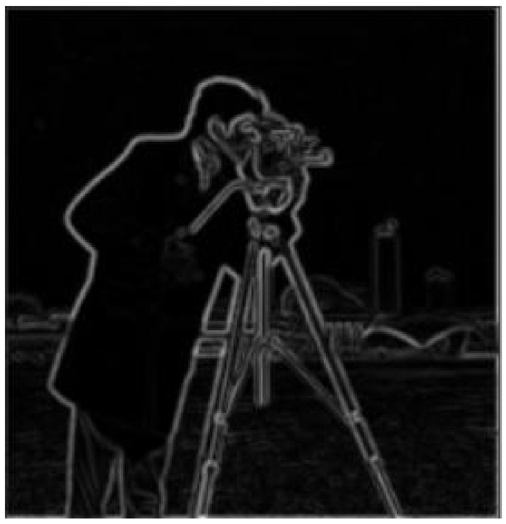


Fig. 1.13 Cameraman image after applying gradient magnitude

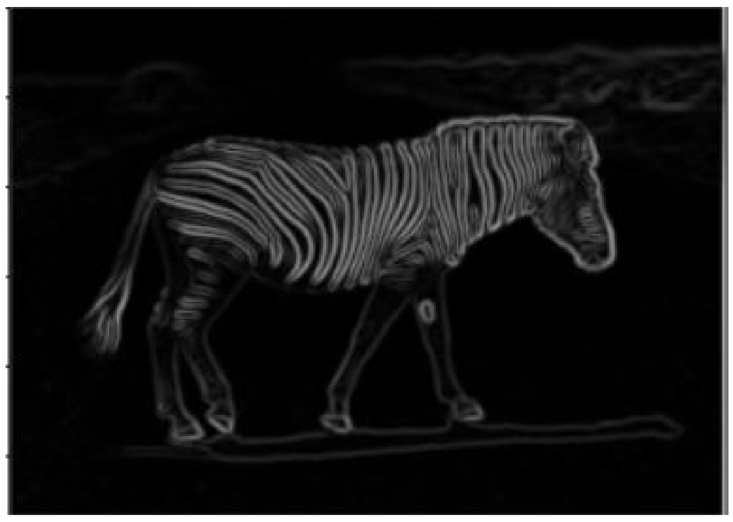


Fig. 1.14 Zebra image after applying gradient magnitude

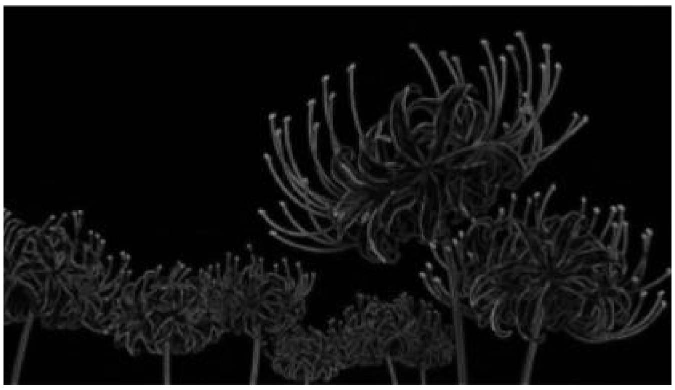


Fig. 1.15 Spider Lily after applying gradient magnitude

1.6 Binary Edge Images

The binary edge image can be computed by setting a threshold for the gradient magnitude image and depending on the threshold we set we we calculate the edges. We find the best case threshold for each image and use it to find the final image. The final results are given below along with its threshold.

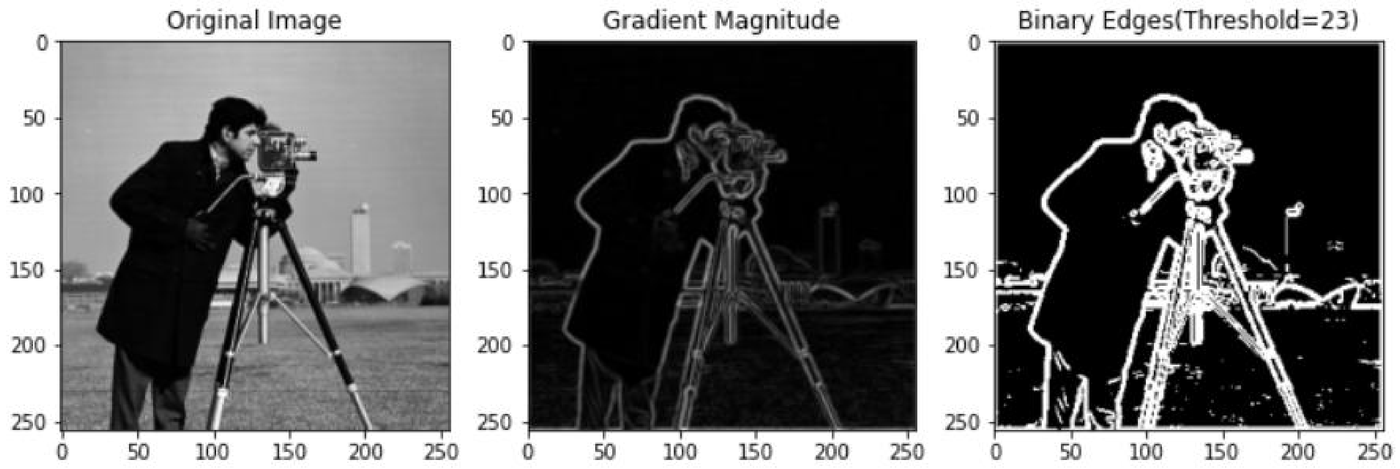


Fig 1.16 Cameraman - comparison of Original Image vs Gradient Magnitude Image vs Binary Edge Image

For the zebra image, we get a good outline of the zebra with a threshold of 14. But this doesn't give us a good edge detection for the zebras face. Depending on our requirement we will have to change the threshold to fulfill our needs. We have a better face with a threshold of 35.

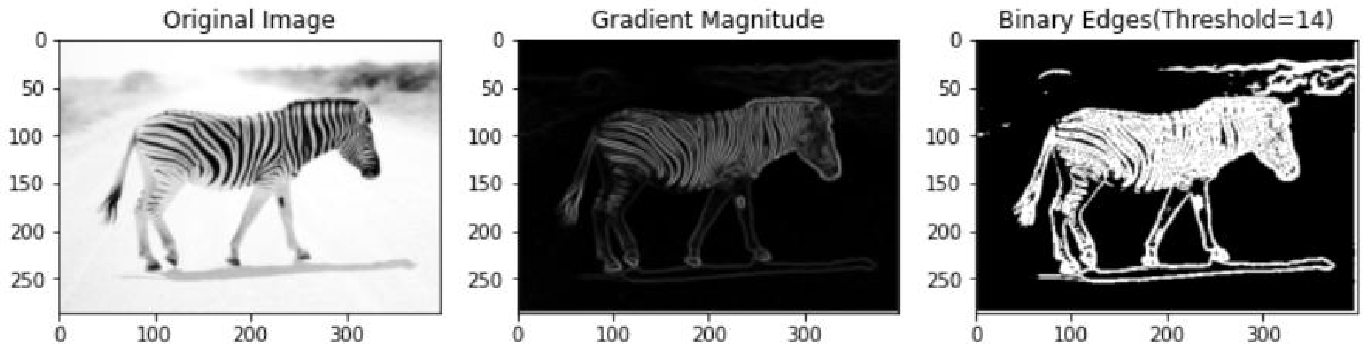


Fig. 1.17 Zebra - comparison of Original Image vs Gradient Magnitude Image vs Binary Edge Image(Threshold=14)

Resulting image when threshold is set to 35 is given below. This gives a better edge description inside the zebra but it has a bad outline, especially the leg and the back of the zebra.

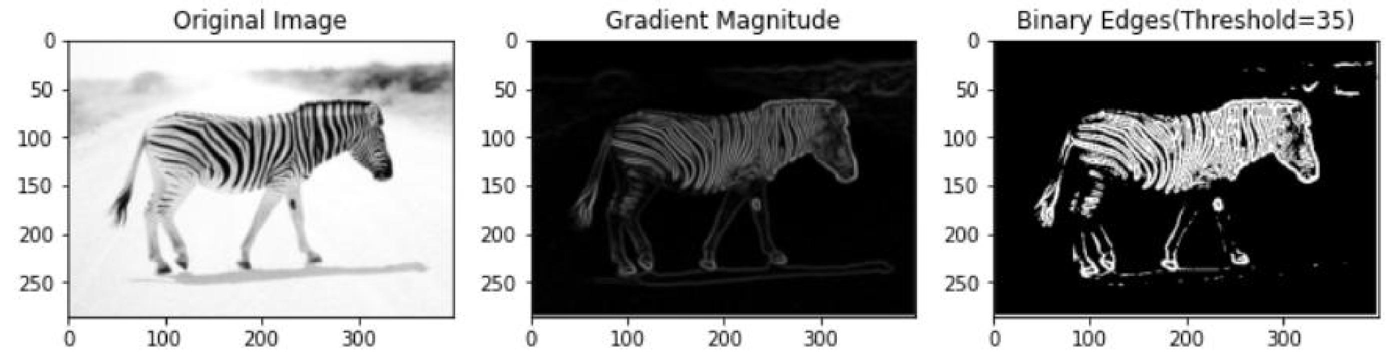


Fig. 1.18 Zebra - comparison of Original Image vs Gradient Magnitude Image vs Binary Edge Image(Threshold=35)

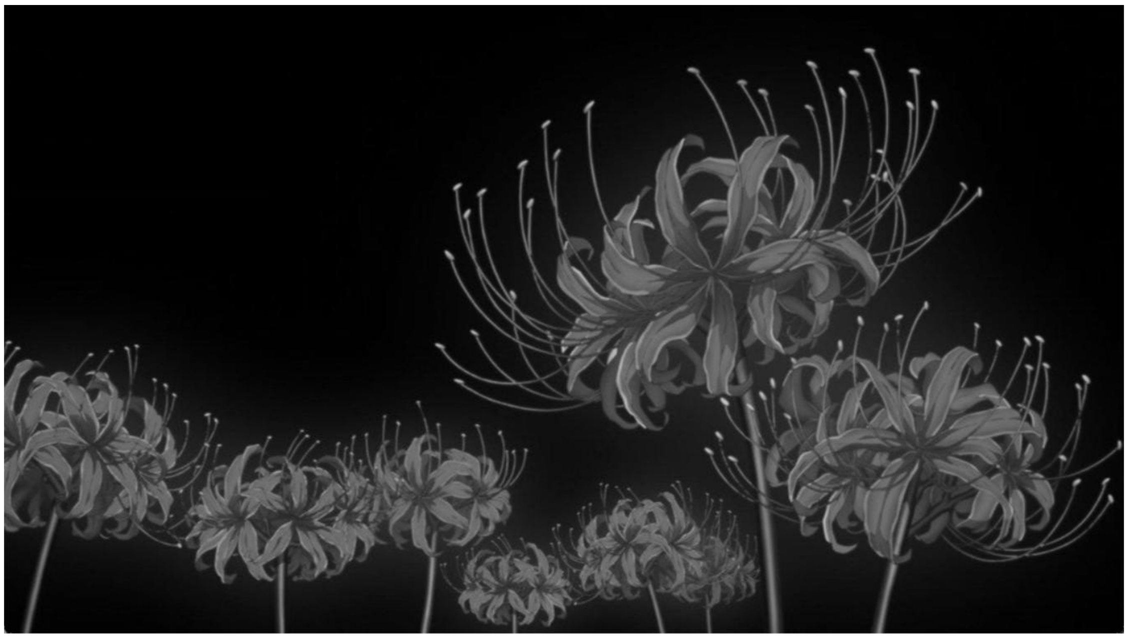


Fig. 1.19 Spider Lily - Original Image

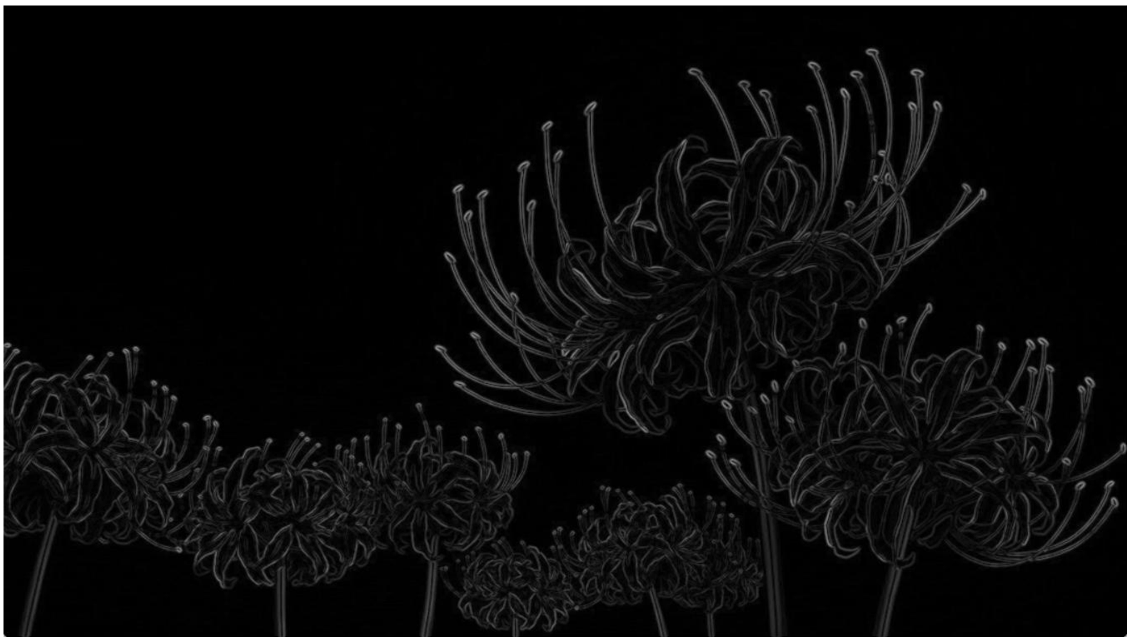


Fig. 1.20 Spider Lily - Gradient Magnitude Image

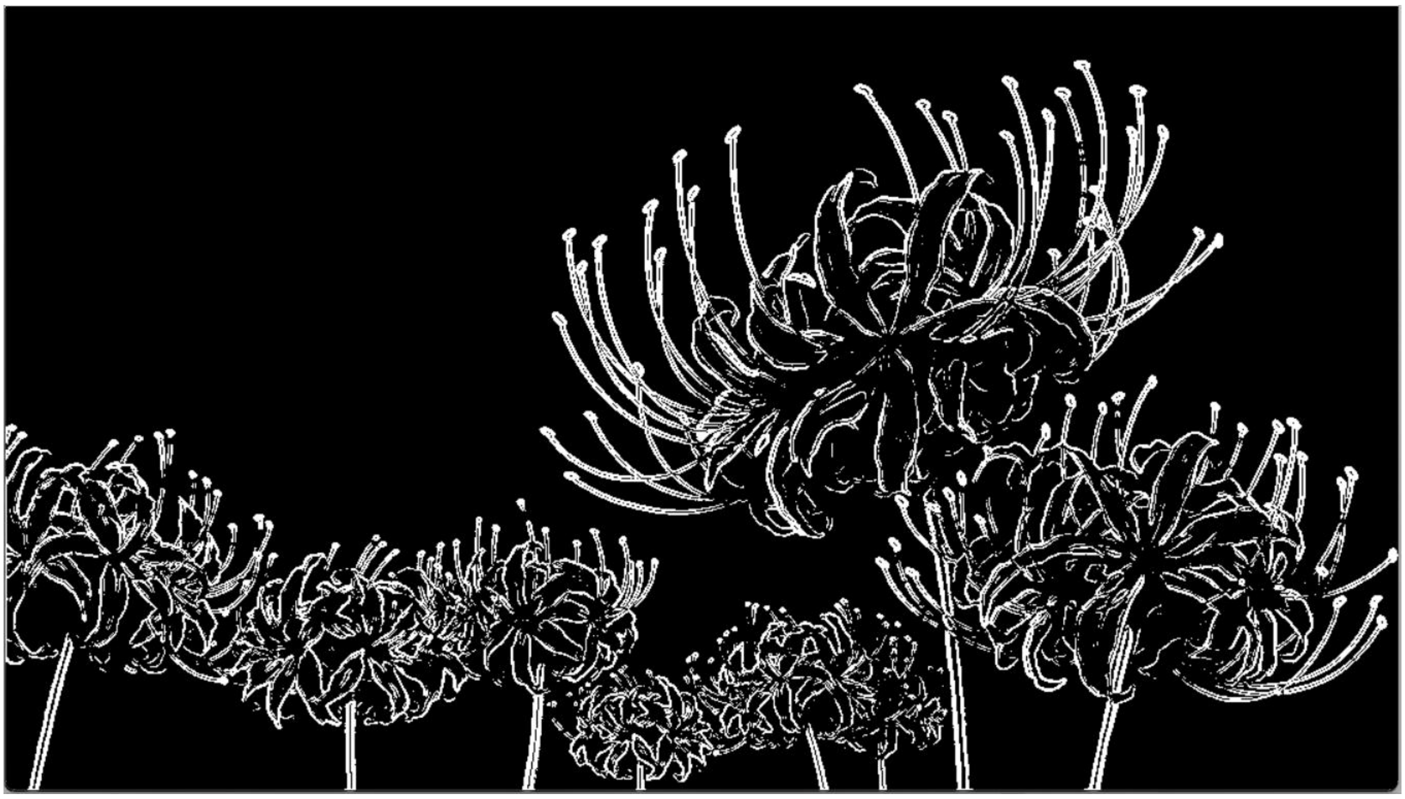


Fig. 1.21 Spider Lily - Binary Edge Image

A2) Cross Correlation and Template Matching

2.1 Introduction

Template matching is the process where we have an image and a template and try to locate the template in the image.

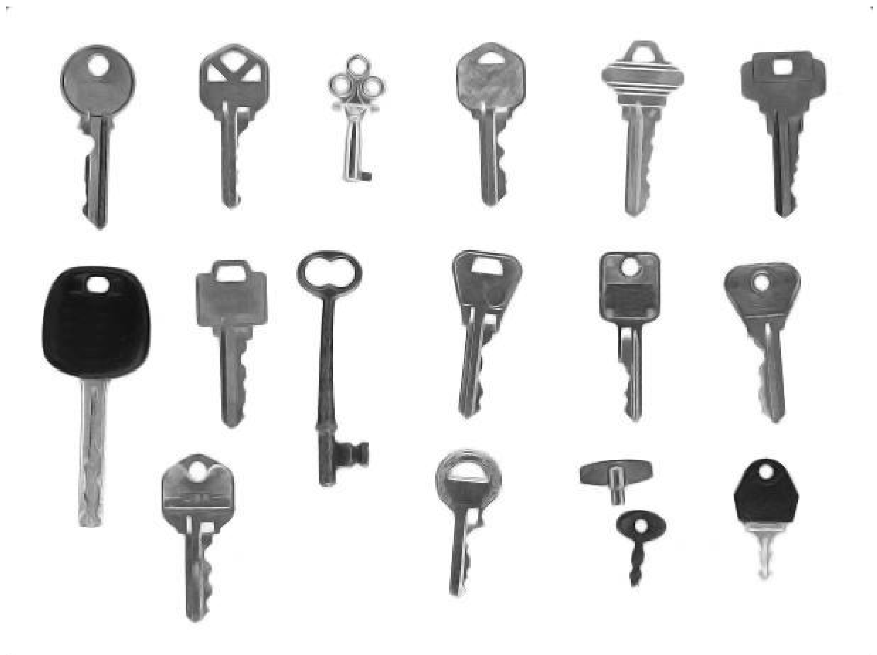


Fig. 2.1 Multiple Keys original image

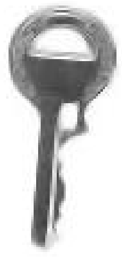


Fig. 2.2 Single Key BW original image(Chosen from multiple keys)

2.2 Cross Correlation

The multiple keys image and single key image are used to implement cross correlation, with the template being the mask.

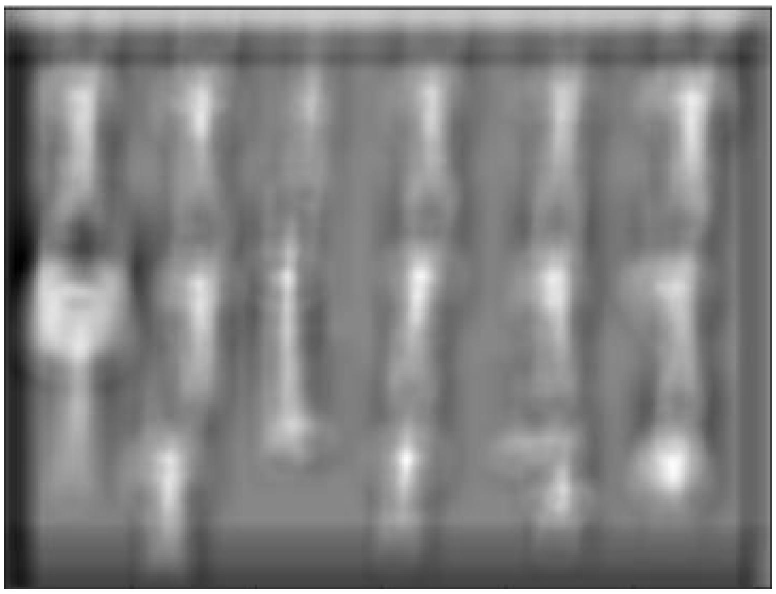


Fig. 2.3 Resulting image from template and original image

In this image, the place where we have the strongest peak will be the best match for the template provided(since template is taken from the same image we will get a perfect match.) The rest of the peaks will be a good match for the template(here key), meaning we can also find the locations of the other keys.

Finding the position of the peak in Fig. 2.3, we get the following result:

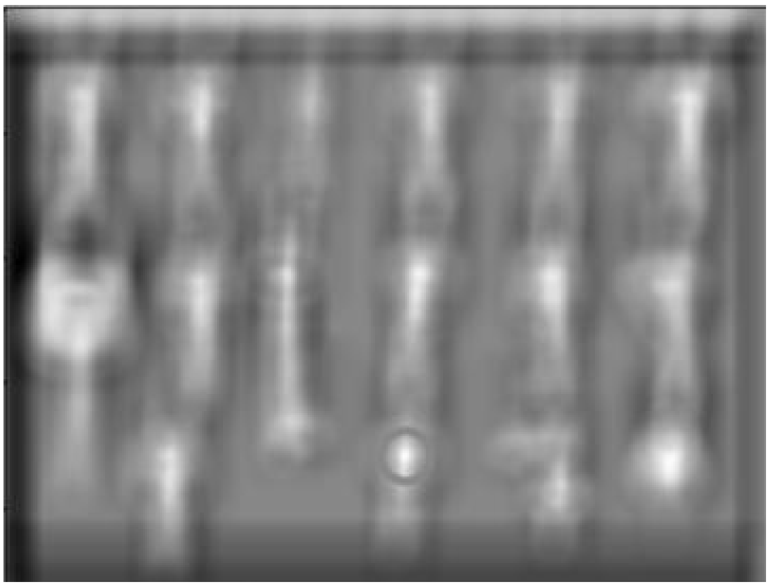


Fig. 2.4 Resulting image after finding the peak

```
Peak matching: 21884380.0
Peak's x coordinate: 321
Peak's y coordinate: 359
```

Matching the location in the original image we get the following result:

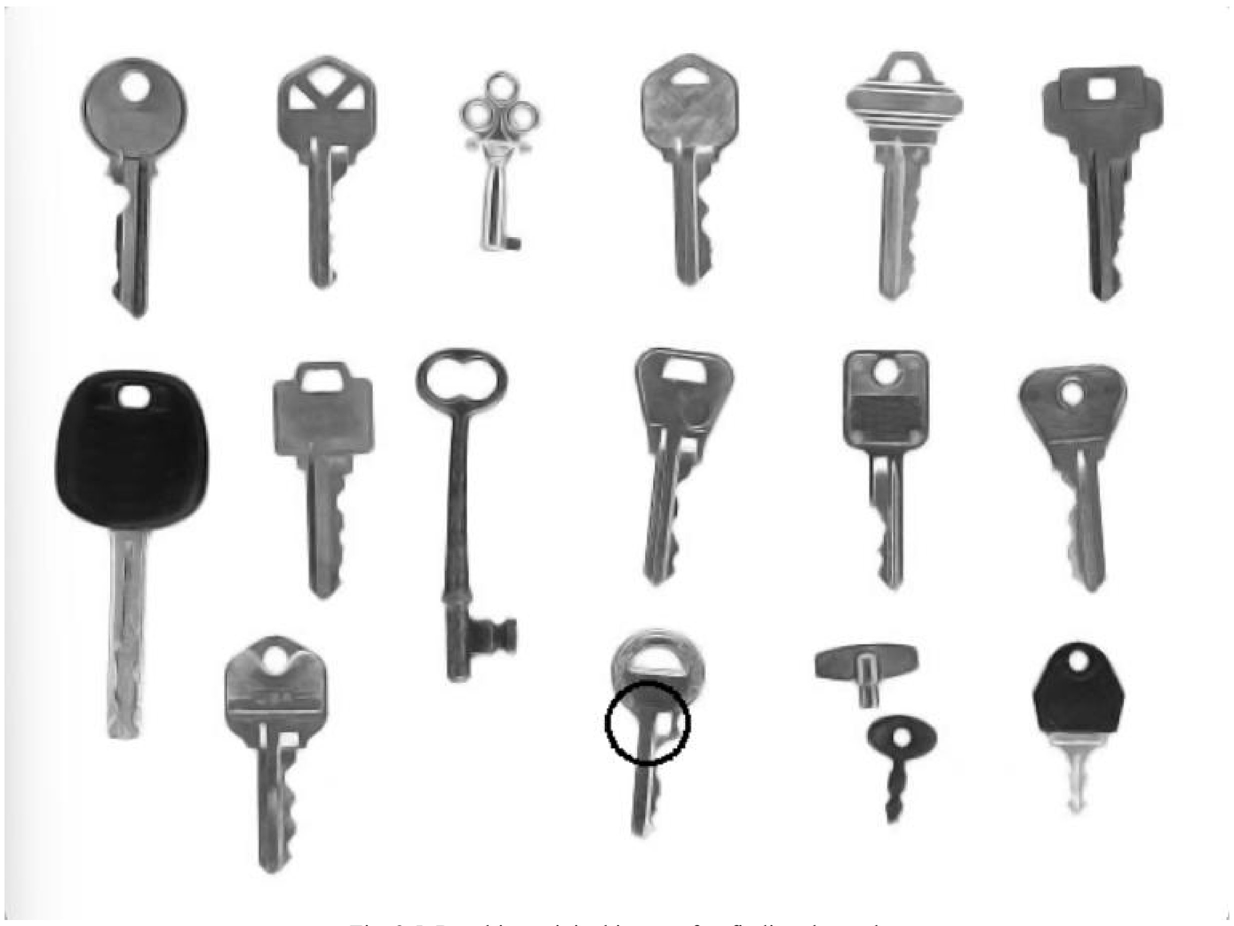


Fig. 2.5 Resulting original image after finding the peak

A3) Creative Part - Color Edge Detection

3.1 Introduction

This will be similar to normal edge detection. We can do this both by converting the image into “LCH” color shape, in python as well as converting the original image to black and white, and using the edge detection discussed in A1.

Images used:

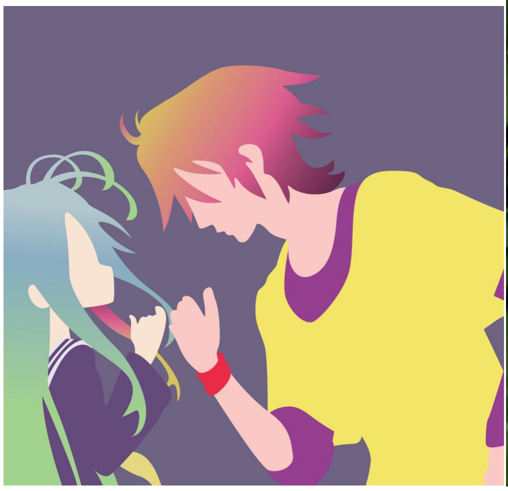


Fig. 3.1 SS - Original Image

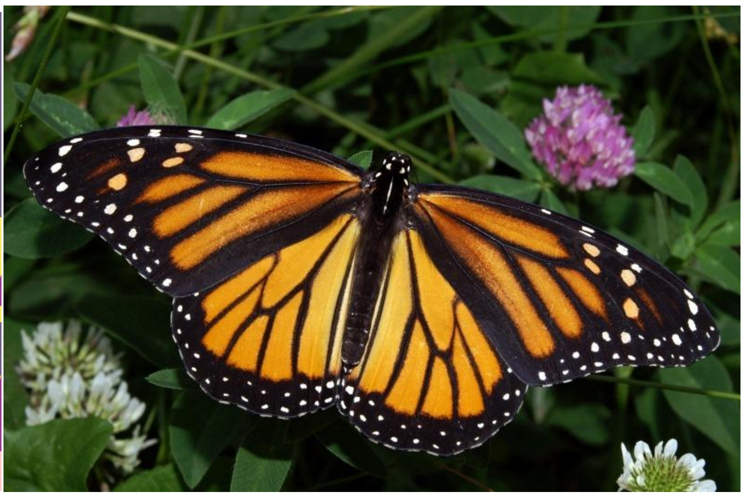


Fig. 3.2 Monarch - Original Image

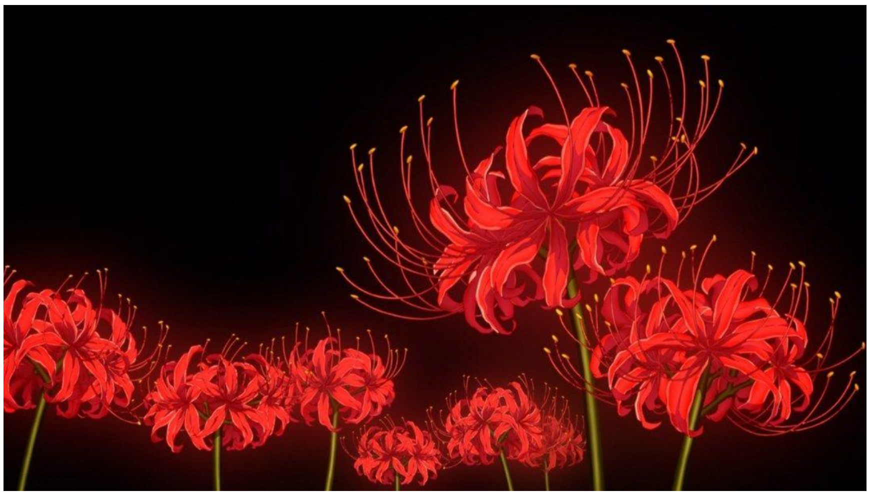


Fig. 3.3 Spider Lily - Original Image

Separate the images into three channels: LCH(Lightness Chroma Hue)

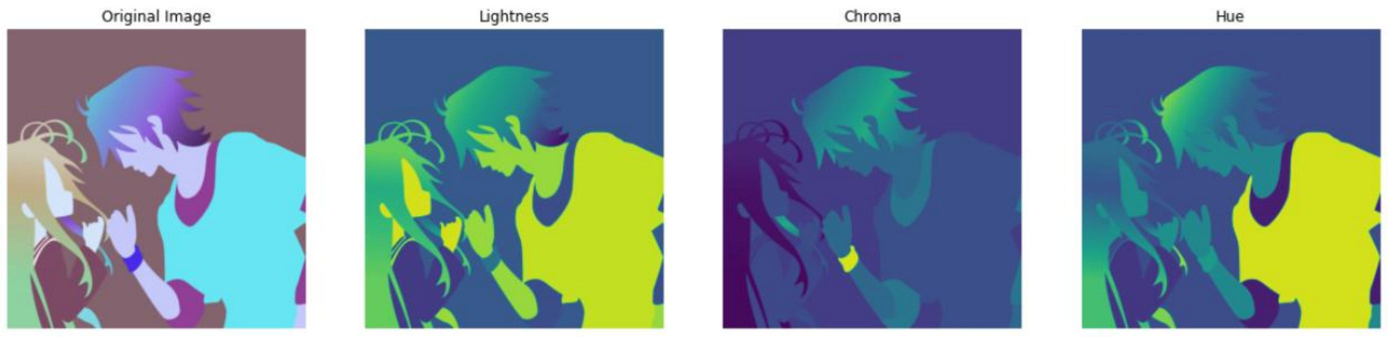


Fig. 3.4 SS after conversion to LCH

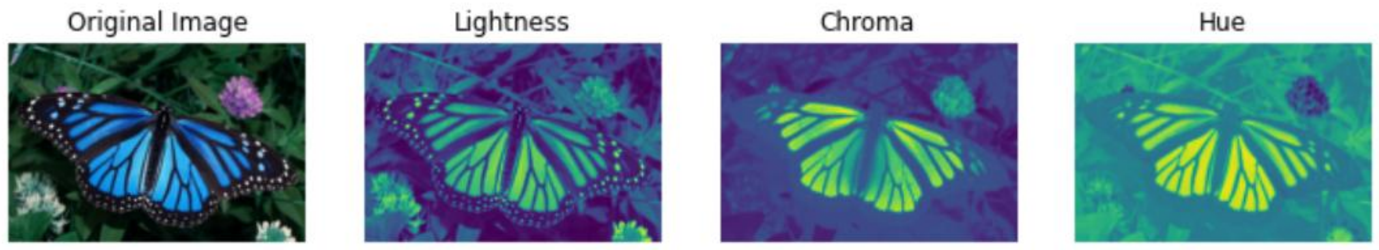


Fig. 3.5 Monarch after conversion to LCH

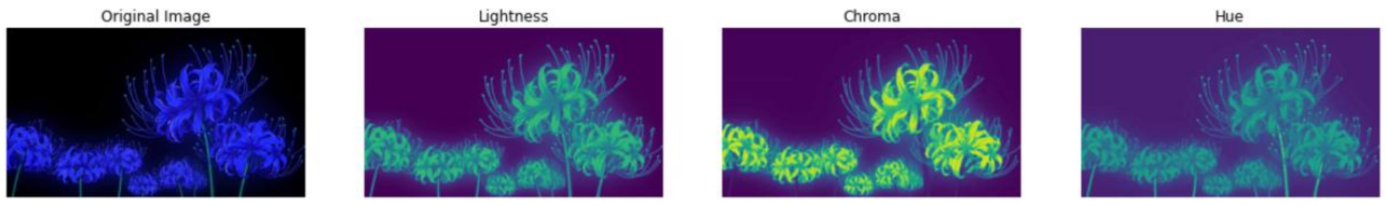


Fig. 3.6 Spider Lily after conversion to LCH

3.2 Denoising the image with Gaussian Filter

We start by denoising the image with a 1D Gaussian filter first in the vertical and then horizontal direction. For creating this Gaussian Filter we use the following formula:

We use a Gaussian Filter of size 5:

Gx = [[0.19205063 0.20392638 0.20804597 0.20392638 0.19205063]]

Gy = [[0.19205063]
[0.20392638]
[0.20804597]
[0.20392638]
[0.19205063]]

After denoising the image using the Gx and Gy filters we get the following results:

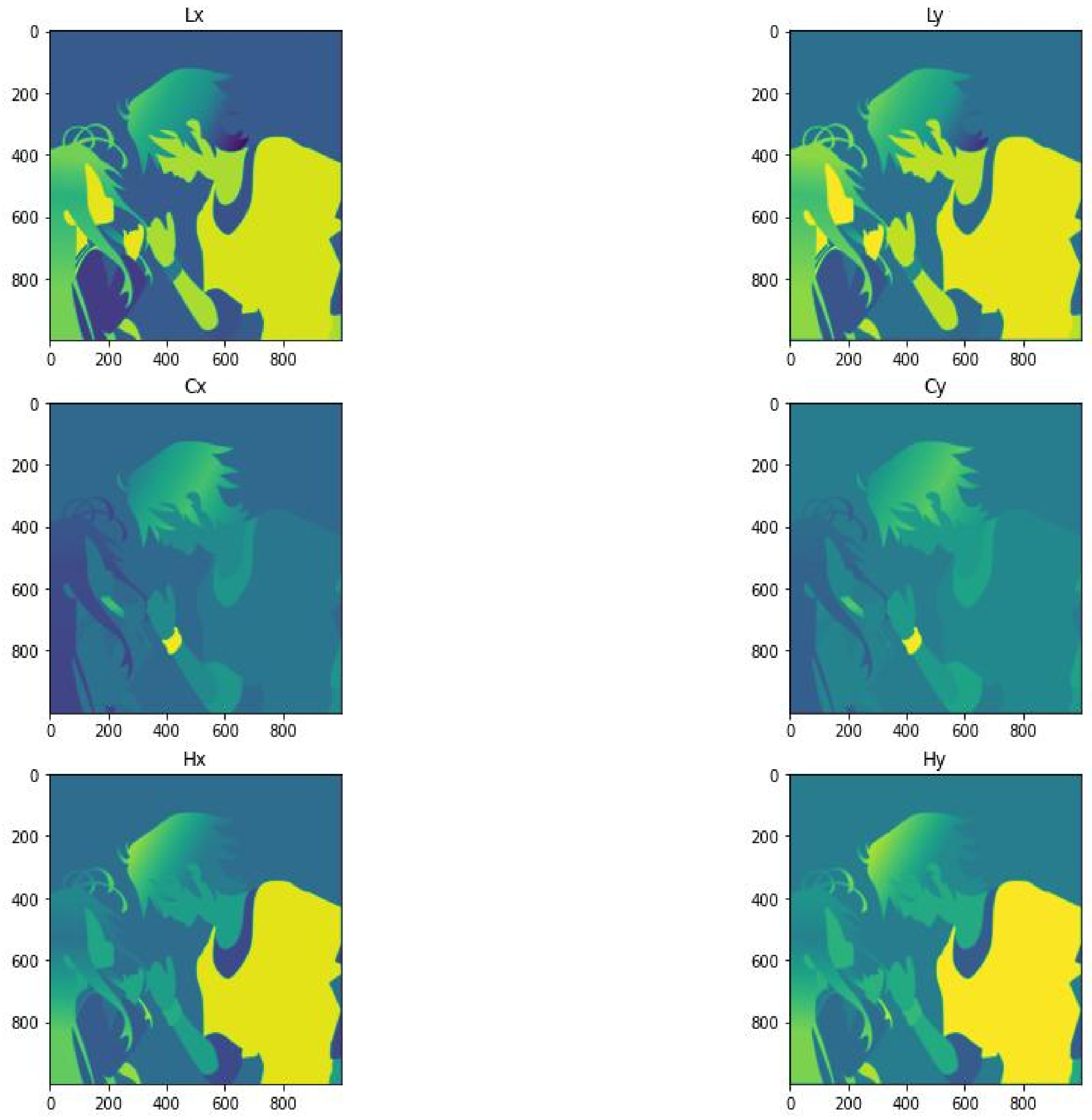


Fig. 3.7 SS after denoising using Gaussian Filter

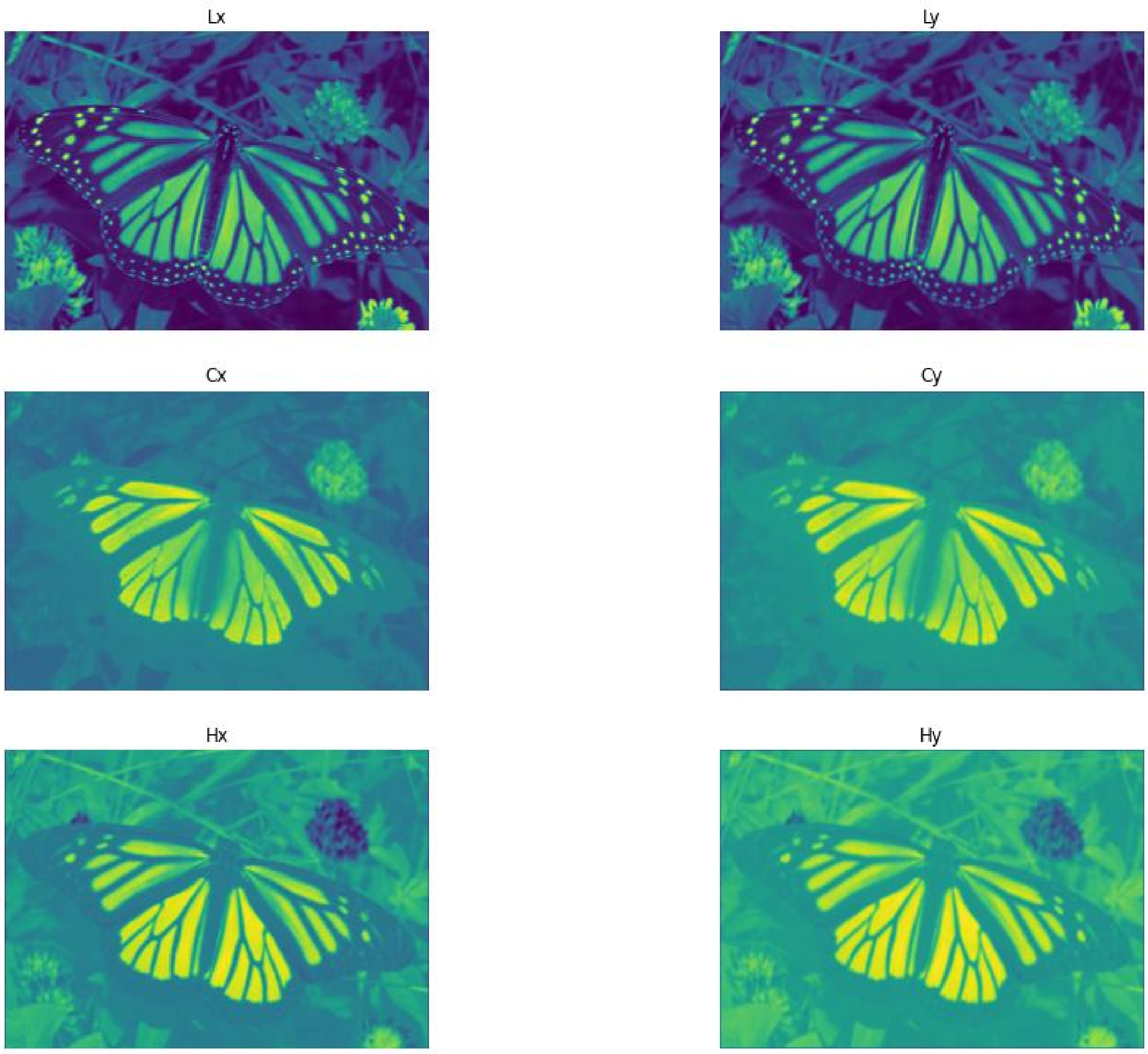


Fig. 3.8 Monarch after denoising using Gaussian Filter

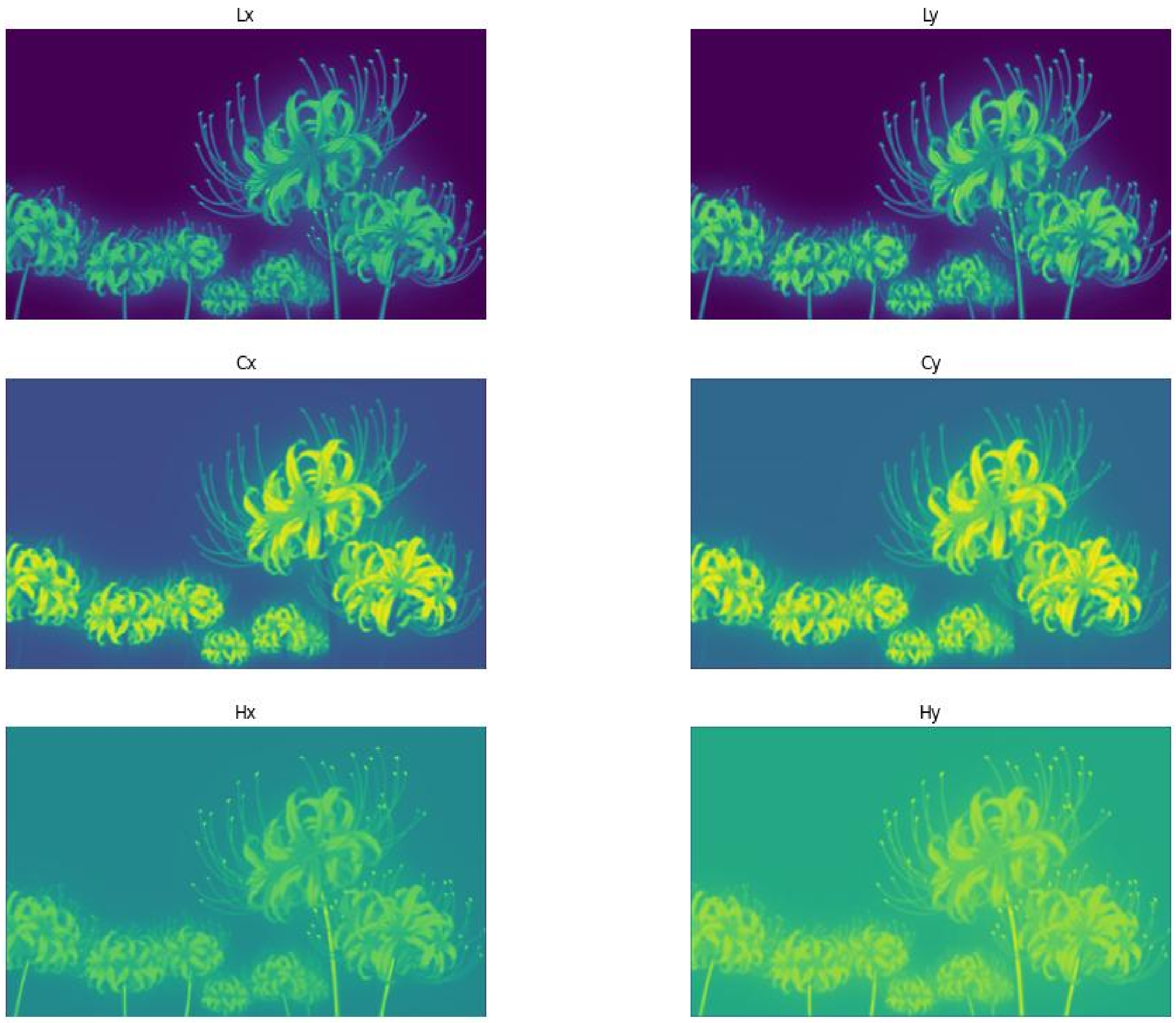


Fig. 3.9 Spider Lily after denoising using Gaussian Filter

3.3 Gradient Magnitude Image

Similar to A1, we use the same derivative filter and compute the Gradient Magnitude Image.

Dx = [-1 0 1] and Dy = [-1 0 1]^T

The gradient magnitude image is computed in the following way

$$\sqrt{(image\_dx)^2 + (image\_dy)^2}$$

Where image_dx and image_dy are the images after applying the Dx and Dy derivative filters.

The resulting image after computing the gradient magnitude is:

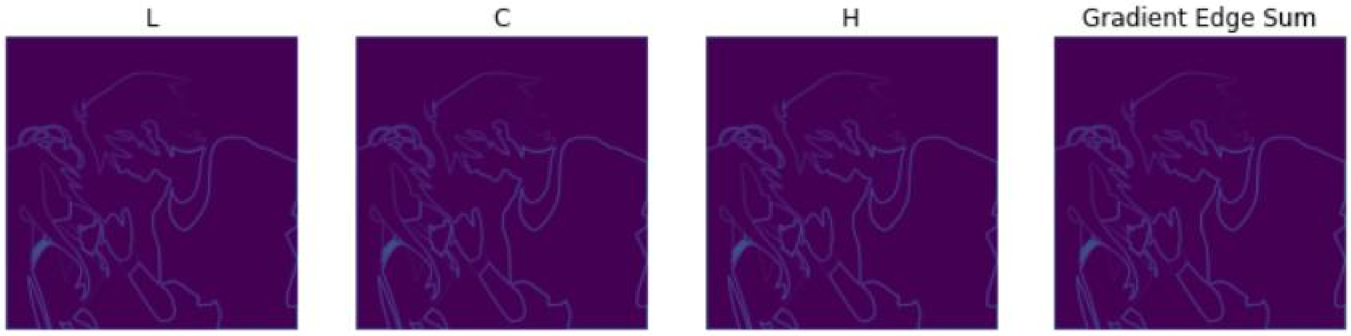


Fig. 3.10 SS - Gradient Magnitude Image

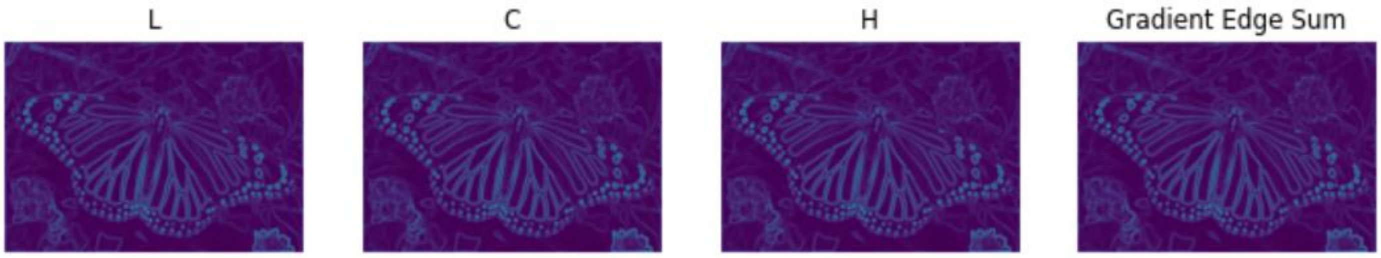


Fig. 3.11 Monarch - Gradient Magnitude Image

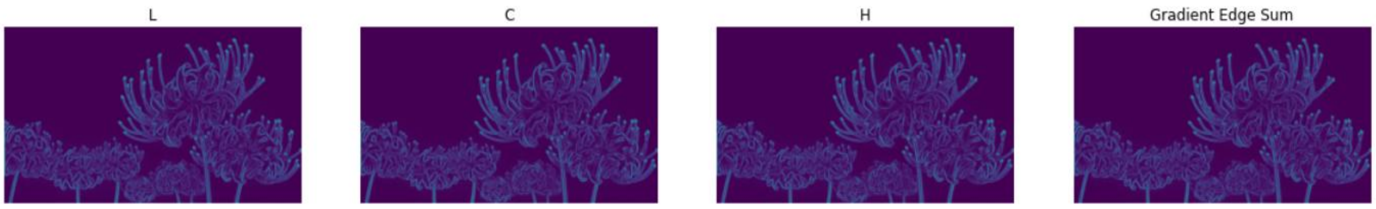


Fig. 3.12 Spider Lily - Gradient Magnitude Image

3.4 Binary Edge Images

The binary edge image can be computed by setting a threshold for the gradient magnitude image and depending on the threshold we set we we calculate the edges. We find the best case threshold for each image and use it to find the final image.

The final results are given below along with its threshold.

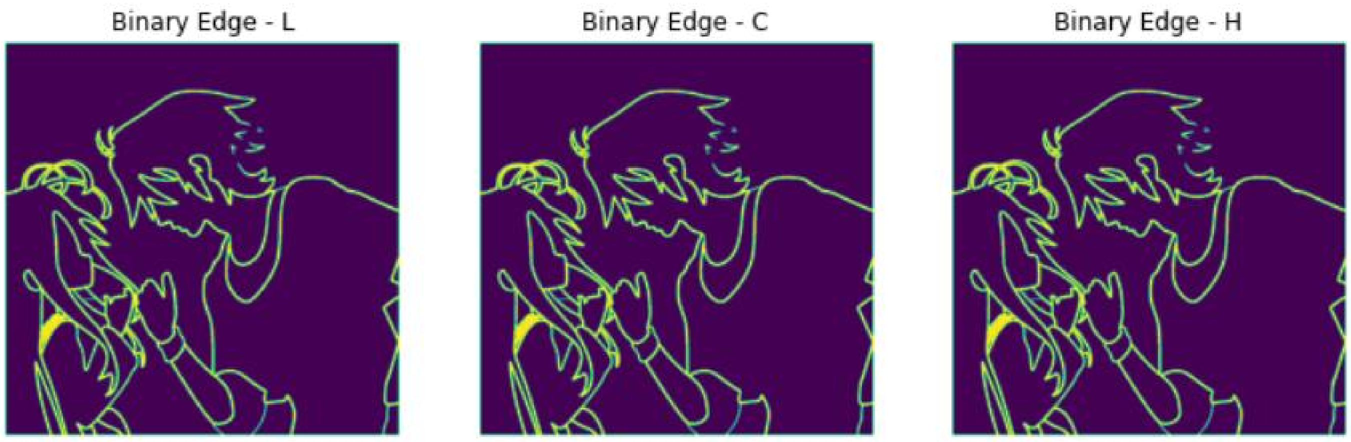


Fig. 3.13 SS - Binary Edges of LCH

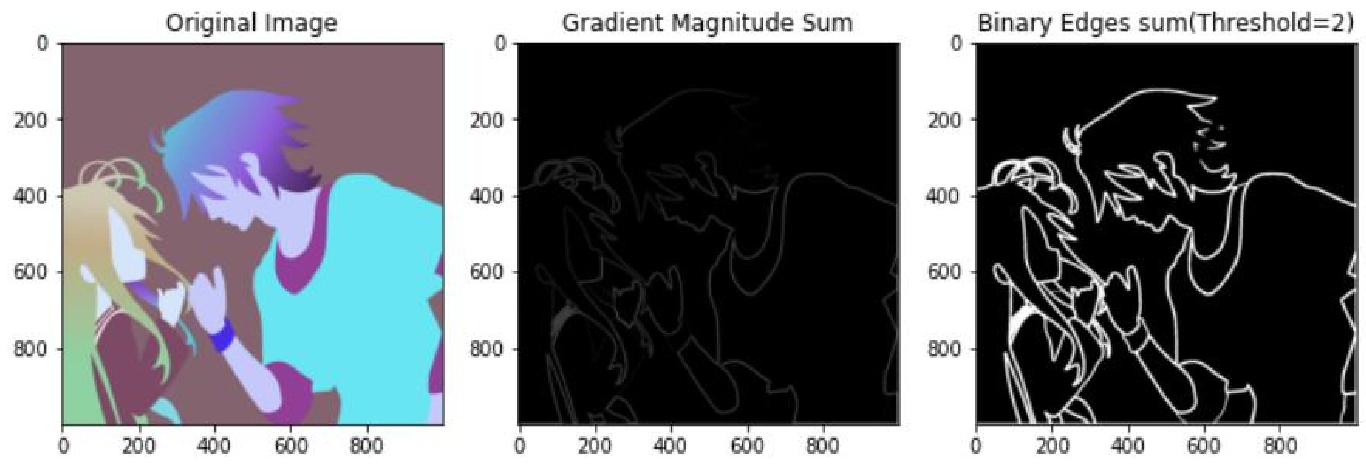


Fig. 3.14 SS - comparison of results

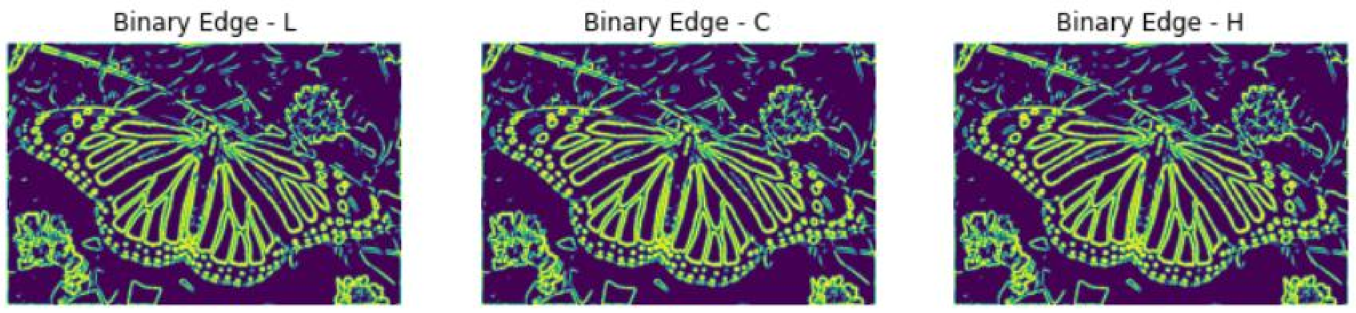


Fig. 3.15 Monarch - Binary Edges of LCH

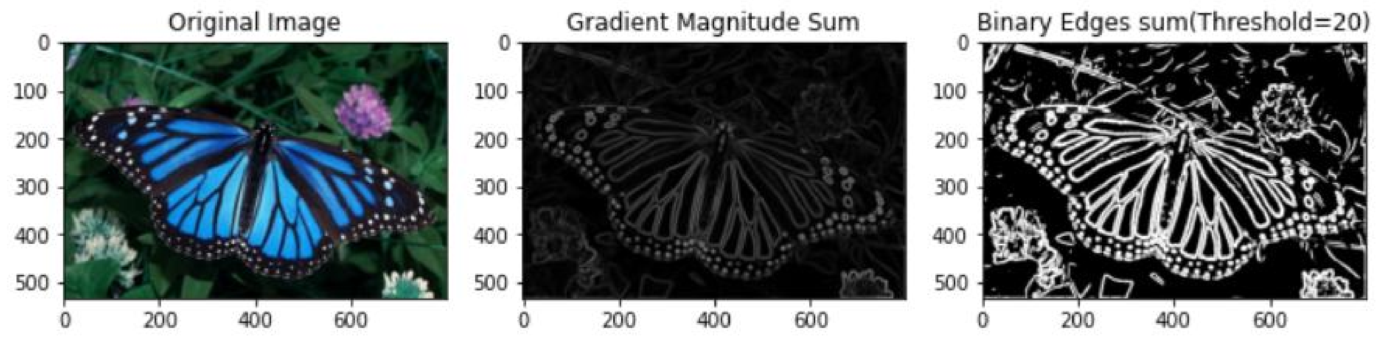


Fig. 3.16 Monarch - comparison of results

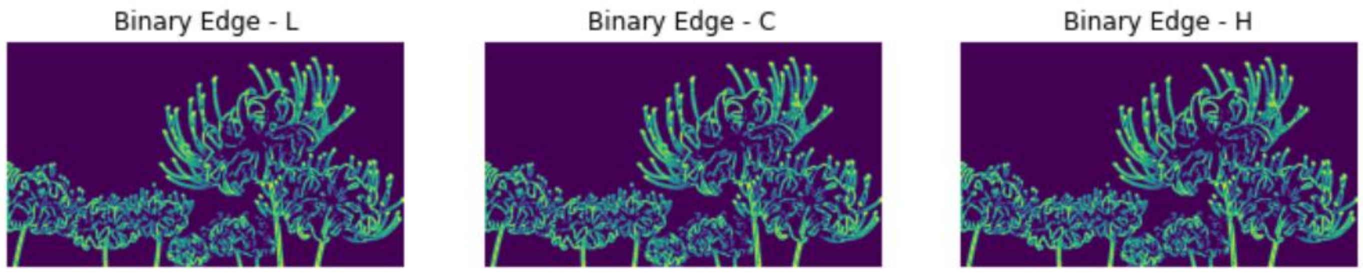


Fig. 3.17 Spider Lily - Binardy Edges of LCH

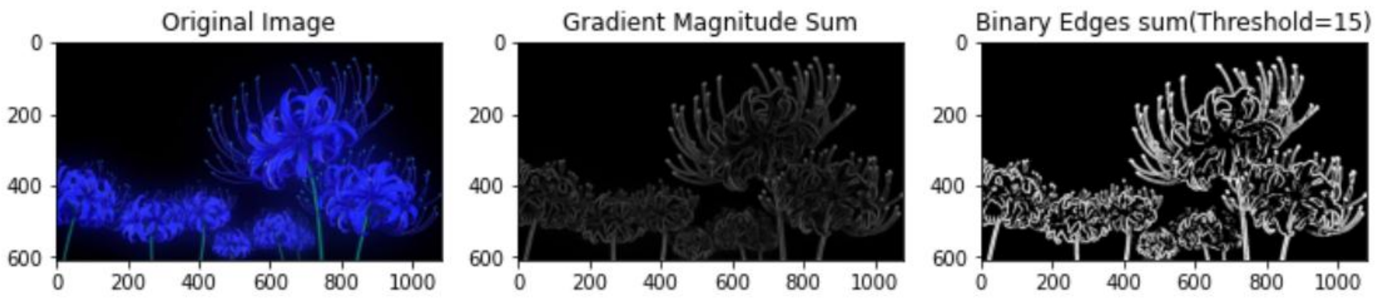


Fig. 3.18 Spider Lily - comparison of results

As we can see from the Spider Lily results, converting the image to black & white and doing the edge detection gives us similar results.

project-2

1 Computer Vision Project 2

1.1 1. Convolution and Derivative Filters - spider_lily-modified.jpg

[1]:
```
import numpy as np
import cv2  # For loading and displaying images
import matplotlib.pyplot as plt
```

[2]:
```
image = cv2.imread('spider_lily-modified.jpg', cv2.IMREAD_GRAYSCALE)
image = image.astype(float)  # Convert to float
```

[3]:
```
def convolution(f, I):
    f_height, f_width = f.shape
    I_height, I_width = I.shape
    pad_height, pad_width = f_height // 2, f_width // 2

    I_padded = np.pad(I, ((pad_height, pad_height), (pad_width, pad_width)),␣
 ↪mode='constant')

    im_conv = np.zeros(I.shape, dtype=np.float32)

    for i in range(I_height):
        for j in range(I_width):
            im_conv[i, j] = np.sum(f * I_padded[i:i + f_height, j:j + f_width])

    return im_conv
```

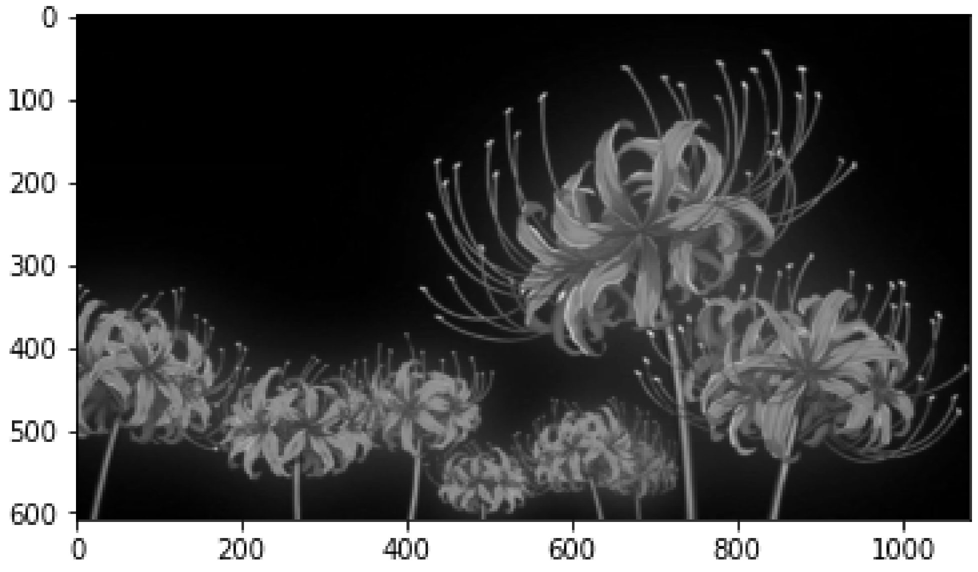
[4]:
```
sigma = 3
x_range = np.linspace(-int(sigma/2),int(sigma/2),sigma)
# print(x_range)
gaussian_filter = [ (1 / (sigma * np.sqrt(2*np.pi)) * np.exp(-x**2/
 ↪(2*sigma**2))) for x in x_range ]
total = sum(gaussian_filter)
gaussian_filter = [[x/total for x in gaussian_filter]]
Gx = np.array(gaussian_filter)
Gy = Gx.reshape(-1,1)
print("Gx =",Gx)
print("Gy =",Gy)
```

```
Gx = [[0.32710442 0.34579116 0.32710442]]
Gy = [[0.32710442]
 [0.34579116]
 [0.32710442]]
```

```
[5]: image_filtered_x = convolution(Gx, image)
     image_filtered_y = convolution(Gy, image_filtered_x)
```

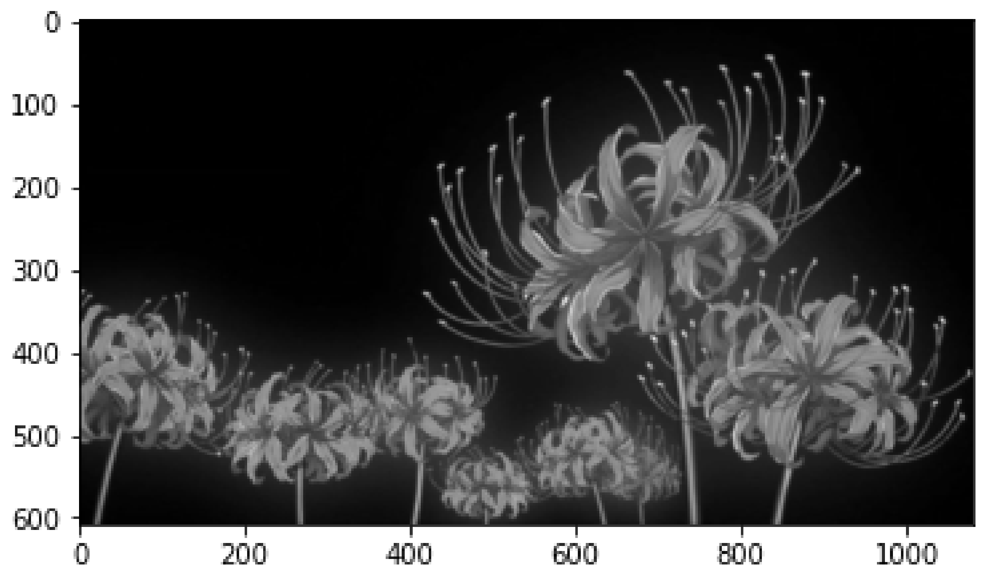
```
[6]: plt.imshow(image_filtered_x, cmap='gray')
```

```
[6]: <matplotlib.image.AxesImage at 0x211ade898b0>
```



```
[7]: plt.imshow(image_filtered_y, cmap='gray')
```

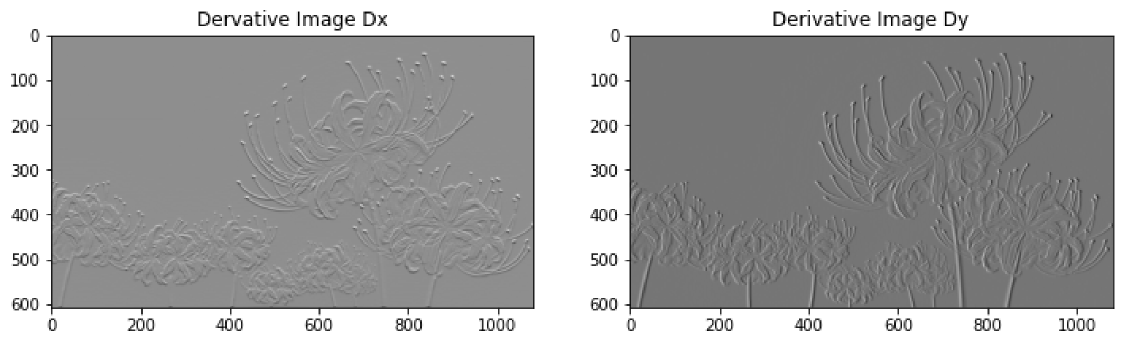
```
[7]: <matplotlib.image.AxesImage at 0x211ae1f3c40>
```



```
[8]: derivative_filter_x = np.array([[-1], [0], [1]])
     derivative_filter_y = derivative_filter_x.reshape((1, -1))
```

```
[9]: image_dx = convolution(derivative_filter_x, image_filtered_y)
     image_dy = convolution(derivative_filter_y, image_filtered_y)
```

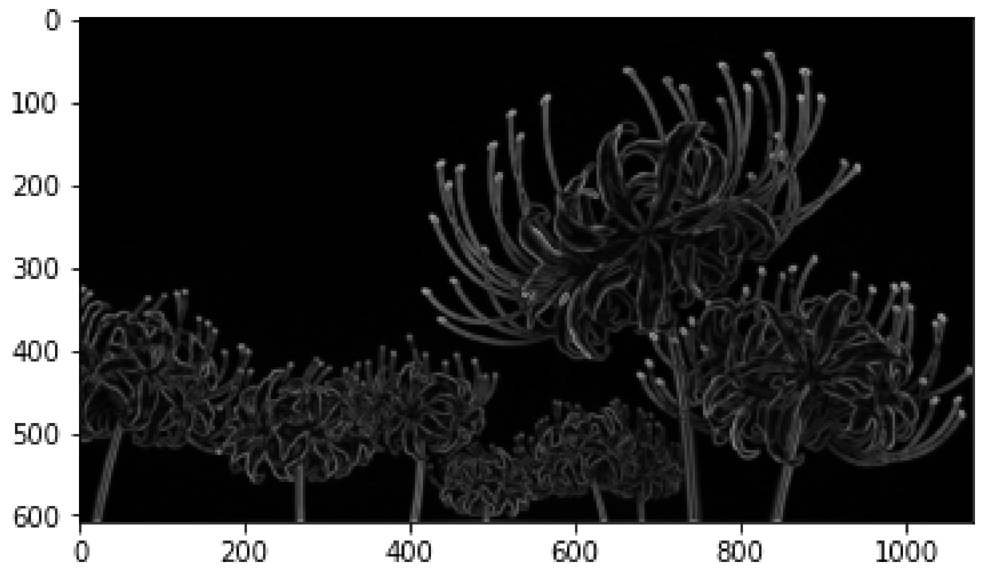
```
[10]: plt.figure(figsize=(12, 12))

      plt.subplot(1, 2, 1)
      plt.imshow(image_dx, cmap='gray')
      plt.title('Dervative Image Dx')

      plt.subplot(1, 2, 2)
      plt.imshow(image_dy, cmap='gray')
      plt.title('Derivative Image Dy')

      plt.show()
```



[11]:
```
gradient_magnitude = np.sqrt(image_dx ** 2 + image_dy ** 2)
```

[12]:
```
plt.imshow(gradient_magnitude, cmap='gray')
```

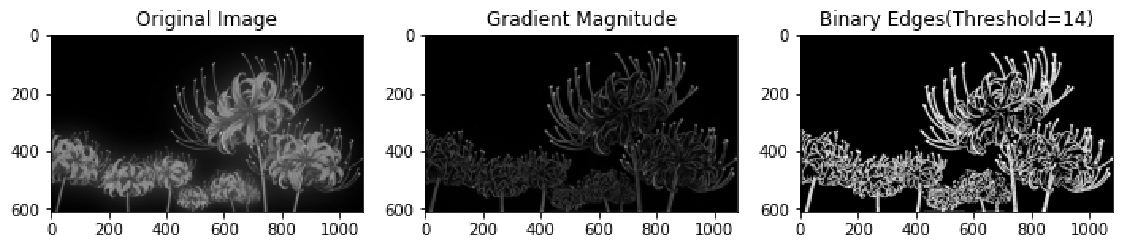
[12]:
```
<matplotlib.image.AxesImage at 0x211ae3122e0>
```



[13]:
```
threshold = 14
binary_edges = (gradient_magnitude > threshold).astype(np.uint8) * 255
```

[14]:
```
cv2.imshow('Original Image', image.astype(np.uint8))
cv2.imshow('Gradient Magnitude', gradient_magnitude.astype(np.uint8))
cv2.imshow('Binary Edges', binary_edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[15]:
```
plt.figure(figsize=(12, 12))
plt.subplot(1, 3, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')

plt.subplot(1, 3, 2)
plt.imshow(gradient_magnitude, cmap='gray')
plt.title('Gradient Magnitude')

plt.subplot(1, 3, 3)
plt.imshow(binary_edges, cmap='gray')
plt.title('Binary Edges(Threshold=14)')

plt.show()
```



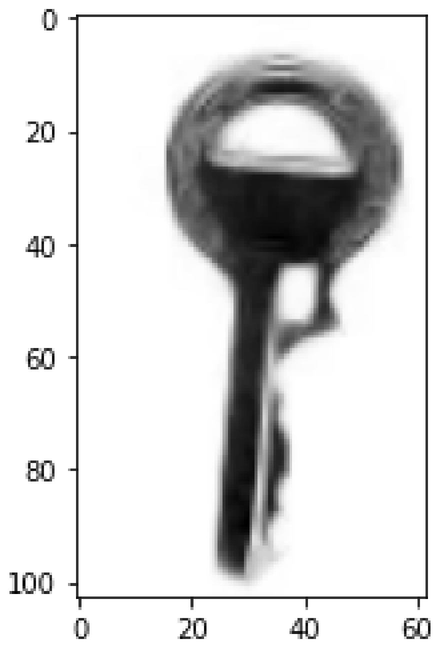
1.2 Cross Correlation and Template Matching

[16]:
```
original_image = cv2.imread('multiplekeys.png', cv2.IMREAD_GRAYSCALE)
template = cv2.imread('singlekey_bw.png', cv2.IMREAD_GRAYSCALE)
```

[17]:
```
min_value = np.min(template)
max_value = np.max(template)
mean_value = np.mean(template)
```

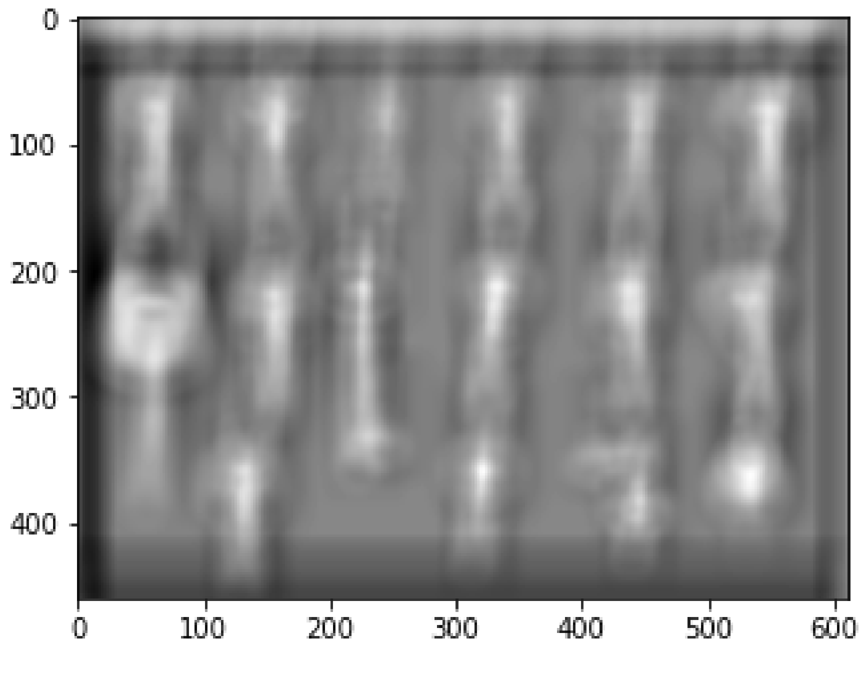
[18]:
```
normalized_template = template - mean_value
plt.imshow(normalized_template, cmap='gray')
```

[18]:
```
<matplotlib.image.AxesImage at 0x211b1ee3f10>
```



```
[19]: correlation_image = convolution(normalized_template, original_image)
      plt.imshow(correlation_image, cmap='gray')
```

```
[19]: <matplotlib.image.AxesImage at 0x211b1e94280>
```



[20]:
```
peak,peak_x,peak_y = 0,0,0

for i in range(correlation_image.shape[0]):
    for j in range(correlation_image.shape[1]):
        if(peak<correlation_image[i][j]):
            peak = correlation_image[i][j]
            peak_y,peak_x = i,j

print("Peak matching:",peak,"\nPeak's x coordinate:",peak_x,"\nPeak's y␣
 ↪coordinate:",peak_y)
# print(correlation_image.shape)
```

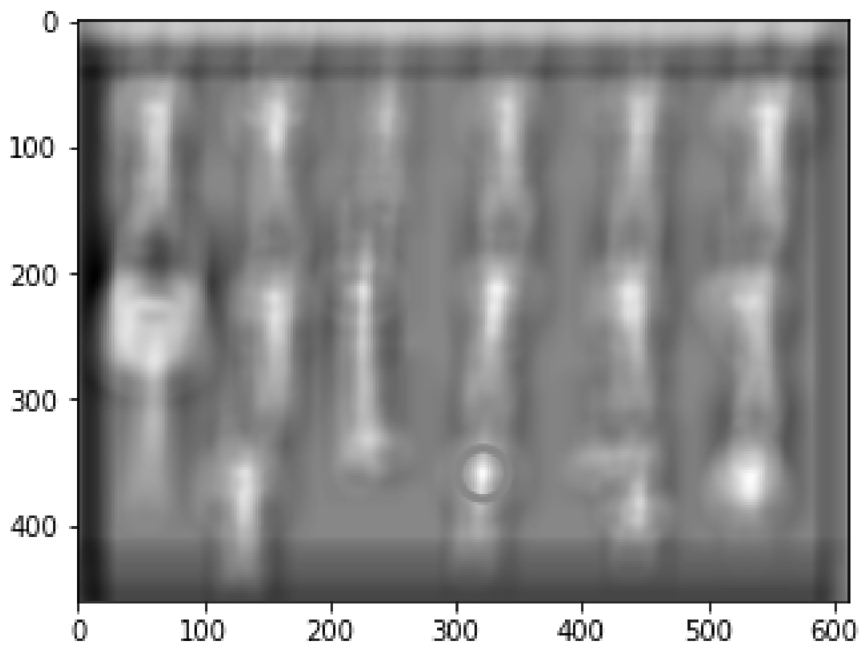
```
Peak matching: 21884380.0
Peak's x coordinate: 321
Peak's y coordinate: 359
```

[21]:
```
cv2.circle(correlation_image, (peak_x, peak_y), 20, (0, 0, 0), 5)
```

[21]:
```
array([[  6304977.5 ,   4782022.  ,   3169721.5 , …,   3263821.8 ,
          3854820.5 ,   4626314.  ],
       [  6282293.5 ,   4727595.  ,   3082921.  , …,   3326519.5 ,
          3955679.  ,   4761452.5 ],
       [  6265925.  ,   4679234.  ,   3002292.5 , …,   3381737.2 ,
          4049052.8 ,   4889200.5 ],
       …,
       [-12563124.  , -13500901.  , -14436205.  , …,  -1952994.6 ,
         -1004182.7 ,    -54070.79],
       [-12475578.  , -13382922.  , -14286474.  , …,  -2028417.5 ,
         -1110627.2 ,   -193245.88],
       [-12413943.  , -13290541.  , -14162417.  , …,  -2102245.5 ,
         -1217208.8 ,   -333292.97]], dtype=float32)
```

[22]:
```
plt.imshow(correlation_image, cmap='gray')
# cv2.imshow("correlation image",correlation_image)
# cv2.waitKey(0)
# cv2.destroyAllWindows()
```

[22]:
```
<matplotlib.image.AxesImage at 0x211b1e01a00>
```



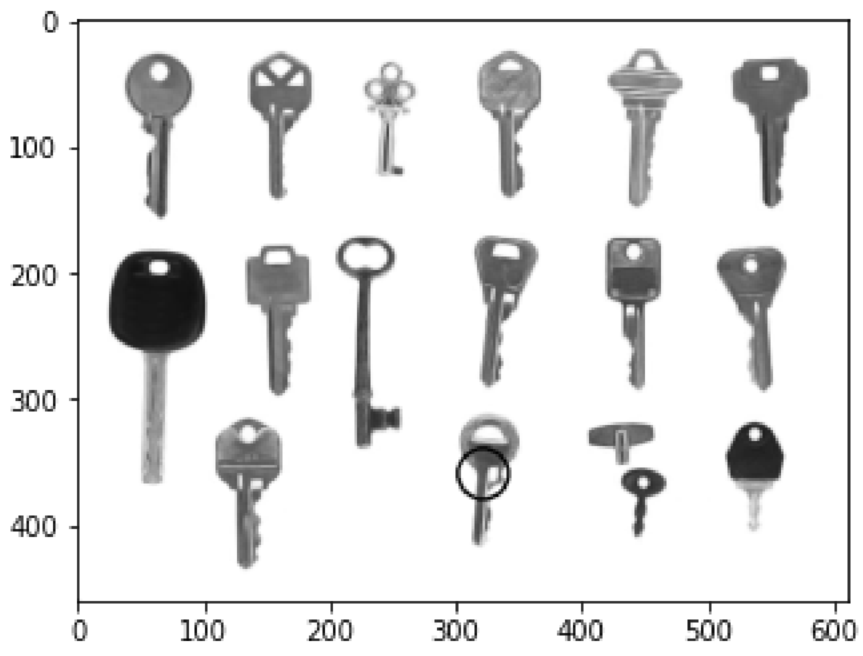
[23]:
```
original_image_with_peak = original_image.copy()
cv2.circle(original_image_with_peak, (peak_x, peak_y), 20, (0, 255, 0), 2)
```

[23]:
```
array([[107, 172, 208, …, 254, 169, 107],
       [169, 226, 244, …, 255, 251, 175],
       [217, 248, 253, …, 255, 254, 235],
       …,
       [254, 255, 255, …, 255, 255, 255],
       [169, 251, 254, …, 255, 255, 249],
       [107, 175, 235, …, 255, 249, 107]], dtype=uint8)
```

[24]:
```
cv2.imshow("Original Image", original_image)
cv2.imshow("Template", template)
cv2.imshow("Peak Image", original_image_with_peak)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

[25]:
```
plt.imshow(original_image_with_peak, cmap='gray')
```

[25]:
```
<matplotlib.image.AxesImage at 0x211ae2fb280>
```



1.3 Creative Part - Color image edge detection

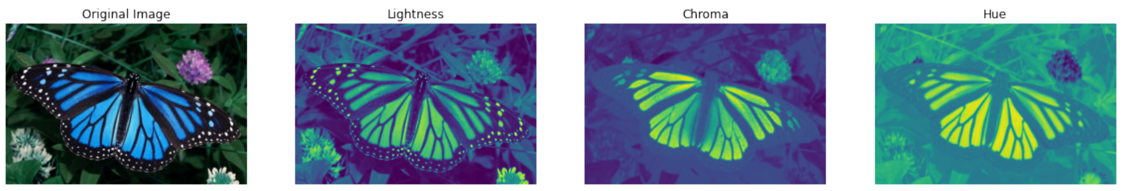
[26]:
```
color_image = cv2.imread('monarch.jpg')
plt.imshow(color_image)
```

[26]:
```
<matplotlib.image.AxesImage at 0x211b1e38490>
```

[27]:
```
lch_image = cv2.cvtColor(color_image, cv2.COLOR_BGR2Luv)
```

[28]:
```
l_channel, c_channel, h_channel = cv2.split(lch_image)
```

[29]:
```
plt.figure(figsize= (20, 12))

plt.axis("off")
plt.subplot(1, 4, 1)
plt.imshow(color_image)
plt.title('Original Image')

plt.axis("off")
plt.subplot(1, 4, 2)
plt.imshow(l_channel)
plt.title('Lightness')

plt.axis("off")
plt.subplot(1, 4, 3)
plt.imshow(c_channel)
plt.title('Chroma')

plt.axis("off")
plt.subplot(1, 4, 4)
plt.imshow(h_channel)
plt.title('Hue')

plt.axis("off")
plt.show()
```



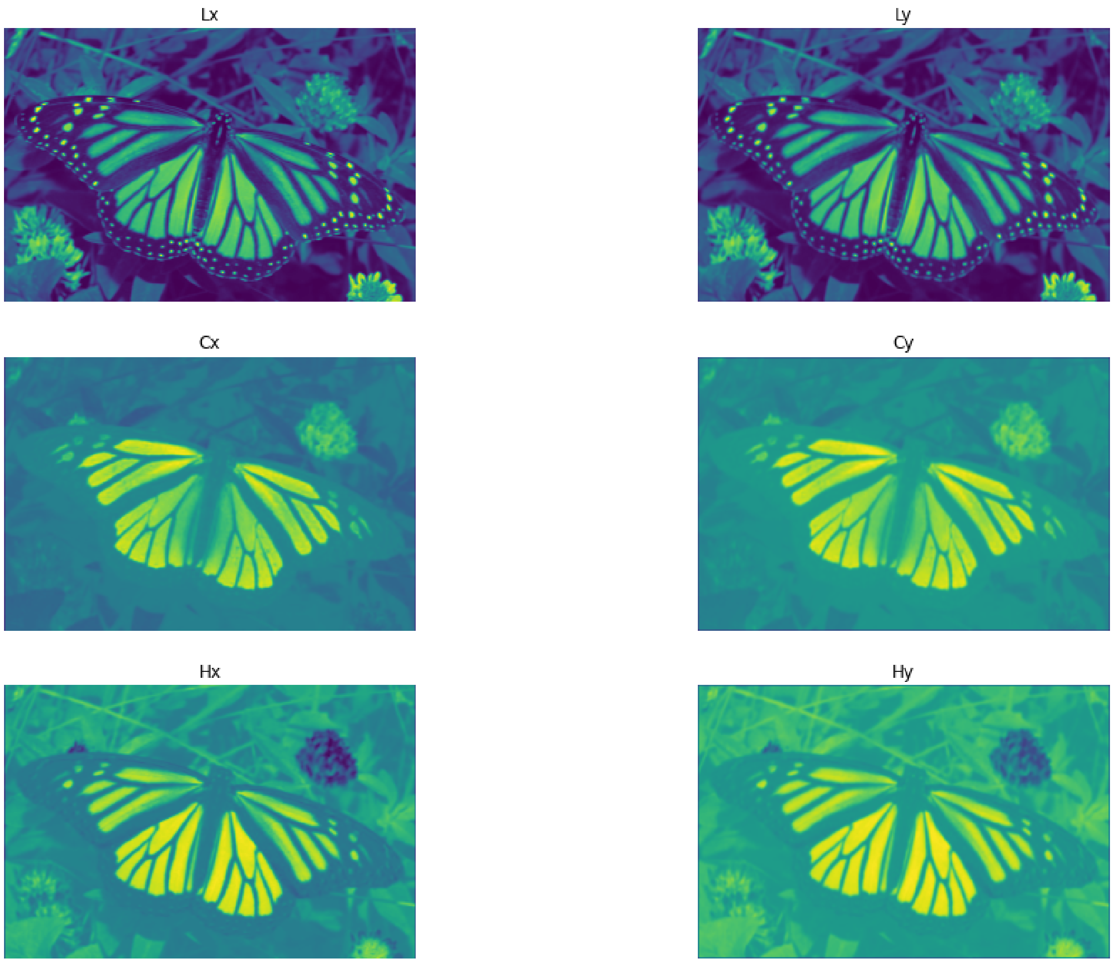
[30]:
```
sigma = 5
x_range = np.linspace(-int(sigma/2),int(sigma/2),sigma)
# print(x_range)

gaussian_filter = [ (1 / (sigma * np.sqrt(2*np.pi)) * np.exp(-x**2/
 ↪(2*sigma**2))) for x in x_range ]
total = sum(gaussian_filter)
```

```
gaussian_filter = [[x/total for x in gaussian_filter]]

Gx = np.array(gaussian_filter)
Gy = Gx.reshape(-1,1)
print("Gx =",Gx)
print("Gy =",Gy)
```

```
Gx = [[0.19205063 0.20392638 0.20804597 0.20392638 0.19205063]]
Gy = [[0.19205063]
 [0.20392638]
 [0.20804597]
 [0.20392638]
 [0.19205063]]
```

[31]:
```
l_smoothed_x = convolution(Gx, l_channel)
l_smoothed_y = convolution(Gy, l_smoothed_x)

c_smoothed_x = convolution(Gx, c_channel)
c_smoothed_y = convolution(Gy, c_smoothed_x)

h_smoothed_x = convolution(Gx, h_channel)
h_smoothed_y = convolution(Gy, h_smoothed_x)
```

[32]:
```
plt.figure(figsize=(16, 12))

plt.axis("off")
plt.subplot(3, 2, 1)
plt.imshow(l_smoothed_x)
plt.title('Lx')

plt.axis("off")
plt.subplot(3, 2, 2)
plt.imshow(l_smoothed_y)

plt.title('Ly')

plt.axis("off")
plt.subplot(3, 2, 3)
plt.imshow(c_smoothed_x)
plt.title('Cx')

plt.axis("off")
plt.subplot(3, 2, 4)
plt.imshow(c_smoothed_y)
plt.title('Cy')

plt.axis("off")
plt.subplot(3, 2, 5)
plt.imshow(h_smoothed_x)
plt.title('Hx')

plt.axis("off")
plt.subplot(3, 2, 6)
plt.imshow(h_smoothed_y)
plt.title('Hy')

plt.axis("off")
plt.show()
```



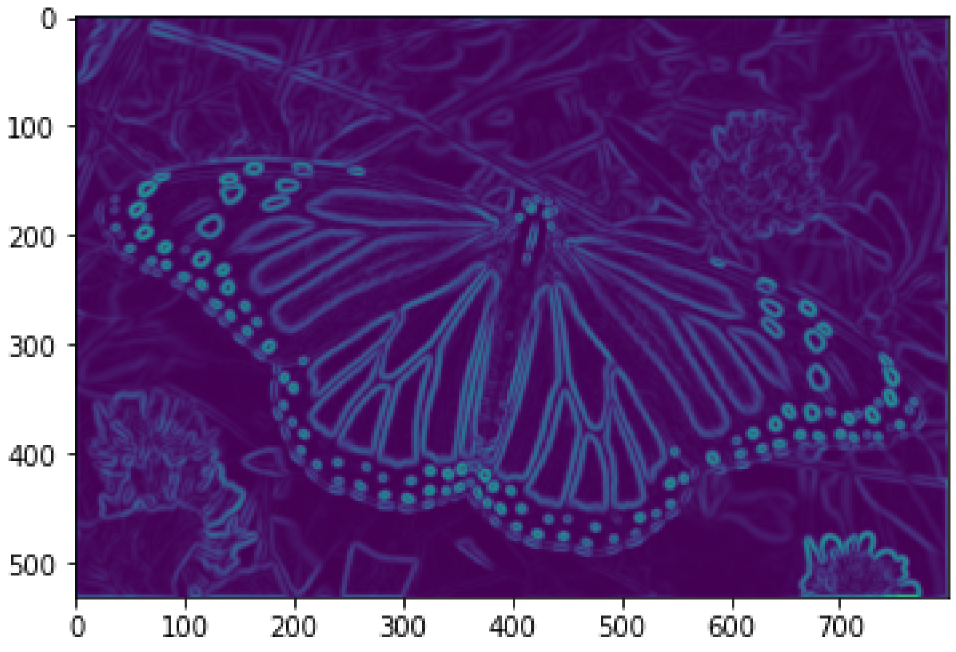
[33]:
```
derivative_filter_x = np.array([[-1], [0], [1]])
derivative_filter_y = derivative_filter_x.reshape((1, -1))
```

[34]:
```
l_dx = convolution(derivative_filter_x, l_smoothed_y)
l_dy = convolution(derivative_filter_y, l_smoothed_y)
```

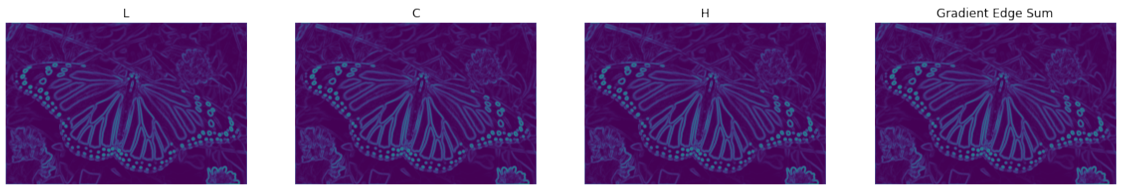
```
grad_mag_l = np.sqrt(l_dx ** 2 + l_dy ** 2)

c_dx = convolution(derivative_filter_x, c_smoothed_y)
c_dy = convolution(derivative_filter_y, c_smoothed_y)
grad_mag_c = np.sqrt(l_dx ** 2 + l_dy ** 2)

h_dx = convolution(derivative_filter_x, h_smoothed_y)
h_dy = convolution(derivative_filter_y, h_smoothed_y)
grad_mag_h = np.sqrt(l_dx ** 2 + l_dy ** 2)
```

[35]:
```
grad_edge_sum = grad_mag_l + grad_mag_c + grad_mag_h
plt.imshow(grad_edge_sum)
```

[35]:
```
<matplotlib.image.AxesImage at 0x211b56ada00>
```

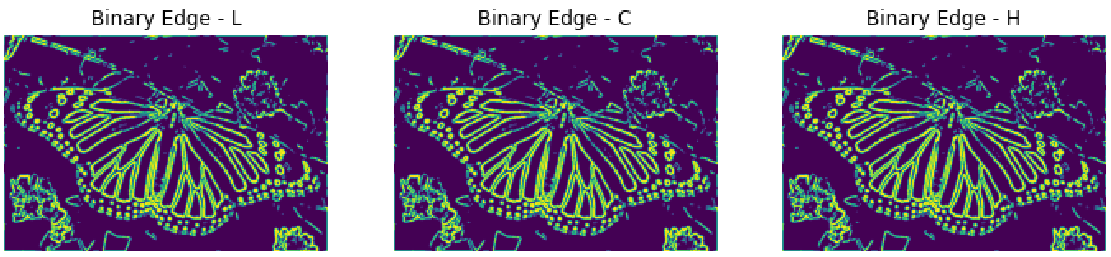


[36]:
```
plt.figure(figsize=(20, 12))

plt.axis("off")
plt.subplot(1, 4, 1)
plt.imshow(grad_mag_l)
plt.title('L')

plt.axis("off")
plt.subplot(1, 4, 2)
plt.imshow(grad_mag_c)
plt.title('C')

plt.axis("off")
plt.subplot(1, 4, 3)
plt.imshow(grad_mag_c)
plt.title('H')

plt.axis("off")
plt.subplot(1, 4, 4)
plt.imshow(grad_edge_sum)
plt.title('Gradient Edge Sum')

plt.axis("off")
plt.show()
```

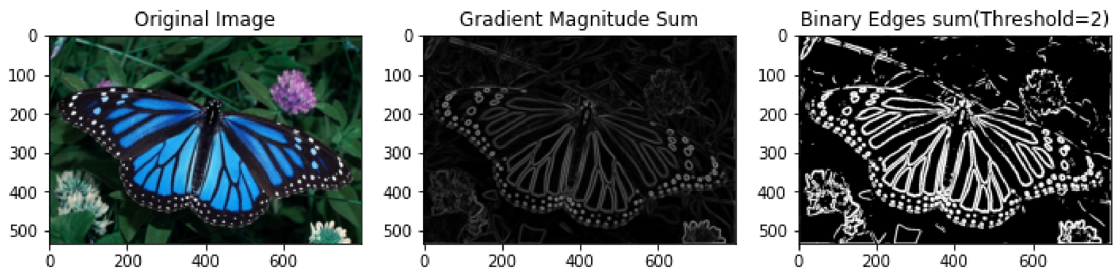


[37]:
```
threshold = 20
binary_edges_l = (grad_mag_l > threshold) * 255
binary_edges_c = (grad_mag_c > threshold) * 255
binary_edges_h = (grad_mag_h > threshold) * 255
```

[38]:
```
plt.figure(figsize=(12, 12))

plt.axis("off")
plt.subplot(1, 3, 1)
plt.imshow(binary_edges_l)
plt.title('Binary Edge - L')

plt.axis("off")
plt.subplot(1, 3, 2)
plt.imshow(binary_edges_c)
plt.title('Binary Edge - C')

plt.axis("off")
plt.subplot(1, 3, 3)
plt.imshow(binary_edges_h)
plt.title('Binary Edge - H')

plt.axis("off")
plt.show()
```



```
[39]: binary_edge_sum = binary_edges_l + binary_edges_c + binary_edges_h

      plt.figure(figsize=(12, 12))
      plt.subplot(1, 3, 1)
      plt.imshow(color_image, cmap='gray')
      plt.title('Original Image')

      plt.subplot(1, 3, 2)
      plt.imshow(grad_edge_sum, cmap='gray')
      plt.title('Gradient Magnitude Sum')

      plt.subplot(1, 3, 3)
      plt.imshow(binary_edge_sum, cmap='gray')
      plt.title('Binary Edges sum(Threshold=2)')

      plt.show()
```



```
[ ]:
```