COMP37212 Computer Vision - Coursework 1

Chen Bo Calvin Zhang (10403253)

April 14, 2021

Image 1 is the original image used for all the tasks detailed below.

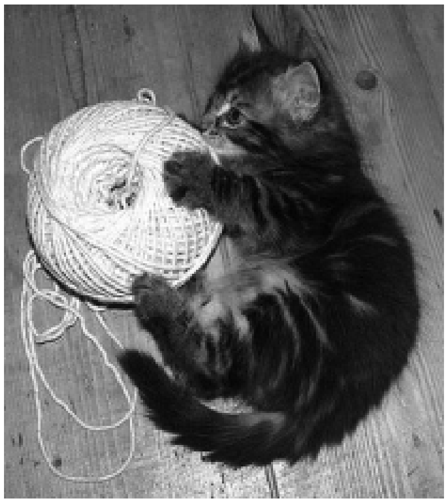


Figure 1: Original image

1 Exercise 1

The first task asked to write a short function that performs the convolution between an image, and a 3×3 structuring element, by performing an explicit looping over the image pixels. Moreover, the edges of the input image should be padded with zeros to deal with the edges and corners of the original image. The function in Listing 1 performs the padding and the convolution, given an image and the kernel we want to convolve the image with as parameters.

I convolved the original image with two smoothing kernels (1). The first one simply takes the average of the intensity of the pixels in the neighbourhood and

```python
def convolve(image, kernel):

    kernel_sum = kernel.sum()

    # pad image with zeros
    pad_img = np.zeros((image.shape[0] + 2, image.shape[1] + 2))
    pad_img[1:image.shape[0]+1, 1:image.shape[1]+1] = image

    # convolved image (here we know the dimension is the same as the original
    # because we are using a 3 x 3 kernel with zero padding and stride of 1)
    conv_img = np.zeros((image.shape[0], image.shape[1]))

    # convolve image
    for y in range(image.shape[1]):  # rows
        for x in range(image.shape[0]):  # columns
            # matrix * matrix is an element-wise multiplication
            conv_img[x, y] = (kernel * pad_img[x: x+kernel.shape[0],
                                  y: y+kernel.shape[1]]).sum()

    return conv_img/kernel_sum
```

Listing 1: Convolution function

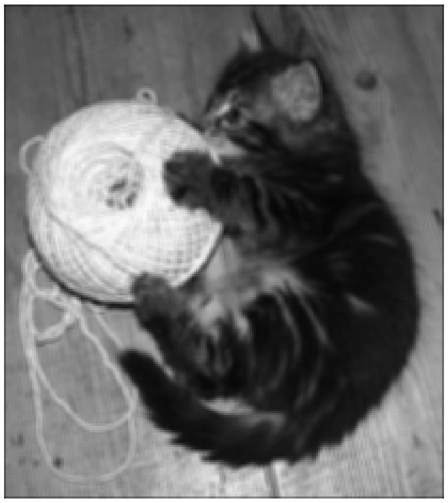
the second one takes a weighted average. The choice of values for the weighted average kernel has been made so that the central pixel will be more important the the surrounding ones. In particular, the values follow a 2-dimensional Gaussian distribution, giving Gaussian blurring as a result. The two kernels can be easily represented using the code in Listing 2.

$$M_{avg} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad M_{weighted} = \begin{bmatrix} 0.5 & 1 & 0.5 \\ 1 & 2 & 1 \\ 0.5 & 1 & 0.5 \end{bmatrix} \tag{1}$$

```python
avg_kernel = np.ones((3, 3))
wavg_kernel = np.array([[.5, 1, .5], [1, 2, 1], [.5, 1, .5]])
```

Listing 2: Averaging kernels

The resulting image for the averaging kernel can be seen in Figure 2a and the one for the weighted average is shown in Figure 2b. The image smoothed with a simple average seems to be smoother overall, but the one obtained using the weighted average has managed to get rid of some of the sharpness on the background while retaining enough information on the edges, without softening them too much.


(a) average

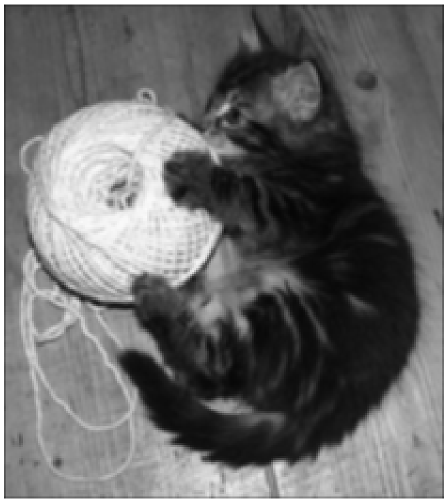

(b) weighted average

Figure 2: Smoothed images

2 Exercise 2

In the second task, I convolved the image with kernels to compute the horizontal and vertical gradient edges, which were then combined to find the edge strength image given by the gradient magnitude. In particular, both the Prewitt (2) and Sobel (3) kernels were used. These kernels are implemented in the code in Listing 3.

$$P_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad P_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{2}$$

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{3}$$

```
prewitt_x = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
prewitt_y = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])
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]])
```

Listing 3: Prewitt and Sobel edge detectors

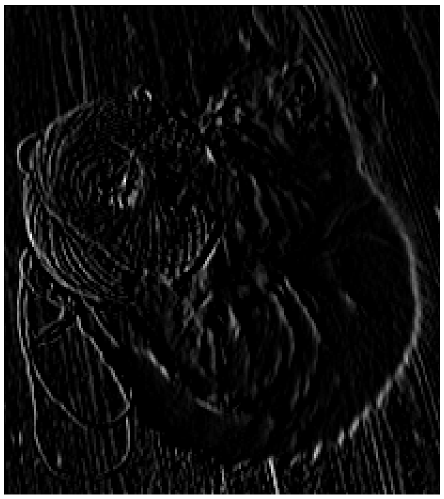
The convolution function (Listing 4) used is slightly different from the one in section 1 in that it does does not pad the image. This is done so that the final convolved image does not have some bands on the side (this is because the zero rows/columns would be strongly detected by the vertical/horizontal edge detectors). The loss of two pixels in each dimension is not very significant in this example as the cat is roughly centred and no important information is lost in the downsizing caused by the convolution. Moreover, the final convolved image is scaled so that its values are in the range $[0, 255]$. This is done as we cannot divide by the sum of values in the kernels, which is always zero (e.g., $(-1)+(-1)+(-1)+0+0+0+1+1+1 = 0$).

```python
def convolve(image, kernel):

    # convolved image (both the x and y axis loose 2)
    # we can hardcode the dimensions because we know we are using a 3 × 3 kernel
    conv_img = np.zeros((image.shape[0]-2, image.shape[1]-2))

    # convolve image
    for y in range(conv_img.shape[1]):  # rows
        for x in range(conv_img.shape[0]):  # columns
            # matrix * matrix is an element-wise multiplication
            conv_img[x, y] = (kernel * image[x: x+kernel.shape[0],
                                             y: y+kernel.shape[1]]).sum()

    return (conv_img / np.max(conv_img)) * 255  # range from 0 to 255
```

Listing 4: Convolution function with no padding

Below we can see the images obtained by convolving the Prewitt kernels in Figure 3 and those from the Sobel kernel in Figure 4. Unlike the images shown in the lectures, these have a black background instead of a grey one; this is however not a problem as it just depends on how the image is exported and it ultimately gives the same result when computing the gradient magnitude. Then, to obtain the edge strength image, we need to compute the gradient magnitude. This can be done with Equation (4) or Listing 5. The resulting images are shown in Figures 5a and 5b.

$$\|\nabla I\| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2} \tag{4}$$


(a) P_x

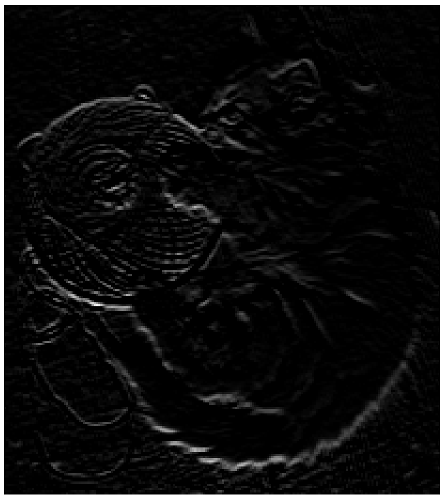
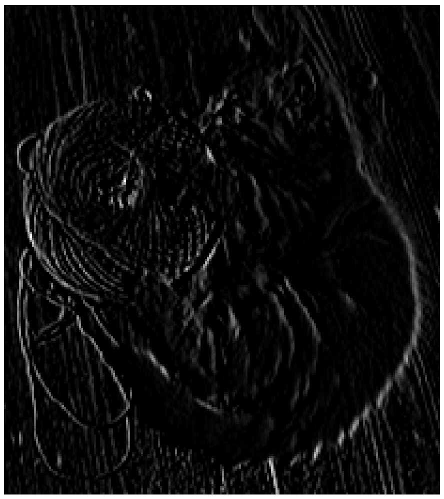

(b) P_y

Figure 3: Output of convolution with Prewitt kernels


(a) S_x

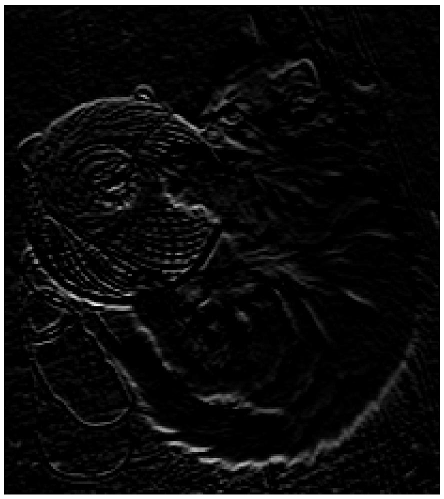

(b) S_y

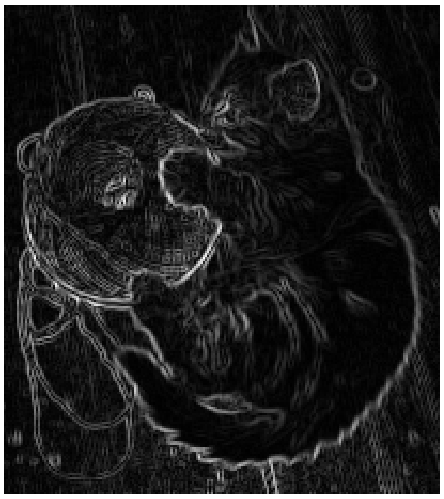
Figure 4: Output of convolution with Sobel kernels

3 Exercise 3

The third task asked to perform thresholding of the edge strength images, and hence display the major edges of the images. Firstly, I had to find a suitable threshold value. This can be done by plotting the histogram of the pixel intensities.

```python
prewitt_grad_img = np.sqrt(np.power(prewitt_x_img, 2) + np.power(prewitt_y_img, 2))
prewitt_grad_img = (prewitt_grad_img / np.max(prewitt_grad_img)) * 255
sobel_grad_img = np.sqrt(np.power(sobel_x_img, 2) + np.power(sobel_y_img, 2))
sobel_grad_img = (sobel_grad_img / np.max(sobel_grad_img)) * 255
```

Listing 5: Compute the gradient magnitudes and edge strength images


(a) Prewitt edge strength image

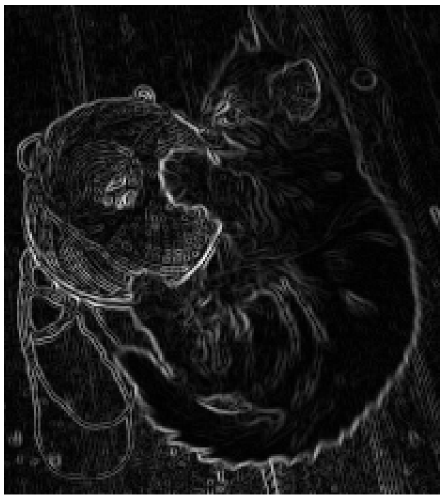

(b) Sobel edge strength image

Figure 5: Edge strength images

Unlike the example shown during the lectures, in this task we are performing thersholding on edge strength images rather than the original (or smoothed) image. This means that we cannot find the right threshold by looking at the dip between two peaks as the edge strength image will have a majority of dark (close to zero intensity) pixels and only few light pixels indicating the edges as seen from Figure 5a and Figure 5b. There seems to be a small dip around intensity 60 in the histogram in Figure 7 with a peak following slightly on the right, but the change is not strong enough to indicate that we can find the threshold with this kind of inspection. Hence, to find the threshold value of 60, I considered the pixel intensity so that most of the pixels (i.e., the dark ones) would be below the threshold and the lighter ones (the minority, hence the edge lines) would be above the threshold.

As we can see from Figure 8, it was not possible to detect the edges of the cat without detecting other patterns as well. In particular, this is confirmed by the edge strength images in Figure 5, which show that the edge strength for the wool ball is actually stronger than the edges for the cat itself, hence choosing a higher threshold would only result in discarding more “cat edges”. The code for the thresholding can be found in Listing 6.

```python
# threshold the gradient magnitude
threshold = 60

prewitt_thresh = np.zeros((prewitt_grad_img.shape[0], prewitt_grad_img.shape[1]))
for y in range(prewitt_grad_img.shape[1]):
    for x in range(prewitt_grad_img.shape[0]):
        if prewitt_grad_img[x, y] > threshold:
            prewitt_thresh[x, y] = 255

sobel_thresh = np.zeros((sobel_grad_img.shape[0], sobel_grad_img.shape[1]))
for y in range(sobel_grad_img.shape[1]):
    for x in range(sobel_grad_img.shape[0]):
        if sobel_grad_img[x, y] > threshold:
            sobel_thresh[x, y] = 255
```

Listing 6: Threhsold the Prewitt and Sobel edge strength images

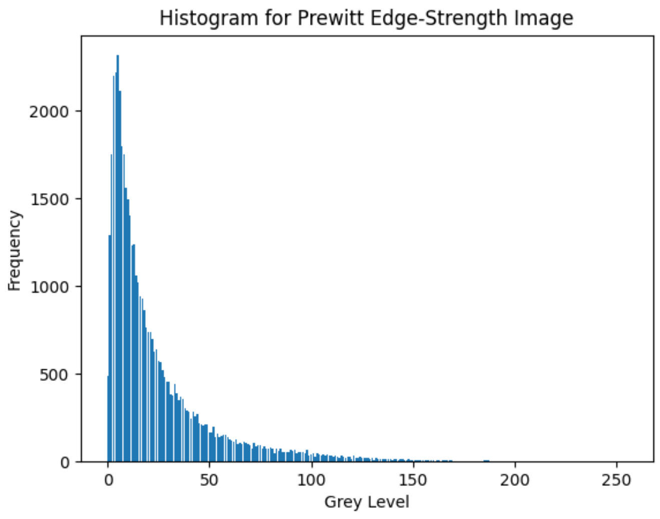


Figure 6: Prewitt intensities histogram

4 Exercise 4

In this task, I performed edge detection on the image smoothed using the weighted average kernel. As expected, we obtained stronger edges in Figure 12 (thicker due to the edges being blurred over more pixels) that capture more of the cat and the

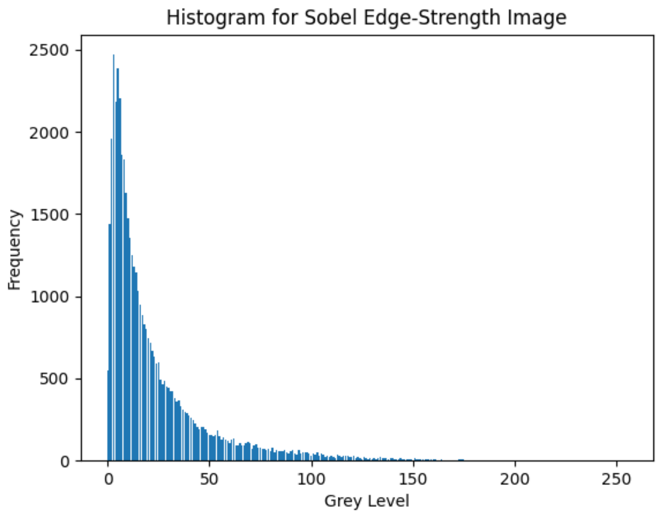
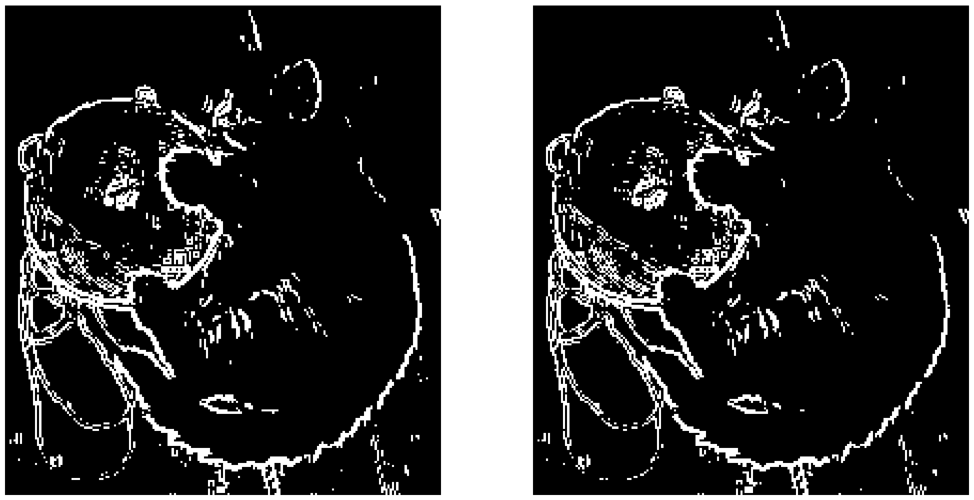


Figure 7: Sobel intensities histogram



(a) Prewitt thresholded image

(b) Sobel thresholded image

Figure 8: Edges (only retaining edges above the threshold)

rounded contour of the wool ball, while ignoring background and texture information. Already from the edge strength images, we could have expected a better detection because we can see that, comparing Figure 5 and Figure 11, the smoothed images give much stronger (brighter) edges around the cat and the wool ball, while discarding edges in the texture of the wooden floor. This result was expected as we know that by smoothing an image, we get rid of noise that could potentially be detected as an edge and strengthen the real edges.

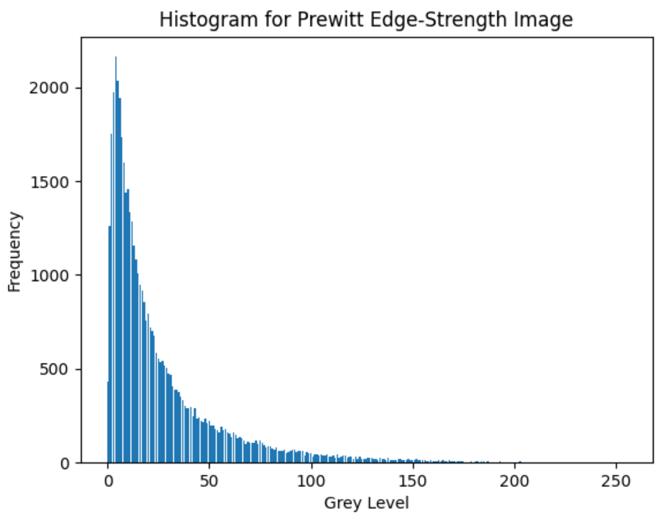


Figure 9: Prewitt intensities histogram on smoothed image

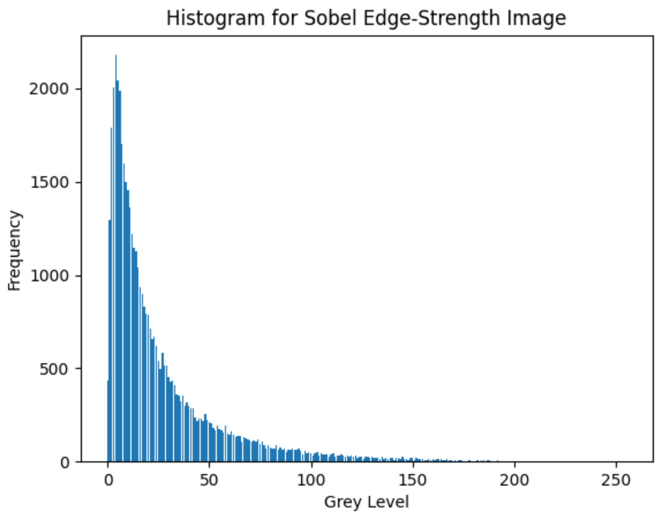
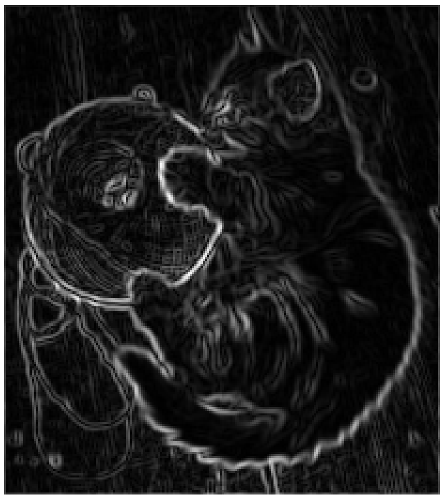


Figure 10: Sobel intensities histogram on smoothed image


(a) Prewitt edge strength image

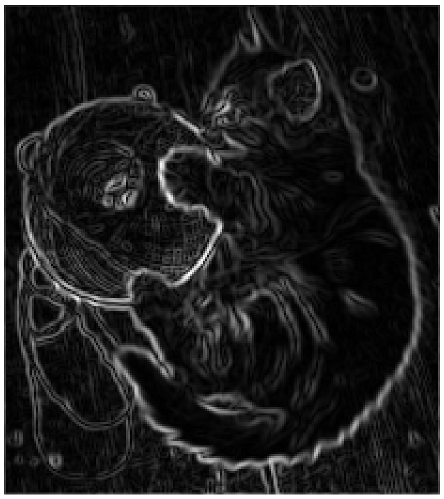
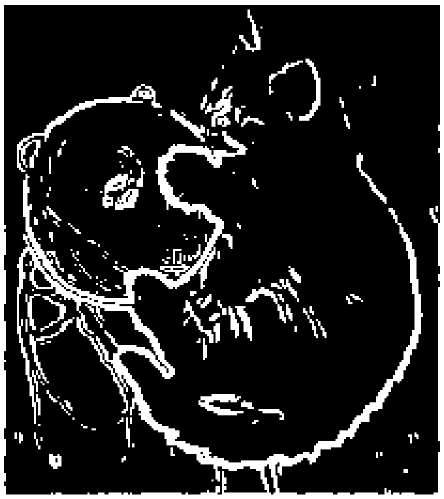

(b) Sobel edge strength image

Figure 11: Edge strength images on weighted average smoothed image


(a) Prewitt thresholded image

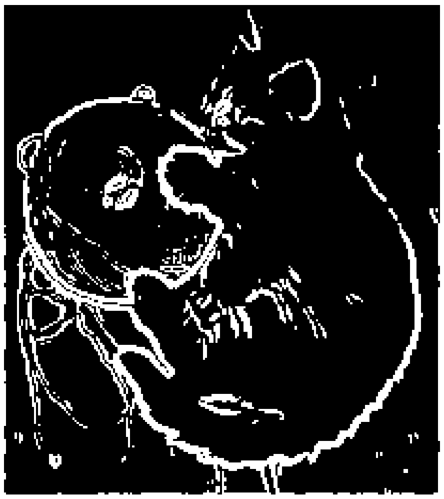

(b) Sobel thresholded image

Figure 12: Edges (only retaining edges above the threshold) on weighted average smoothed image

5 Further Observations

In an attempt to look for multiple peaks in the histograms, a further experiment has been run. The histograms for the original image and the smoothed (weighted average) image were plotted in Figure 15. As we can see in Figure 13, we have multiple peaks and by choosing a threshold of 60 we can detect most of the cat, however we cannot detect the whole animal without also detecting other parts of the image as the cat's pixel intensities vary from very dark ones to light ones.

However, the fact that the threshold for both edge detection and object detection is 60 is a coincidence as the two values are not correlated (e.g., if we had a very white cat on top of a grey floor, we would have had a similar edge strength image, hence a similar edge strength threshold, but a higher threshold for the detection of the cat itself from the original or smoothed image).

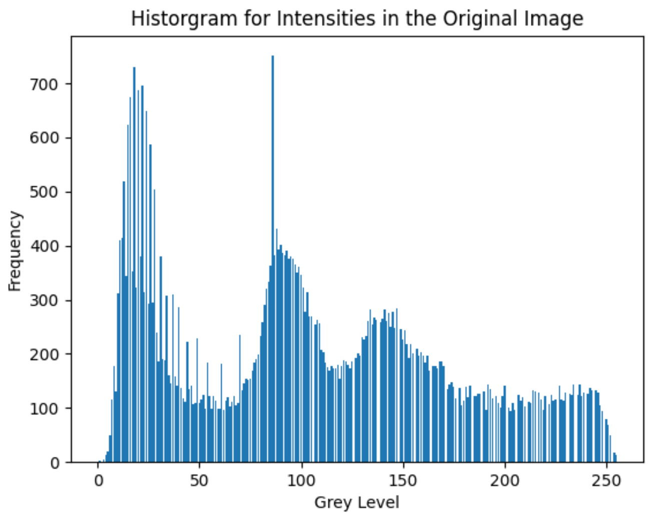


Figure 13: Intensities histogram on the original image

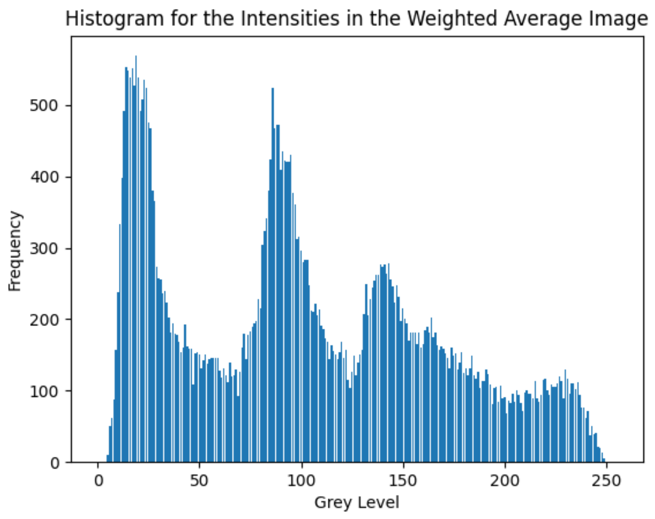
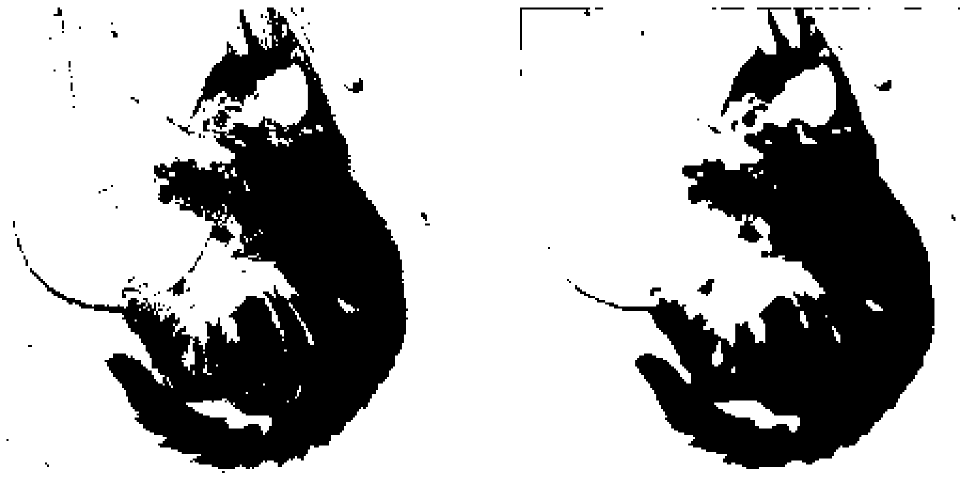


Figure 14: Intensities histogram on smoothed image



(a) Thresholded original image (b) Thresholded smoothed image

Figure 15: Thresholded original and smoothed images