Computer Vision Coursework1 Report

Haojing Tong

February 2024

1 Results

This coursework let us compute the effect of convolving an image with the average and the weighted average smoothing kernels.

1.1 Step1

Firstly, I generate the original image and the convolved image. I showed the comparison below.

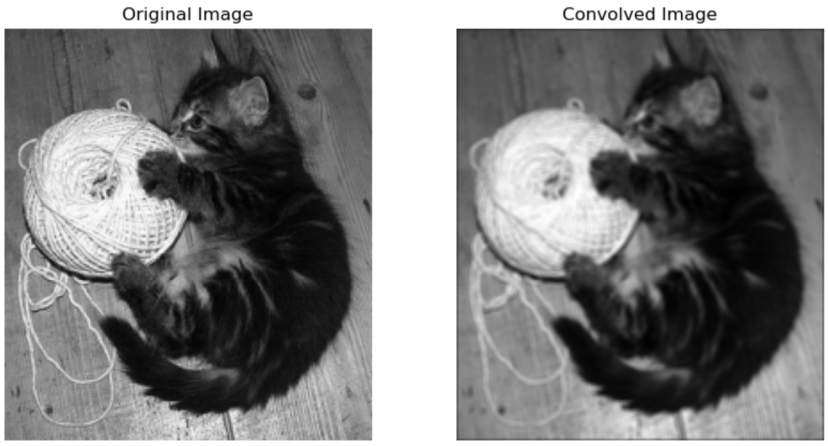

Figure 1: Here is the Comparisons between Original Image and Convolved Image

For the step1 task, I use the method of convolution function. Firstly, I padded the image which should larger than input image with 2 rows and columns, initialize them as zeros. During convolution, border practice is common, help the kernel to be applied properly at the edges of the image. After that, it is the main convolution operation: According to the instructions, the code iterates every pixels of the padded image, including the outermost layer of the padding(x-1:x+2,y-1:y+2). It calculates the convolution at each position by

multiplying the extracted region with the kernel and sum the results. And then store them to 'convolvedImage', which represents the filtered image. At last, do the normalization and date type conversion work. The generated convolved image looks more smooth and reduce the noise point, it would help the later steps.

1.2 Step2

Secondly, I generated the Horizontal and Vertical Gradient Image. Based on them, I calculate the gradient magnitude image. I showed the comparisons below.

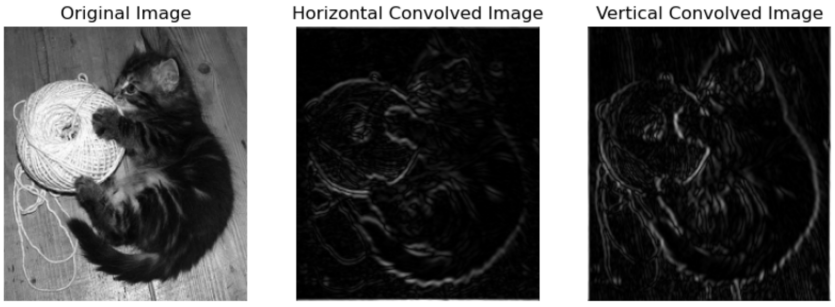

Figure 2: Here is the Comparisons between Original Image, horizontal gradient image and vertical gradient image.

For the step2 task, I firstly define two kernels, one is for the horizontal gradient image and the other is for the vertical gradient image. After that, I applied the convolution function which I defined before to detect edge. After that, I show the two images.

Once we have the image 'Gx' and 'Gy', I considered how can I combined them together to compute Gradient Magnitude. It should be computed for each pixel by taking the square root of the sum of the squares of 'Gx' and 'Gy' images.

1.3 Step3

After I get the Gradient Magnitude Image, I was planning acquire the threshold value of the image, such that I can generate a most clear image which could represent the edge of the cat. Firstly, I plotted the image histogram for the edge-strength image. Below is the figure of the histogram.

When I checked the histogram, the best threshold value should be around 10 to 20, because the drop speed between 10 to 20 become slow obviously. So I try the each threshold value and comparing the generated image. I found when the value is 15, the image have the best effectiveness.

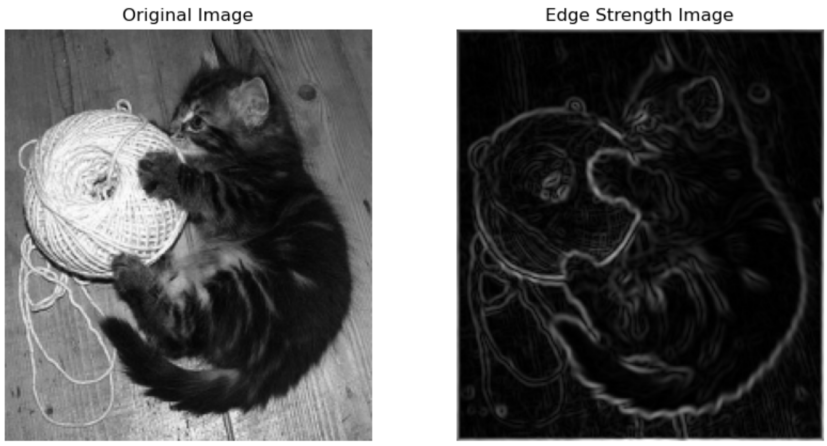

Figure 3: Here is the Comparisons between Original Image and Gradient Magnitude Image.

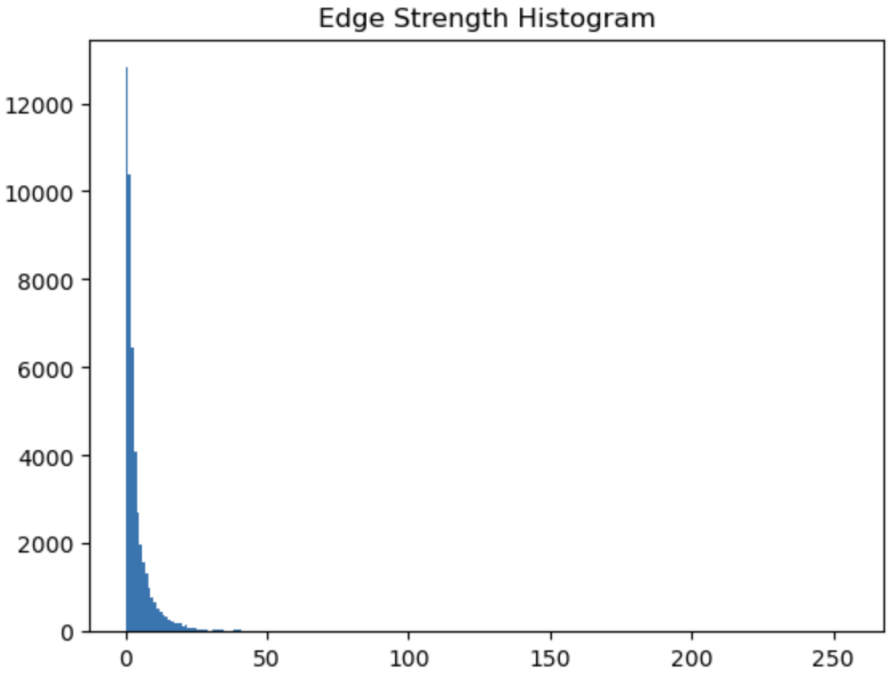

Figure 4: Here is the edge strength histogram.

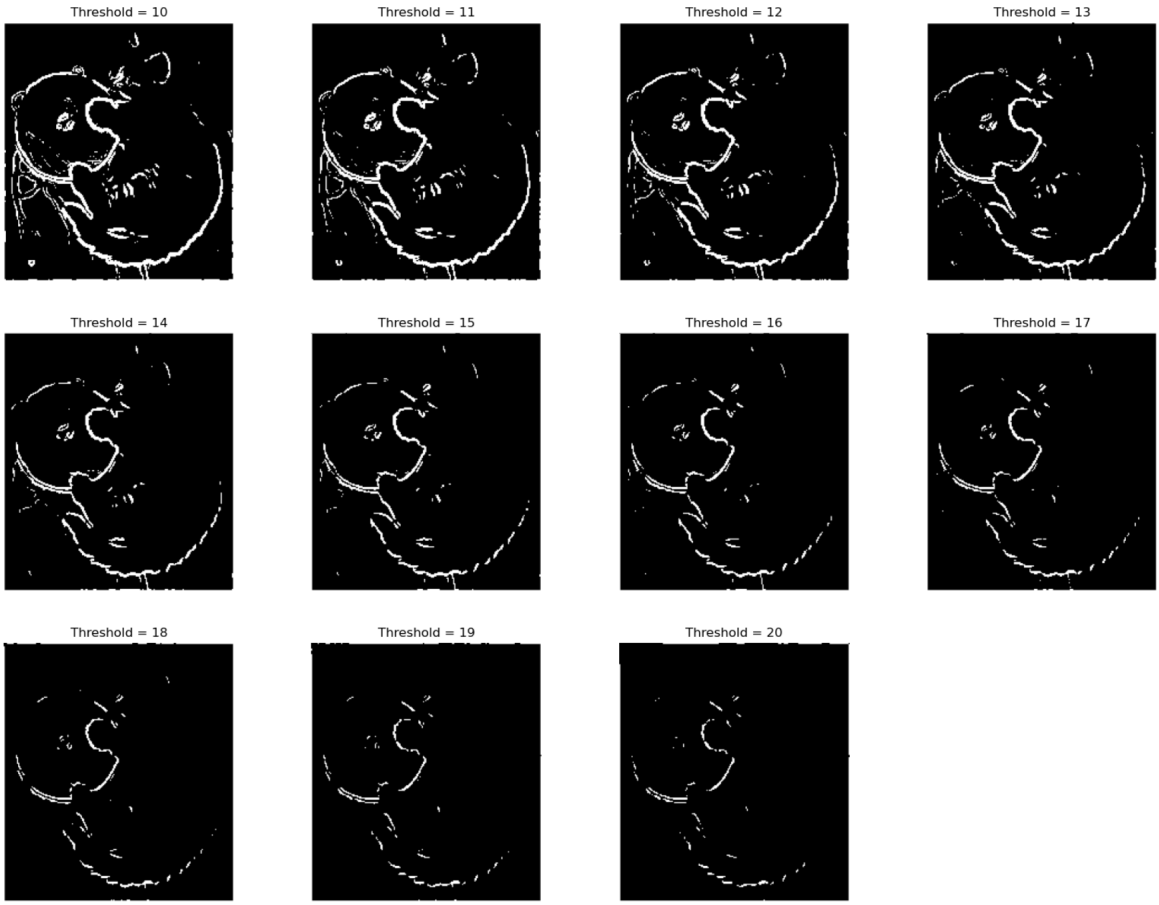

Figure 5: Here is the comparison of different threshold value image.

Figure 6: Here is the Threshold Edge Image, when the threshold value equals 15.

1.4 Step4

For the step4, I need to repeat the image using the different kernel to find a weight mean from the original images.

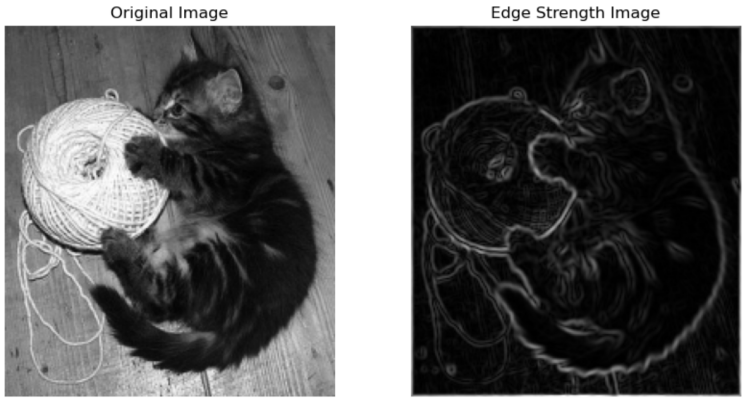

Figure 7: Here is the weighted mean edge strength image.

I find the difference of the edge strength weighted or not are basically same. It should be more smoothing but it is not really obvious.

2 Conclusions

For the step1: The convolved image looks more smooth.

Foe the step2: The horizontal gradient kernel highlights the vertical edges and the vertical gradient kernel highlights the horizontal edges. The combining gradients highlights the edges in all directions, which makes the edge more obviously.

For the step3: The threshold method make the image just show the black and white two colors, which perfectly represents the edges of the image. But it is basically impossible to show the complete edges. Through the comparison of different values of threshold, I can get the thresholded image with best effectiveness.

For the step4: I think the weighted mean has more blurred effectiveness theoretically, but the experiment results does not show it very clear.

3 Questions

For the Question in Step3: Can you find a threshold value that gives the edges of the cat, but not the patterns in the fur, or the wood-grain?

I can find the threshold value of the cat in the given picture. When the threshold value is 15, the image show the clearest edges.

For the Question in step4: What difference has the weighted-mean smoothing made to the edges detected?

For my own experiment results, I think the weighted mean has more blurred effectiveness, but the effect is not very obviously.

4 Appendix

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
```

Load the Image

In [2]:
```
# Read and display the uploaded image
img = Image.open('kitty.bmp').convert('L')
img = np.asarray(img, dtype=float)
plt.imshow(img, cmap="gray")
plt.axis('off')
plt.show()

img.dtype
```

Figure 8: Code Snippet 1

Step1: Convolution Function

In [3]:
```
def convolve2d(image, kernel):
    image_padded = np.zeros((image.shape[0] + 2, image.shape[1] + 2), dtype=float)
    image_padded[1:-1, 1:-1] = image  # Add zero padding to the input image

    rows, cols = image_padded.shape
    convolvedImage = np.zeros((rows-2, cols-2))

    for x in range(1, rows - 1):     # Loop over every pixel of the image
        for y in range(1, cols - 1):
            region = image_padded[x-1:x+2, y-1:y+2]
            convolvedImage[x-1,y-1] = np.sum(region * kernel)
    convolvedImage = convolvedImage.astype(float)
    return abs(convolvedImage)/4
```

In [4]:
```
# Smoothed Kerner
kernel = np.ones((3, 3), np.float32) / 9

# Weighted Kernel
weighted_kernel = np.array([[0.8, 1.2, 0.8],
                    [1.2, 2, 1.2],
                    [0.8, 1.2, 0.8]])/10

# Apply convolution
convolved_img = convolve2d(img, kernel)

# Display the original and convolved images
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(convolved_img, cmap='gray')
plt.title('Convolved Image')
plt.axis('off')

plt.show()
```

Figure 9: Code Snippet 2

Step2: Horizontal & Vertical Gradients Image

In [5]:
```
kernel_Gx = np.array([[-1, -2, -1],       # Horizontal Edges
                      [ 0,  0,  0],
                      [ 1,  2,  1]])

kernel_Gy = np.array([[-1, 0, 1],         # Vertical Edges
                      [-2, 0, 2],
                      [-1, 0, 1]])

# Apply convolution
Gx = convolve2d(convolved_img, kernel_Gx)
Gy = convolve2d(convolved_img, kernel_Gy)

# Display the original and convolved images
plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(Gx, cmap='gray')
plt.title('Horizontal Convolved Image')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(Gy, cmap='gray')
plt.title('Vertical Convolved Image')
plt.axis('off')

plt.show()
```

Figure 10: Code Snippet 3

Step3: Perform thresholding of the edge strength image

In [7]:
```
# Show the Histogram

plt.hist(gradient_magnitude.flatten(), bins=256, range=[0, 255])
plt.title('Edge Strength Histogram')
plt.show()
```

Figure 11: Code Snippet 4

In [8]:
```
threshold_value = 15

thresholded_image = np.where(gradient_magnitude > threshold_value, 255, 0).astype(float)
```

In [9]:
```
# Show the threshold image

plt.imshow(thresholded_image, cmap='gray')
plt.title('Thresholded Edge Image')
plt.axis('off')
plt.show()
```

Figure 12: Code Snippet 5

In [25]:
```
threshold_values = [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

plt.figure(figsize=(20,15))
for i, threshold in enumerate(threshold_values):
    plt.subplot(3, 4, i+1)  # Create a new figure for each threshold
    plt.imshow(np.where(gradient_magnitude > threshold, 255, 0).astype(float), cmap='gray')
    plt.title(f'Threshold = {threshold}')
    plt.axis('off')
plt.show()
```

Figure 13: Code Snippet 6

Weighted Mean of Kernel and Repeat it

In [10]:
```
# Weighted Kernel

weighted_kernel = np.array([[0.8, 1.2, 0.8],
                    [1.2, 2, 1.2],
                    [0.8, 1.2, 0.8]])/10
```

In [11]:
```
# Apply convolution
convolved_img = convolve2d(img, weighted_kernel)

# Display the original and convolved images
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(convolved_img, cmap='gray')
plt.title('Convolved Image')
plt.axis('off')

plt.show()
```

Figure 14: Code Snippet 7

In [12]:
```
# Apply convolution
Gx = convolve2d(convolved_img, kernel_Gx)
Gy = convolve2d(convolved_img, kernel_Gy)

# Display the original and convolved images
plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(Gx, cmap='gray')
plt.title('Horizontal Convolved Image')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(Gy, cmap='gray')
plt.title('Vertical Convolved Image')
plt.axis('off')

plt.show()
```

Figure 15: Code Snippet 8

In [13]:
```
# Calculate the gradient magnitude
gradient_magnitude = np.sqrt(Gy**2 + Gx**2)

# # Normalize the gradient magnitude to the range 0 to 255
# gradient_magnitude = np.uint8((gradient_magnitude / gradient_magnitude.max()) * 255)


plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(gradient_magnitude, cmap='gray')
plt.title('Edge Strength Image')
plt.axis('off')

plt.show()
```

Figure 16: Code Snippet 9

```
[ ]: plt.hist(gradient_magnitude.flatten(), bins=256, range=[0, 255])
     plt.title('Edge Strength Histogram')
     plt.show()

[ ]: threshold_value = 13

     thresholded_image = np.where(gradient_magnitude > threshold_value, 255, 0).astype(float)

[ ]: # Show the threshold image

     plt.imshow(thresholded_image, cmap='gray')
     plt.title('Thresholded Edge Image')
     plt.axis('off')
     plt.show()

[ ]: threshold_values = [10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

     plt.figure(figsize=(20,15))
     for i, threshold in enumerate(threshold_values):
         plt.subplot(3, 4, i+1)  # Create a new figure for each threshold
         plt.imshow(np.where(gradient_magnitude > threshold, 255, 0).astype(float), cmap='gray')
         plt.title(f'Threshold = {threshold}')
         plt.axis('off')
     plt.show()

[ ]:
```

Figure 17: Code Snippet 10