COMP37212-Computer Vision Report

Coursework-1, ID-10849808

Introduction

In this report, we explore the comprehensive application of average smoothing and weighted-average smoothing. The process involves several steps from converting the original image to grayscale image, applying smoothing filters, detecting edges using Sobel Operator and finally performing thresholding. The objective of the experiment is to distinguish the edge detection by using different smoothing filters.

Design/Implementation

Python has been used for the implementation and most of the functions have been written from scratch. Numpy arrays are used for processing matrices and OpenCV is used for I/O operations along with grayscaling the image using cvtColor() function which uses COLOR_BGR2GRAY parameter to change original image to grayscale image

The **Convolution Function** applies a kernel (mean/gaussian) to an image to perform blurring, sharpening and edge detection. The convolution is made of the formula given in the lecture slide (shown in the figure 1.1) where it first pads the image to preserve its dimensions after convolution, normalize the padded image for numerical stability and then iteratively computes the convolution by adding the product of the kernel with image's neighborhood. The implementation of the function is shown in figure 1.2.

The **padImage() function** adds padding around the image, creating a new, larger image to prevent edge and corner issues during convolution. It initializes a padded array filled with constant values, then inserts the original image at the center with this padding technique.

$$\underset{\text{OUTPUT}}{\tilde{I}(x,y)} = \frac{\sum_a \sum_b g(a,b) I(x+a,y+b)}{\sum_c \sum_e g(c,e)} \quad \text{NORMALIZING}$$

Fig 1.1 MATHEMATICAL FORMULA FOR 2D - CONVOLUTION
(Source: Lecture-3 Slides COMP37212)

```python
#---------CONVOLUTION--------------
def convolution(image, kernel):
    height, width = image.shape
    k_height, k_width = kernel.shape

    #padding and converting to 64F
    pad_size_x = k_height // 2
    pad_size_y = k_width // 2
    padded_image = padImage(image, 0, pad_size_x, pad_size_y, pad_size_x, pad_size_y)
    padded_image = cv2.normalize(padded_image, None, 0, 1, cv2.NORM_MINMAX, dtype=cv2.CV_64F)

    result = np.zeros((height, width), dtype=np.float64)
    for i in range(height):
        for j in range(width):
            value = 0.0
            for x in range(k_height):
                for y in range(k_width):
                    value = value + padded_image[i + x, j+y] * kernel[x,y]
            result[i,j] = value

    return result

def padImage(image, value, pad_x1, pad_y1, pad_x2, pad_y2):
    height, width = image.shape
    new_x = height + pad_x1 + pad_x2
    new_y = width + pad_y1 + pad_y2
    padded_result = np.full((new_x, new_y), value, dtype=image.dtype)
    padded_result[pad_x2 : height+pad_x2, pad_y1 : width+pad_y1] = image
    return padded_result
```

Fig 1.2 CONVOLUTION FUNCTION

Part-1: Average (Mean) Smoothing Filter

The average filter is a simple smoothing filter for convolution. It creates a matrix where all elements are equal, indicating equal weighting for all pixels under the kernel. This filter reduces noise and detail, however, it does not consider the neighbouring pixels which may not represent the same object as the selected pixels and will have an equal amount of influence on the resulting pixel. The implementation of the filter is shown in figure 1.3.

```python
#-------------AVERAGE SMOOTHING-------------
def averageKernel(size):
    average_kernel = np.ones((size,size), dtype = np.float64) / (size*size)
    print(average_kernel)
    return average_kernel

#making average kernel
average_kernel = averageKernel(size=7)
#calling average convolved image
average_smooth_image = convolution(gray_kitty_image,average_kernel )
#to normalize from [0,255] arrays to uint8 to display image
average_smooth_image = cv2.normalize(average_smooth_image, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
```

Fig 1.3 AVERAGE SMOOTHING FILTER

Part-2: Weighted-Average (Gaussian) Smoothing Filter

The weighted average filter smoothes the image using gaussian function based on size and standard deviation (sigma). The kernel values decrease with distance from the center, applying more weight to closer pixels following the Gaussian distribution. Application of this kernel to an image, effectively blurs the image, reduces the noise and detail better as compared to the mean filter. The blurred output image helps in better edge detection. The implementation of the filter is shown in figure 1.4.

```
#--------------WEIGHTED-AVERAGE (GAUSSIAN) SMOOTHING--------------
def gaussianKernel(size, sigma):
    #gaussian kernel
    size = int(size) // 2
    x, y = np.mgrid[-size:size+1, -size:size+1]
    g = ( 1 / ( 2 * np.pi * sigma**2 )) * np.exp( - ( (x**2 + y**2) / (2* sigma**2) ) )
    weighted_mean_kernel = g / np.sum(g)

    return weighted_mean_kernel
```

Fig 1.4 WEIGHTED-AVERAGE SMOOTHING FILTER

Part-3: Edge Detection

Image gradients are obtained with Sobel Operators along x and y directions for images from average and gaussian smoothing respectively to highlight the edges (as shown in the figure 1.10). Horizontal and vertical gradients are normalized to display directional edge information. The edge (gradient) magnitude is calculated from the formula in the lecture slides and normalized to fit within the 0-255 range.

```
#horizontal and vertical gradients
avg_gradient_x_image = convolution(average_smooth_image, along_x)
avg_gradient_y_image = convolution(average_smooth_image, along_y)

#edge strength after combined image
avg_gradient_magnitude = np.sqrt((avg_gradient_x_image.astype(np.float64) **2) + (avg_gradient_y_image.astype(np.float64) **2) )

#normalizing
avg_gradient_magnitude = cv2.normalize(avg_gradient_magnitude, None, 0, 255, cv2.NORM_MINMAX, dtype=cv2.CV_8U)
#to normalize from [0,255] arrays to uint8 to display image
avg_gradient_x_image = cv2.normalize(avg_gradient_x_image, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
avg_gradient_y_image = cv2.normalize(avg_gradient_y_image, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)
```

Fig 1.5 EDGE GRADIENTS AND MAGNITUDE (for average filter image) (similar for gaussian)

Part-4: Thresholding

A simple thresholding logic is used for segmenting objects (cat, woolen ball, grains) from the background. It compares each pixel of the image with the threshold value to effectively segment the image (to detect the edge of the cat).

Experiments & Results

Experiment-1: Kernel Size

Smoothing kernels are experimented with different kernel sizes (3x3, 5x5, 7x7). For weighted - average kernel, the sigma with value 2.50 is kept constant over all sizes. The results of the experiment are shown in figure 1.7.

- **Outcome:**
    - For kernels with sizes 3x3 and 5x5 , it becomes apparent that there is not much difference in smoothing of the image, however, it is observable that size 5x5 makes the image slightly blurrier
    - For kernel size of 7x7, it facilitates better object detection since it reduces noise and lessens details. It smooths the image significantly to focus more on large structure and shapes in the image.

Experiment-2: Sigma Values

From Experiment-1, 7x7 kernel size is most accurate to have a good blurred image. Now, in this experiment, we use different values of sigma (0.2, 1.0, 2.5) for the gaussian kernel. The results of the experiment are shown in figure 1.8.

- **Outcome:** Increasing the sigma value from 0.2 to 2.5 progressively blurs the image, reduces noise and small scale structures (details). Sigma value of 0.2 preserves most of the detail and noise, while sigma 2.5 makes many details and edges indistinct which *helps in detecting the objects* in the image.

Experiment-3: Threshold Values

From the above two experiments, we can say that the kernel size of 7x7 and sigma value of 2.5 gives the better edge detection of the object. In this experiment, we use different thresholds (18, 20, 25, 30, 35, 40) to see the edges of the cat with differences in noise levels. The results of the experiment are shown in figure 1.9.

- **Outcome:** After testing different threshold values, it is concluded that the best threshold value is 21 which gets the outline of major features of the cat and woolen ball.

Conclusion

In conclusion, the experiments and results show that 7x7 kernel size combined with a sigma value of 2.5 for gaussian smoothing and threshold value of 21 is optimal for edge detection in the given image. This reduces noise and lessens minor details which allows for clearer detection of objects. Whereas, an average filter with size 7x7 significantly detects the object, however, it also detects all the noises and details. Figures 1.6, 1.11, 1.12 show the difference between two edge images. Therefore, the experiment depicts the importance of selecting appropriate kernel size, sigma and threshold values for precise edge detection.

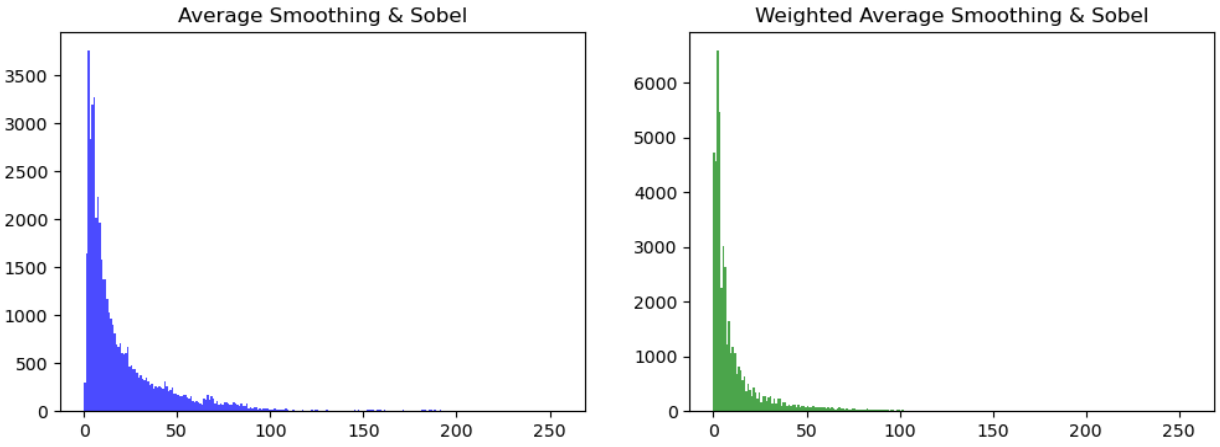


FIG. 1.6 Histograms; LEFT: Average Kernel, RIGHT: Weighted-Average Kernel

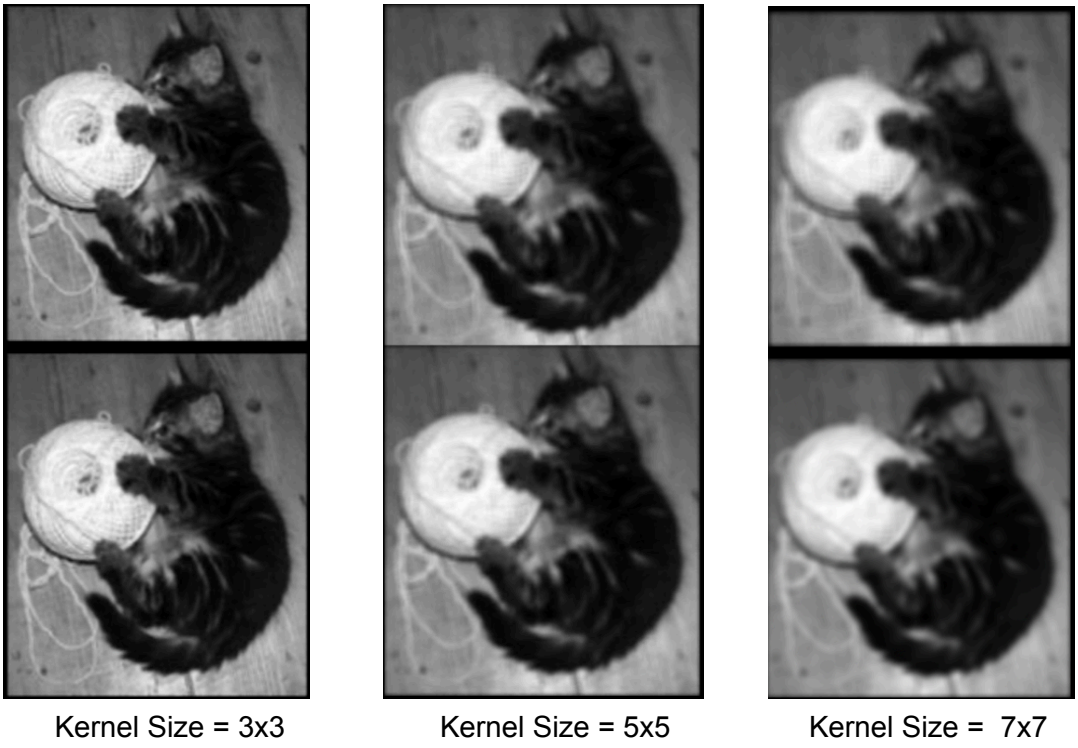
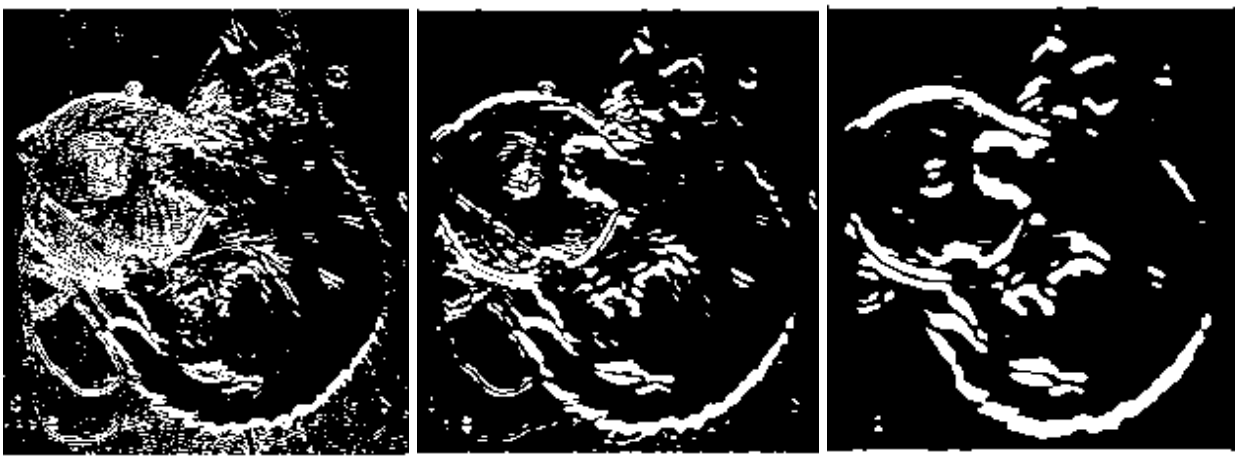


FIG. 1.7 Effect of Different Kernel Size (top row: average convolved image, bottom row: gaussian convolved (sigma=2.50))



Sigma = 0.2 Sigma = 1.0 Sigma = 2.5

FIG. 1.8 Effect of Different Sigma Value on Gaussian Filter (threshold value = 20)

Threshold = 21 Threshold = 30 Threshold = 40

FIG. 1.9 Effect of Different Threshold Values With 7x7 Gaussian Filter (sigma=2.5)

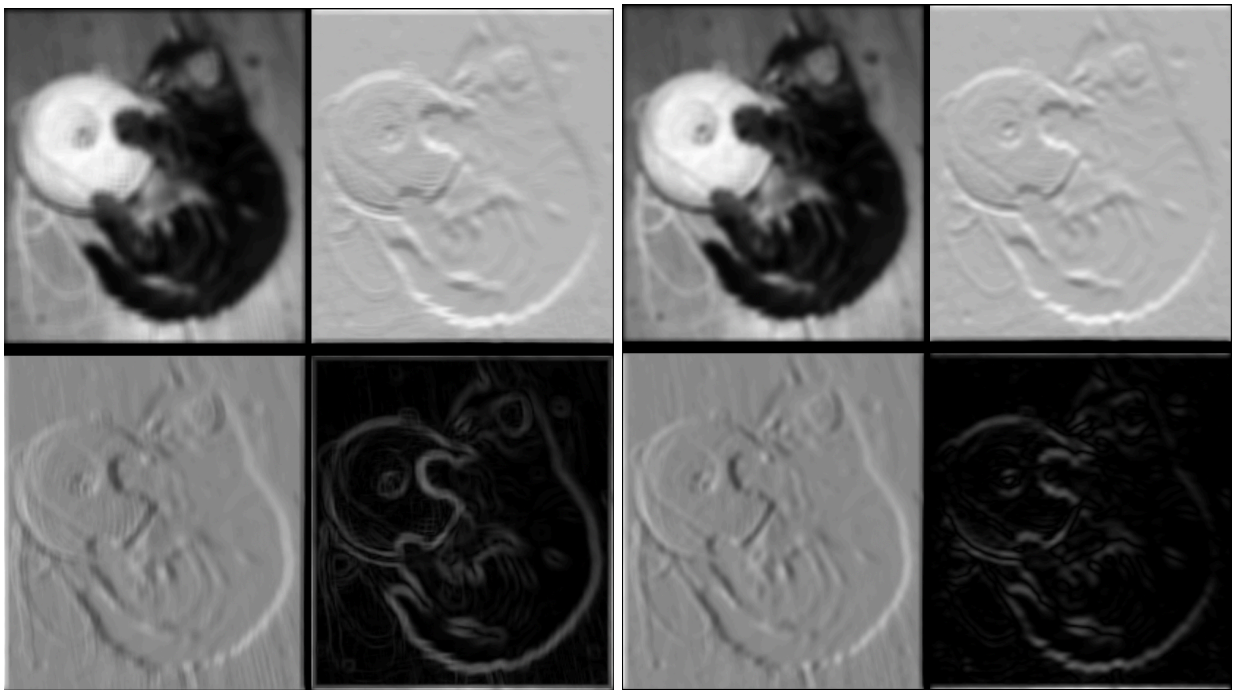


FIG. 1.10 Image Gradients and Magnitude; LEFT: Average Filter, RIGHT: Gaussian Filter
(top-left: smoothed image, top-right: Y-gradient, bottom-left: X-gradient, bottom-right: edge strength)

FIG 1.11 Edge Detected - Gaussian Kernel

FIG 1.12 Edge Detected - Average Kernel

Appendix

- Loading image using opencv with error handle along with grayscaling:

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

#load image
kitty_image = cv2.imread('kitty.bmp')
cv2.namedWindow('Source Image')
cv2.imshow('Source Image', kitty_image)

# Check for success of opening of the image
if kitty_image is None:
    print('Error: failed to open', kitty_image)
    sys.exit()

# Convert to greyscale and save it
gray_kitty_image = cv2.cvtColor(kitty_image, cv2.COLOR_BGR2GRAY)
cv2.namedWindow('Grayscaled Image')
cv2.imshow('Grayscaled Image', gray_kitty_image)
```

- Sobel Operators: (used in image gradients in Figure. 1.5.)

```
along_x = np.array([[-1,0,1],
                    [-2,0,2],
                    [-1,0,1]], dtype=np.float64)
along_y = np.array([[-1,-2,-1],
                    [0,0,0],
                    [1,2,1]], dtype=np.float64)
```

- Saving image with spacebar and closing all windows:

```
# Wait for spacebar press before closing,
# otherwise window will close without you seeing it
while True:
    k = cv2.waitKey(1)
    if k == ord(' ') or k == ord('\t'):
        print('Saving average_smooth_image.png')
        cv2.imwrite('average_smooth_image.png', average_smooth_image)
        print('Saving avg_gradient_x_image.png')
        cv2.imwrite('avg_gradient_x_image.png', avg_gradient_x_image)
        print('Saving avg_gradient_y_image.png')
        cv2.imwrite('avg_gradient_y_image.png', avg_gradient_y_image)
        print('Saving avg_gradient_magnitude.png')
        cv2.imwrite('avg_gradient_magnitude.png', avg_gradient_magnitude)
        #saving the graph
        print('Saving img_mean_hist.png')
        fig.savefig("histogram.png")
        print('Saving thresholded_image.png')
        cv2.imwrite('thresholded_image.png', thresholded_image)
        print('Saving gaussian_smooth_image.png')
        cv2.imwrite('gaussian_smooth_image.png', gaussian_smooth_image)
        print('Saving gaussian_gradient_x_image.png')
        cv2.imwrite('gaussian_gradient_x_image.png', gaussian_gradient_x_image)
        print('Saving gaussian_gradient_y_image.png')
        cv2.imwrite('gaussian_gradient_y_image.png', gaussian_gradient_y_image)
        print('Saving gaussian_gradient_magnitude.png')
        cv2.imwrite('gaussian_gradient_magnitude.png', gaussian_gradient_magnitude)
        print('Saving thresholded_gaussian_image.png')
        cv2.imwrite('thresholded_gaussian_image.png', thresholded_gimage)
        # print('Saving img_edge_comparison.png')
        # cv2.imwrite('img_edge_comparison.png', img_edge_comparison)
        break

cv2.destroyAllWindows()
```