COMP37212 Coursework 1: Convolution & Kernels

Radu Ionut Pirlog
Student ID: 10722128

February 26, 2024

1 Introduction

In this coursework, I explore the fundamental concepts of image processing, particularly focusing on convolution operations and edge detection using kernels.

The coursework consists of several key tasks:

1. **Convolution with 3x3 Kernel**: Implement a function to perform convolution between an image and a 3x3 kernel. This involves padding the image to handle edge and corner cases, and then convolving the image using explicit looping over pixel values.
2. **Gradient and Edge Detection**: Compute horizontal and vertical gradient images by convolving the original image with appropriate kernels. Use the gradients to derive an edge strength image representing the magnitude of the gradients.
3. **Thresholding and Major Edges**: Perform thresholding on the edge strength image to highlight major edges in the image. Explore histogram analysis to determine an optimal threshold value that emphasizes the edges of interest while minimizing noise from other image features.
4. **Weighted Mean Smoothing**: Repeat the edge detection process starting from the weighted mean of the original image. Compare the resulting edge strength images with those obtained from the unprocessed image to understand the impact of smoothing on edge detection.

2 Convolution with 3x3 Kernel

The function convolution I implemented performs convolution between an image and a 3x3 kernel. Here's a breakdown of how it works:

1. **Padding**: Before performing convolution, the input image is padded with zeros on all sides. This padding ensures that the kernel can be centred at each pixel of the image, including those near the edges and corners. Padding helps maintain the spatial dimensions of the output image after convolution.

2. **Looping over image pixels**: The function iterates over each pixel of the input image, excluding the padded border. For each pixel position (i, j) in the original image, the function computes the convolution operation by taking the element-wise product between the 3x3 kernel and the corresponding pixels in the image patch centred around (i, j). The sum of these products gives the value of the output pixel at position (i, j) in the convolved image.

3. **Convolution operation**: The convolution operation involves sliding the kernel over the entire image, performing the element-wise multiplication and summing for each position. This process effectively applies the kernel's weights to the neighbourhood of each pixel, producing a filtered output image.

The convolution operation effectively smooths the image by averaging pixel values in the neighbourhood of each pixel.

3 Gradient and Edge Detection

To compute horizontal and vertical gradient images, I convolve the original image with appropriate kernels that highlight changes in intensity along the horizontal and vertical directions. The horizontal gradient emphasizes changes in intensity from left to right, while the vertical gradient emphasizes changes from top to bottom.

Here's how I achieve this:

Sobel Kernels: We typically use Sobel kernels for computing gradients. These kernels are designed to approximate the derivative of the image intensity function. The Sobel kernel for the horizontal gradient (Sobel X) highlights changes in intensity along the horizontal direction, while the Sobel kernel for the vertical gradient (Sobel Y) highlights changes along the vertical direction.

Convolution: I convolve the original image with the Sobel X kernel to compute the horizontal gradient image and with the Sobel Y kernel to compute the vertical gradient image. This convolution operation computes the rate of change of intensity in the horizontal and vertical directions at each pixel.

Gradient Magnitude: Once I have the horizontal and vertical gradient images, I compute the gradient magnitude image by taking the square root of the sum of squares of the horizontal and vertical gradients at each pixel location. This represents the overall strength of the gradient or the rate of intensity

change in any direction at each pixel.

Edge Strength Image: The gradient magnitude image represents the edge strength. Pixels with higher values indicate stronger edges, while pixels with lower values indicate weaker edges or smooth regions.

4 Thresholding and Major Edges

Thresholding is a crucial step in image processing, particularly in edge detection, where it helps distinguish between regions of interest (e.g., edges) and background noise or smooth regions. In the context of edge detection, thresholding the edge strength image allows us to identify and highlight major edges in the image while minimizing the inclusion of noise or irrelevant details.

Here's how the thresholding process works and how I determine the threshold value:

Thresholding Process: In thresholding, a thresholding value is set, and any pixel in the edge strength image with a value above this threshold is considered part of an edge, while pixels below the threshold are considered background. This results in a binary image where the edges are represented by white pixels and the background by black pixels.

Choosing the Threshold Value: Selecting an appropriate threshold value is crucial to obtaining meaningful results. It involves finding a balance between detecting edges effectively and minimizing the inclusion of noise or irrelevant details. One common approach is to analyze the histogram of the edge-strength image.

Histogram Analysis: The histogram of the edge-strength image provides insight into the distribution of pixel values. Peaks in the histogram correspond to regions with higher edge strength. By examining the histogram, I can identify a threshold value that separates the major edges from the background.

Determining the Threshold: There are several methods for determining the threshold value, such as manual inspection of the histogram, statistical techniques (e.g., Otsu's method), or empirical observation. In this case, manual inspection combined with empirical observation may be suitable. I aim to select a threshold that captures the prominent edges of the cat while minimizing the inclusion of patterns in the fur or wood grain.

Thresholding Result: Once the threshold value is determined, I apply it to the edge-strength image to obtain a binary image where major edges are highlighted. This binary image can then be used for further analysis or visualization.

For example, in Figure 2, thresholding was applied with a default threshold value set to 100. Pixels with values greater than 100 are retained, while all others are considered background and assigned a value of 0, resulting in a binary image.

5 Weighted Mean Smoothing

In this section, I repeated the previous steps starting from the weighted mean of the original image. I compared the resulting edge-strength images and discussed the differences caused by the weighted-mean smoothing.

Weighted mean smoothing, also known as weighted averaging, differs from simple averaging by assigning different weights to each pixel in the kernel. This weighting allows for more control over the smoothing process and can lead to different effects on the resulting image.

When comparing weighted mean smoothing to simple averaging, we observed that weighted mean smoothing tends to preserve edges better while reducing noise (Fig. 3). This is because the weighted average assigns higher importance to central pixels in the kernel, resulting in a smoother transition around edges.

I have tested 5 kernels that have varying weights. They reflect, in order, equal weights, centre weights, edge weights, corner weights and diagonal weights. The following matrices have been used:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$
Equal Weights Kernel

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$
Center Weights Kernel

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$
Edge Weights Kernel

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$
Corner Weights Kernel

$$\begin{bmatrix} 0 & 0 & 1 \\ 0 & 2 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$
Diagonal Weights Kernel

The resulting images are represented in Fig. 4. When convoluting with 3x3 kernels, we can draw several conclusions based on the characteristics of the kernels and the resulting smoothed images:

Equal Weights Kernel: Convolution with an equal weights kernel results in a simple average of pixel values in the neighbourhood. This tends to produce a slight blurring effect on the image while reducing noise. It is effective for general smoothing purposes.

Center Weights Kernel: A centre weights kernel assigns higher weights to the central pixel and lower weights to surrounding pixels. This emphasizes the importance of the central pixel in the smoothing process, resulting in a more focused smoothing effect around edges and details.

Edge Weights Kernel: Edge weights kernel assigns higher weights to pixels along the edges of the kernel. This kernel is designed to preserve edges while still reducing noise. It provides a balance between edge preservation and noise reduction.

Corner Weights Kernel: Corner weights kernel assigns higher weights to corner pixels and lower weights to other pixels. This kernel emphasizes corner details while still smoothing the image overall. It may be useful for enhancing corner features in certain applications.

Diagonal Weights Kernel: Diagonal weights kernel assigns higher weights to diagonal pixels and lower weights to other pixels. This kernel emphasizes diagonal details while smoothing the image. It can be useful for enhancing diagonal features or detecting diagonal edges.

Furthermore, I explored the effects of Gaussian smoothing, which involves convolving the image with a Gaussian kernel. Gaussian smoothing offers more flexibility in adjusting the smoothing effect through the parameter sigma (σ). A higher sigma value results in a smoother image, while a lower sigma value preserves more detail.

Choosing the appropriate sigma value depends on the level of noise in the image and the desired amount of smoothing. A larger sigma value is suitable for images with high levels of noise, while a smaller sigma value is preferred for images with finer details (Fig. 5).

In terms of kernel size, larger kernels offer more extensive smoothing by incorporating a broader range of neighbouring pixels into the computation. However, this broader scope may lead to excessive blurring of edges, as the smoothing effect tends to average out variations in pixel values. Conversely, smaller kernels preserve edges more effectively because they involve fewer neighbouring pixels in the smoothing process. This preservation of edges comes at the cost of potentially less effective noise reduction, as smaller kernels may not adequately average out noisy pixel values.

The choice of kernel size is therefore a trade-off between noise reduction and edge preservation. A larger kernel size is suitable for reducing noise across a wider spatial range but may sacrifice edge sharpness. On the other hand, a smaller kernel size is preferable for preserving edge details but may not effectively eliminate noise, especially in regions with high noise levels.

Furthermore, when using weighted kernels, such as those with centre emphasis or other customized weight distributions, the smoothing effect can be more controlled compared to using a standard averaging kernel. Increased-size weighted kernels can offer a balance between noise reduction and edge preservation, as they provide more smoothing while still emphasizing central pixels.

However, it's important to note that as the size of the kernel increases, the edges of the resulting image may start to become darker or blackened due to the enhanced smoothing effect. This phenomenon occurs because the weighted averaging process tends to suppress high-frequency components, which include edge information. Therefore, careful consideration of the desired balance between noise reduction and edge preservation is essential when selecting the size and type of kernel for image smoothing operations. (Fig. 6,7)

Finally, when selecting a threshold for edge detection, it's essential to consider the characteristics of the image and the application requirements. A threshold that is too low may detect unwanted noise (Fig. 8), while a threshold that is too high may miss important edges (Fig. 9). Experimentation and visual inspection of the resulting edge-strength image are crucial for determining the optimal threshold value.

When using different types of kernels (e.g., normal, weighted, Gaussian) for edge detection, the choice of thresholding method may vary. For example:

1. **Normal Kernels**: With normal kernels, manual thresholding can be effective, depending on the characteristics of the edge-strength image. Manual thresholding allows for fine-tuning based on visual inspection.
2. **Weighted Kernels**: When using weighted kernels, adaptive thresholding may be more suitable, as the weighting of pixel values in the kernel can lead to spatial variations in edge strength. Adaptive thresholding adjusts the threshold value based on the local characteristics of the image, accommodating variations in edge strength across different regions.
3. **Gaussian Kernels**: Gaussian smoothing tends to produce smoother and more uniformly distributed pixel intensities, which can simplify the thresholding process. Manual thresholding may be effective in this case, as the Gaussian smoothing reduces noise and enhances the bimodal distribution of pixel intensities in the edge-strength image.

By exploring these thresholding methods and experimenting with different threshold values, optimal thresholds can be determined to achieve the desired balance between edge detection and noise suppression in edge-strength images obtained using normal, weighted, or Gaussian kernels. From Figure 10 onwards, I explored different outcomes with different kernel sizes.

6 Conclusion

When it comes to the normal average kernel, finding the right balance between kernel size and threshold value is key. If the kernel is too big, we risk losing out on small edge details, while a kernel that's too small might add unwanted noise to our results. Similarly, setting the threshold too low might pick

up non-edge features as edges, leading to noisy outcomes. On the flip side, setting it too high could cause us to miss important edges altogether. So, it's all about finding that sweet spot where we get clear edges without too much noise, ensuring our edge detection works just right. In Figures 10 - 13, it can be observed the impact of manually visualising the histogram and choosing the best values. In my opinion, an average 3x3 smoothing kernel with a threshold value of 35 produces the best outcome in this situation, although a case can be made for the 7x7 kernel with a threshold value of 25 as well.

When comparing the performance of the weighted smoothing kernel to the averaging kernel, it's evident that the weighted kernel excels in identifying edges without sacrificing detail. Comparing Figure 15 to Figure 12, it's clear that the weighted kernel is more adept at handling images with noise that might be picked up, while accurately detecting edges. Moreover, as depicted in Figure 16, even with a larger kernel size, the significant edges are appropriately highlighted, showcasing the effectiveness of this approach as a compelling edge detection method.

In the case of the Gaussian kernel, the sigma value, representing the spread of the normal distribution, is a crucial factor in achieving a balance. Through experimentation, I've observed a close relationship between the kernel size and the sigma value. When the kernel size increases, it's necessary to opt for a larger sigma value as well. This observation is evident from the visual examination of Figure 19 and Figure 20. In Figure 19, where a small sigma value is used, intricate details are erroneously identified as edges. Conversely, in Figure 20, a clearer outline is reproduced, indicating the effectiveness of choosing an appropriate sigma value in conjunction with the kernel size.

In conclusion, effective edge detection requires finding a delicate balance between parameters such as kernel size, threshold value, and sigma value, with careful consideration of their interplay to achieve optimal results tailored to the image characteristics and application requirements.

7 Code Listing

7.1 Convolution Function

```
import numpy as np

def convolution(image, kernel):
    # Get image dimensions
    height, width = image.shape
    # Get kernel dimensions
    k_height, k_width = kernel.shape
    # Calculate padding
    pad_height = k_height // 2
```

```
    pad_width = k_width // 2
    # Pad the image
    padded_img = pad_image(image, pad_height, pad_height,
    pad_width, pad_width)
    # Initialize result
    result = np.zeros(image.shape).astype(dtype=np.float32)
    # Perform convolution
    for i in range(height):
        for j in range(width):
            result[i, j] = np.sum(padded_img[i:i + k_height,
     j:j + k_width] * kernel)
    return result
```

Listing 1: Convolution Function

7.2 Padding Function

```
import numpy as np

def pad_image(image, top, bottom, left, right, value=0):
    padded_image = []
    for i in range(len(image) + top + bottom):
        if top <= i < top + len(image):
            padded_row = [value] * left + list(image[i - top
    ]) + [value] * right
        else:
            padded_row = [value] * (len(image[0]) + left +
    right)
        padded_image.append(padded_row)
    return np.array(padded_image)
```

Listing 2: Padding Function

7.3 Edge Detection Function

```
import numpy as np

def edge_detection(image):
    # Define Sobel kernels for horizontal and vertical
    gradients
    sobel_kernel_x = np.array([
        [-1, 0, 1],
        [-2, 0, 2],
        [-1, 0, 1]
    ], dtype=np.float64)  # Specify the data type as float64

    sobel_kernel_y = np.array([
        [-1, -2, -1],
```

```
        [0, 0, 0],
        [1, 2, 1]
    ], dtype=np.float64)  # Specify the data type as float64

    # Convolve the image with the Sobel kernels
    gradient_x = convolution(image, sobel_kernel_x)
    gradient_y = convolution(image, sobel_kernel_y)

    # Compute edge strength image (gradient magnitude)
    edge_strength_image = np.sqrt(gradient_x ** 2 +
    gradient_y ** 2)

    # Scale the edge strength image to the range [0, 255]
    edge_strength_image_scaled = (edge_strength_image - np.
    min(edge_strength_image)) / (
            np.max(edge_strength_image) - np.min(
    edge_strength_image)) * 255

    # Convert the scaled edge strength image to uint8
    datatype
    edge_strength_image_scaled_uint8 =
    edge_strength_image_scaled.astype(np.uint8)

    return edge_strength_image_scaled_uint8
```

Listing 3: Edge Detection Function

7.4 Histogram Plotting Function

```
import matplotlib.pyplot as plt

def plot_histogram(image, width=15, height=8):
    # Set the figure size
    plt.figure(figsize=(width, height))
    plt.hist(image.flatten(), bins=np.arange(0, 256, 1),
    color='c')  # Adjust bin size for more detail
    plt.title('Image Histogram')
    plt.xlabel('Pixel Value')
    plt.ylabel('Frequency')
    plt.xticks(np.arange(0, 256, 10))  # Add more ticks on
    the x-axis
    plt.show()
```

Listing 4: Histogram Plotting Function

7.5 Thresholding Function

```
def threshold_edges(edge_strength_image, threshold_value=100):
    # Perform thresholding
    edges_threshold = (edge_strength_image > threshold_value) * 255
    return edges_threshold
```

Listing 5: Thresholding Function

7.6 Gaussian Kernel Function

```
import numpy as np

def gaussian_kernel(size, sigma):
    kernel = np.fromfunction(lambda x, y: np.exp(-(x ** 2 + y ** 2) / (2 * sigma ** 2)), (size, size))
    return kernel / np.sum(kernel)
```

Listing 6: Gaussian Kernel Function

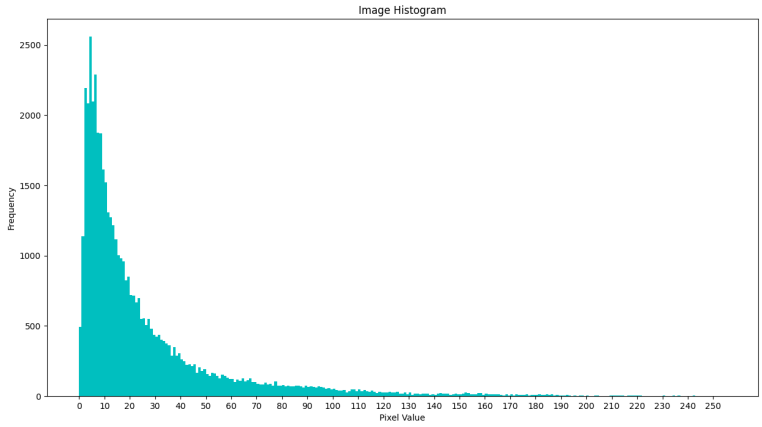

Figure 1: Pixel Value Histogram

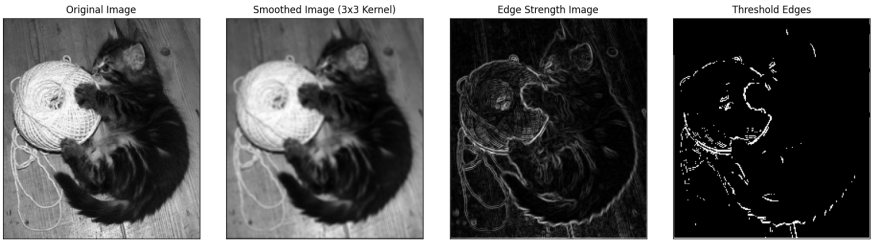

Figure 2: Threshold set to 100

Figure 3: Weighted comparison 3x3

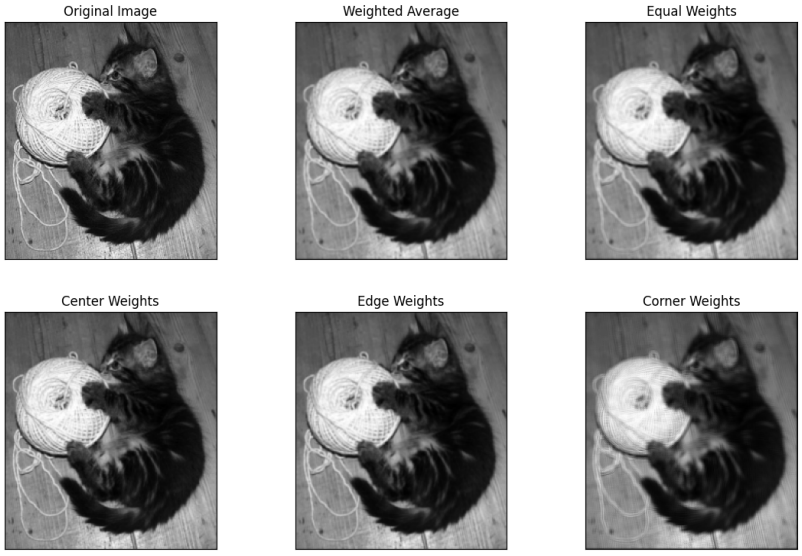

Figure 4: 3x3 Weighted Kernels Comparison

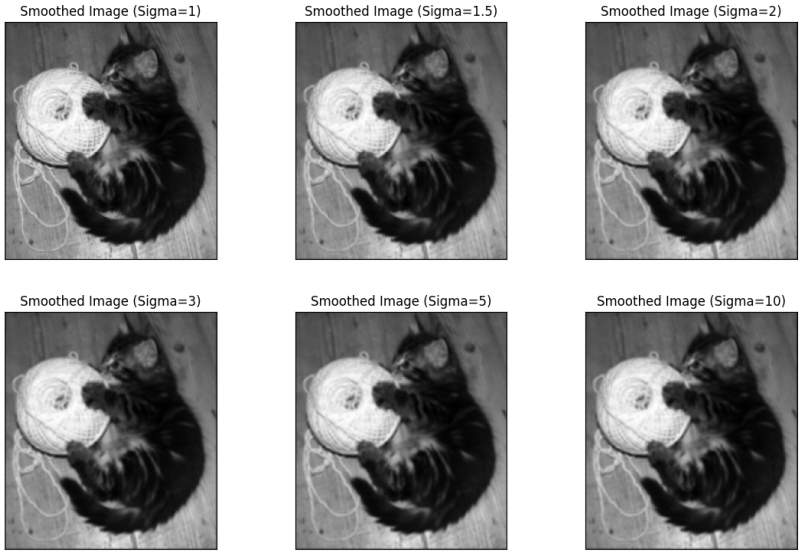

Figure 5: Sigma Values Comparison

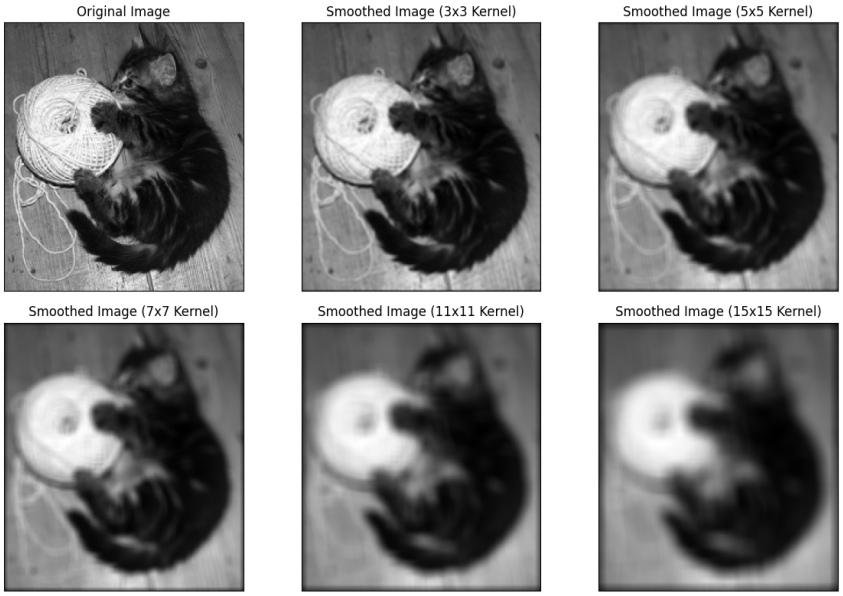

Figure 6: Comparison of Kernel Sizes

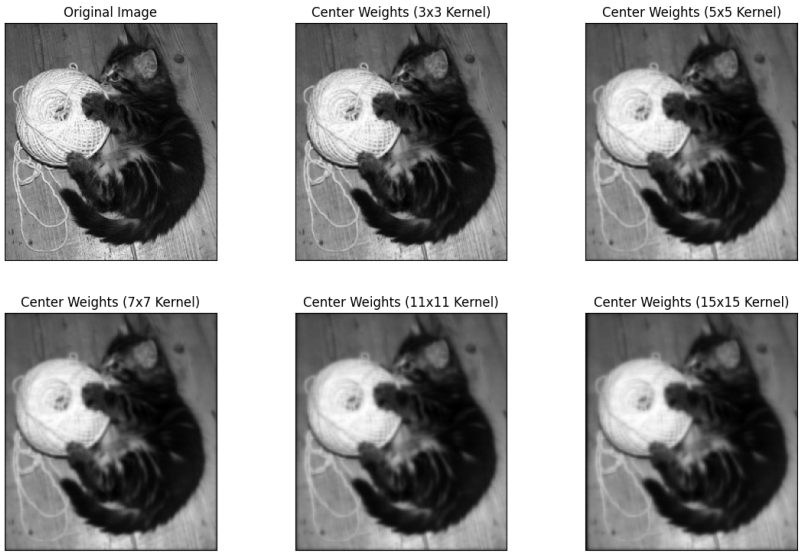

Figure 7: Comparison of Weighted Kernel Sizes

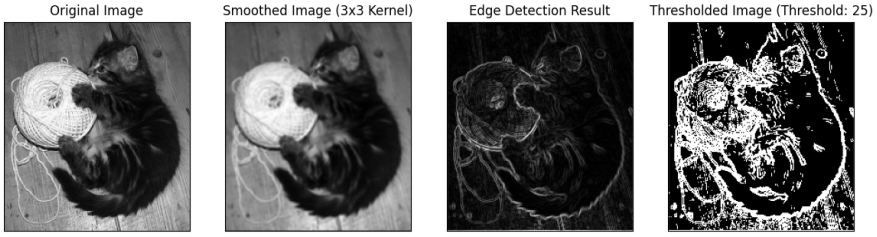

Figure 8: Low Threshold Example

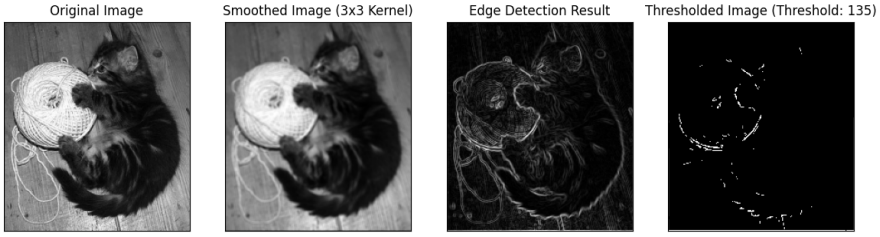

Figure 9: High Threshold Example

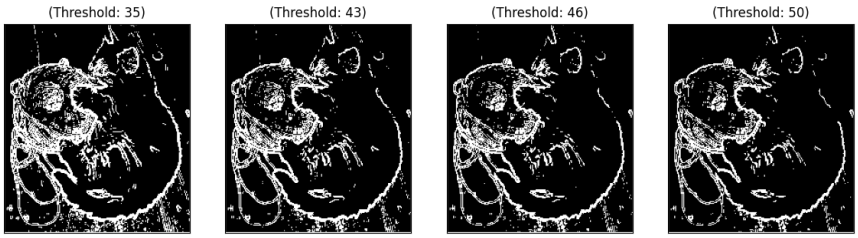

Figure 10: Manual Inspection Threshold for Unsmoothed Image

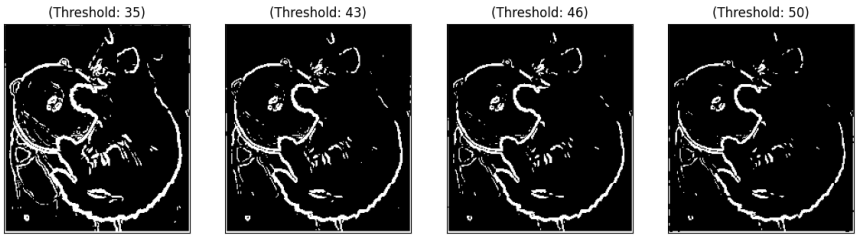

Figure 11: Thresholding for 3x3 Averaging Kernel

Figure 12: Thresholding for 7x7 Averaging Kernel

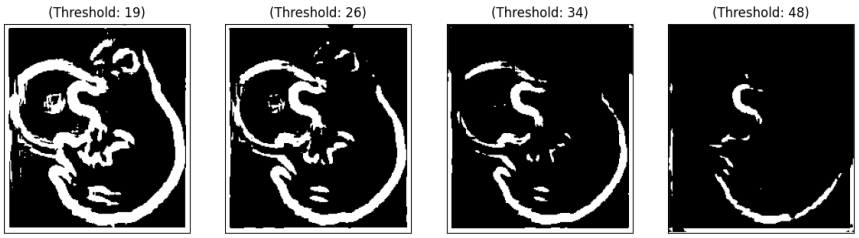

Figure 13: Thresholding for 11x11 Averaging Kernel

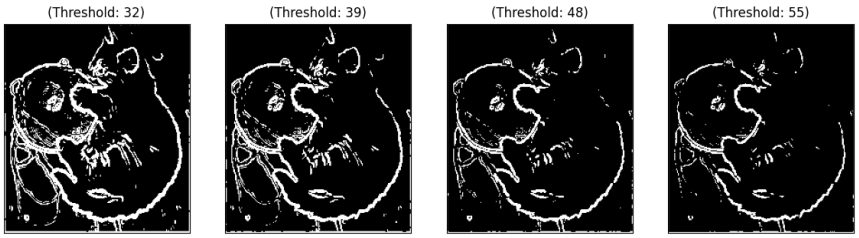

Figure 14: Threshold values for a 3x3 Weighted Smoothing Kernel

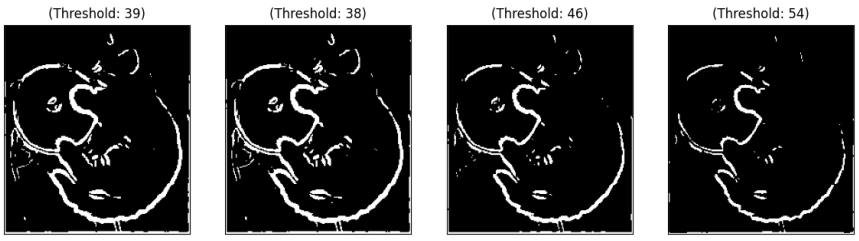

Figure 15: Threshold values for a 7x7 Weighted Smoothing Kernel

Figure 16: Threshold values for a 11x11 Weighted Smoothing Kernel

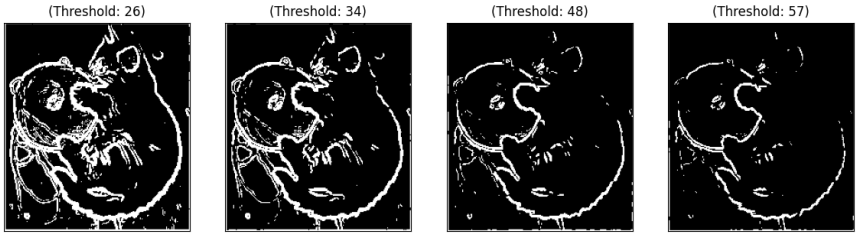

Figure 17: Thresholding with 3x3 Gaussian Kernel and $(\sigma) = 2$

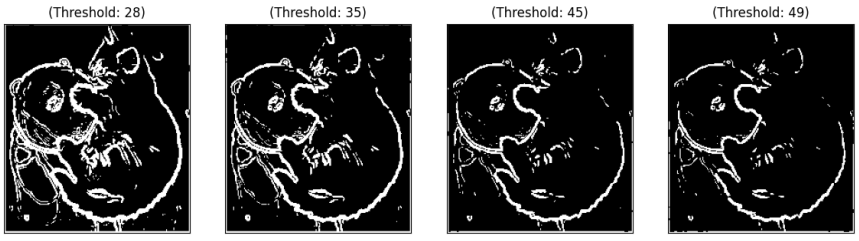

Figure 18: Thresholding with 3x3 Gaussian Kernel and $(\sigma) = 4$

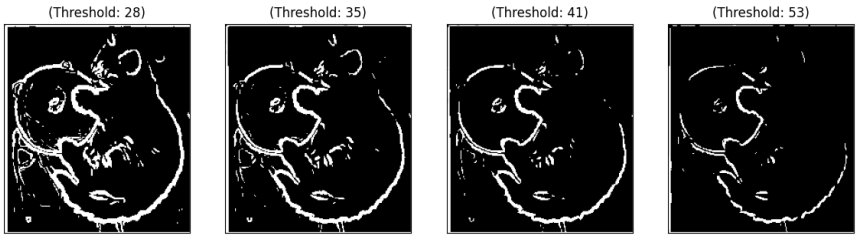

Figure 19: Thresholding with 7x7 Gaussian Kernel and $(\sigma) = 2$

Figure 20: Thresholding with 7x7 Gaussian Kernel and $(\sigma) = 4$