Coursework 1 Report – Usman Ali

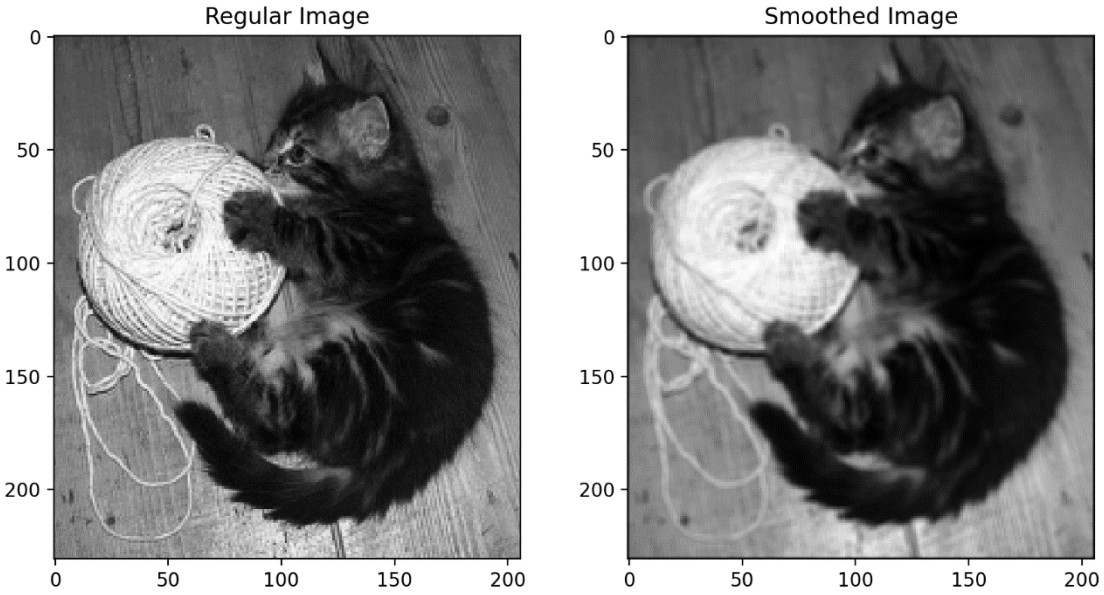

Figure 1

The first step in the experiments involved convoluting the original image with a smoothing kernel and we can see the effect of this convolution from figure 1 above. The smoothing kernel works by reducing noise through averaging the pixels within the area defined by the kernel; this results in a more blurred image.

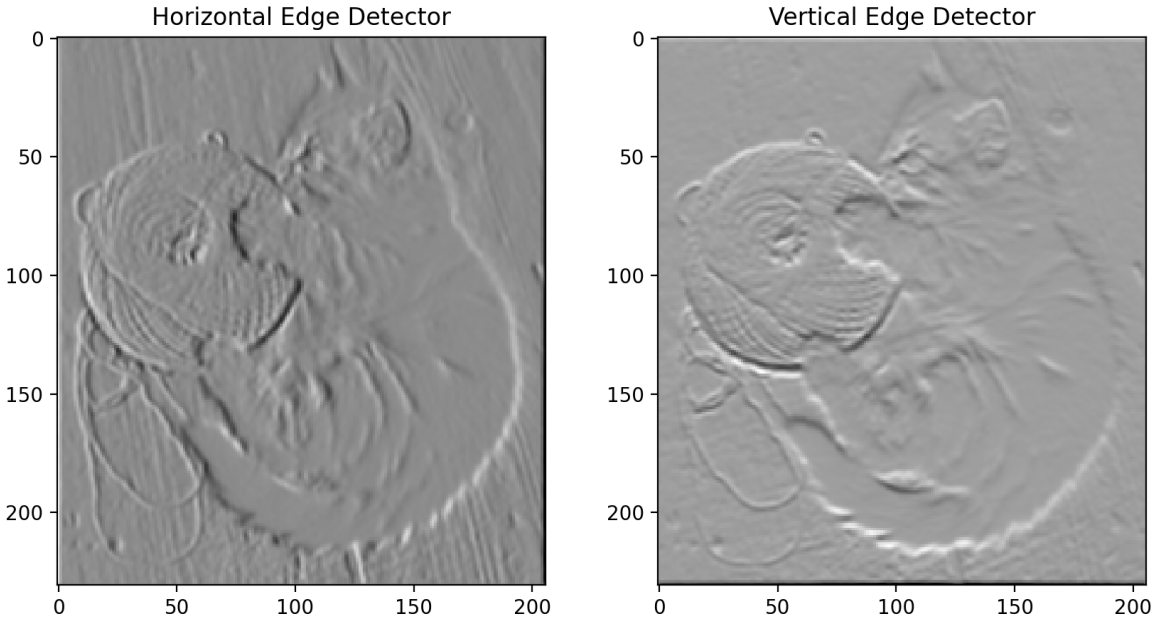

Figure 2

The next step involved convoluting with edge detection kernels to obtain images of the horizontal and vertical edges as can be seen in Figure 2. The Sobel kernel was favoured over the Prewitt kernel for this process due to the fact that it places a higher weight on the central pixels resulting in slightly more enhanced edges and the emphasis on the central pixels makes it somewhat more robust against noise in the image.

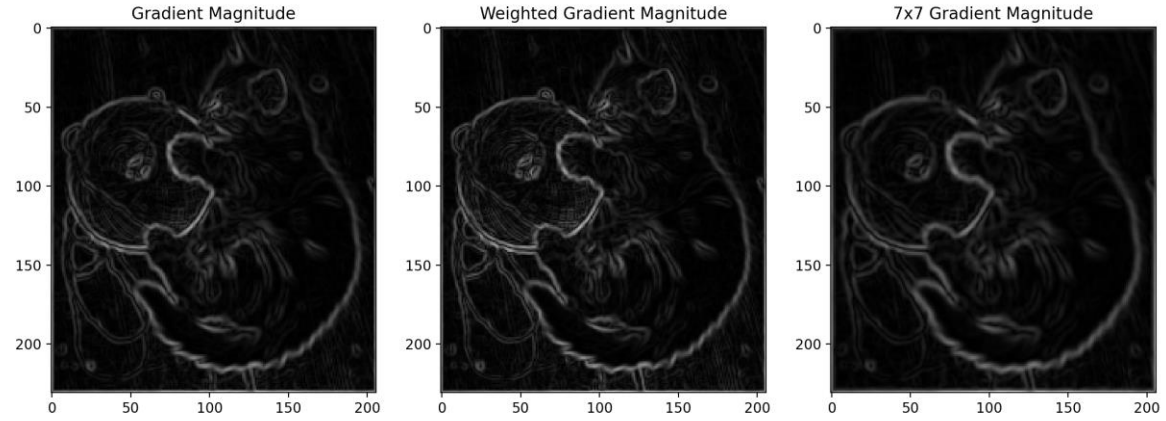

Figure 3

Figure 3 shows three different edge strength images that were obtained using different smoothing kernels. The first image was obtained using a 3x3 averaging kernel whereas the second image was obtained using a 3x3 weighted kernel. Edges appear more pronounced when using weighted-mean smoothing due to the better suppression of noise and finer details. The average kernel treats all pixels in the kernel equally, retaining more noise and resulting in a less distinct edge. Meanwhile, the image on the right uses a 7x7 weighted kernel and as a result the image is more blurred due to the larger kernel as well as significantly reducing noise. However, the edges also appears smoother and less defined with the larger kernel image.

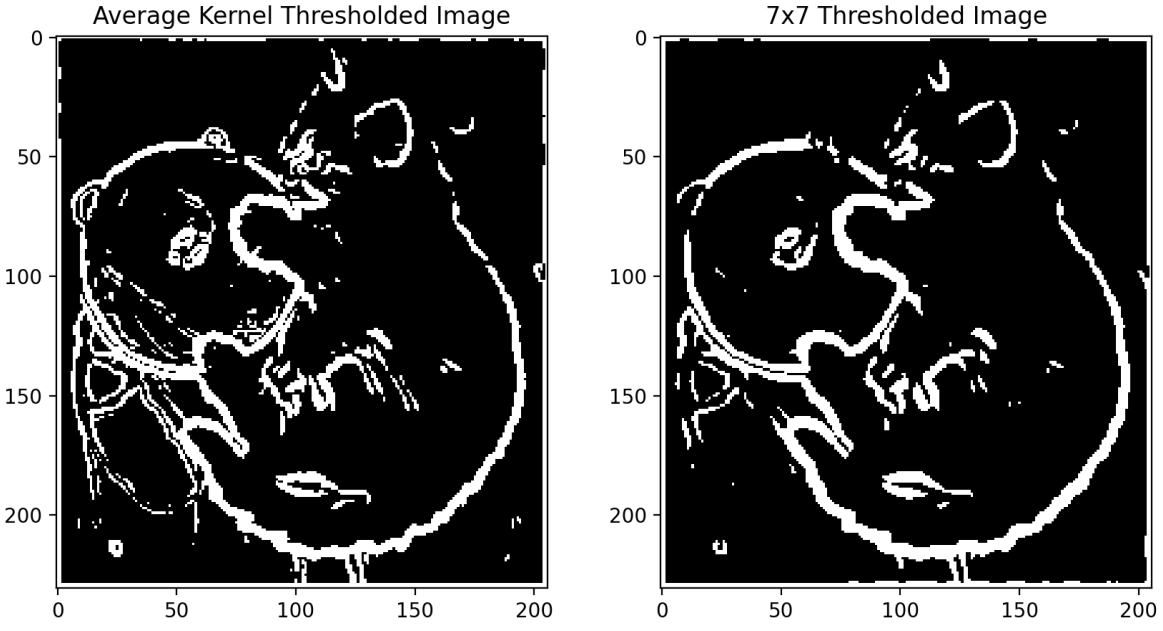

Figure 4

The difference between the 3x3 averaging kernel and the 7x7 weighted kernel is displayed very clearly in figure 4 at a threshold level of 35. There's a significant amount of less noise with the larger kernel but the edges do appear thicker due to the blurring taking place over a larger number of pixels in the kernel. Also, this threshold level was chosen due to the fact that it gives a clear outline of the cat and the ball and minimises noise from fur and other objects in the image.

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

def convolve2d(image, kernel):

    kernel_height, kernel_width = kernel.shape
    pad_height, pad_width = kernel_height // 2, kernel_width // 2

    padded_image = np.zeros((image.shape[0] + (2 * pad_height),
                             image.shape[1] + (2 * pad_width)),
dtype=image.dtype)

    padded_image[pad_height:padded_image.shape[0] - pad_height,
                 pad_width:padded_image.shape[1] - pad_width] = image

    convolved_image = np.zeros_like(image)

    # Perform the convolution
    for y in range(image.shape[0]):
        for x in range(image.shape[1]):
            convolved_image[y, x] = (kernel * padded_image[y:y+kernel_height,
x:x+kernel_width]).sum()

    return convolved_image.astype(np.float32)

def compute_gradient_magnitude(horizontal, vertical):
    gradient_magnitude = np.sqrt(np.square(horizontal) + np.square(vertical))
    gradient_magnitude *= 255.0 / np.max(gradient_magnitude)  # Normalize to
range [0, 255]
    return gradient_magnitude.astype(np.uint8)

def thresholding(image, threshold):
    thresholded_image = np.zeros_like(image)

    for y in range(image.shape[0]):
        for x in range(image.shape[1]):
            # If the pixel value is greater than or equal to the threshold,
set it to maximum
            if image[y, x] >= threshold:
                thresholded_image[y, x] = 255

    return thresholded_image
```

```
# Load the image
image = cv2.imread('kitty.bmp', cv2.IMREAD_GRAYSCALE)
if image is None:
    raise ValueError('Image not found. Make sure the filepath is correct.')

average_kernel = np.ones((3, 3), dtype=np.float32) / 9

# Smoothing image convolution
smoothed_image = convolve2d(image, average_kernel)

# Sobel Kernels
sobel_kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]],
dtype=np.float32)
sobel_kernel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]],
dtype=np.float32)

gradient_x = convolve2d(smoothed_image, sobel_kernel_x)
gradient_y = convolve2d(smoothed_image, sobel_kernel_y)

# Compute the gradient magnitude
gradient_magnitude = compute_gradient_magnitude(gradient_x, gradient_y)

# Step 4: Weighted mean smoothing kernel
weighted_kernel = np.array([[1, 2, 1], [2, 4, 2], [1, 2, 1]],
dtype=np.float32) / 16
weighted_smoothed_image = convolve2d(image, weighted_kernel)
weighted_gradient_x = convolve2d(weighted_smoothed_image, sobel_kernel_x)
weighted_gradient_y = convolve2d(weighted_smoothed_image, sobel_kernel_y)
weighted_gradient_magnitude = compute_gradient_magnitude(weighted_gradient_x,
weighted_gradient_y)


gaussian_kernel_7x7 = np.array([
    [1, 4, 7, 10, 7, 4, 1],
    [4, 12, 26, 33, 26, 12, 4],
    [7, 26, 55, 71, 55, 26, 7],
    [10, 33, 71, 91, 71, 33, 10],
    [7, 26, 55, 71, 55, 26, 7],
    [4, 12, 26, 33, 26, 12, 4],
    [1, 4, 7, 10, 7, 4, 1]
], dtype=np.float32)

gaussian_kernel_7x7 = gaussian_kernel_7x7 / gaussian_kernel_7x7.sum()
gaussian_image = convolve2d(image, gaussian_kernel_7x7)
gaussian_x = convolve2d(gaussian_image, sobel_kernel_x)
gaussian_y = convolve2d(gaussian_image, sobel_kernel_y)
gaussian_gradient_magnitude = compute_gradient_magnitude(gaussian_x,
gaussian_y)
```

```
plt.figure(figsize=(15, 5))
plt.subplot(131), plt.imshow(gradient_magnitude, cmap='gray'),
plt.title('Gradient Magnitude')
plt.subplot(132), plt.imshow(weighted_gradient_magnitude, cmap='gray'),
plt.title('Weighted Gradient Magnitude')
plt.subplot(133), plt.imshow(gaussian_gradient_magnitude, cmap='gray'),
plt.title('7x7 Gradient Magnitude')
plt.show()

threshold_value = 30
edges = thresholding(gradient_magnitude, threshold_value)
edges_7x7 = thresholding(gaussian_gradient_magnitude, threshold_value)

plt.figure(figsize=(10, 5))
plt.subplot(121), plt.imshow(edges, cmap='gray'), plt.title('Average Kernel
Thresholded Image')
plt.subplot(122), plt.imshow(edges_7x7, cmap='gray'), plt.title('7x7
Thresholded Image')
plt.show()
```