Rajat Sharma
40196467
Concordia University
COMP 478/6771 Image Processing
Assignment 2 - Due Date: October 17, 2023

Part I: Theoretical questions

1. (a) (4 points) What is the limiting effect (or the ultimate result) of repeatedly dilating or eroding an image? Assume that a trivial (one point) structuring element is not used.

When an image undergoes repeated dilation with a non-trivial structuring element, its size expands indefinitely as the structuring element is replicated at each pixel. This replication adds new pixels to the foreground, causing the boundaries to progressively enlarge. As this process iterates, the image encompasses a continually growing area.

Conversely, the repeated erosion of an image using a non-trivial structuring element leads to unbounded shrinking. Erosion diminishes the foreground region by comparing the structuring element to the image at each pixel and removing pixels that do not entirely fit within their corresponding image regions. With each successive erosion, the foreground region contracts, and the image steadily approaches a vanishing point.

(b) (4 points) What is that smallest image from which you can start in order for your answer
in part (a) to hold?
Note: Answer the above questions separately for dilation and erosion.

To observe the effects of repeated dilation and erosion, the smallest image should contain a structural element larger than a single pixel. This is because dilation and erosion operations replicate or remove the structuring element at each pixel in the image. When the structuring element is only a single pixel, these operations have a local impact, altering individual pixels without influencing the overall structure of the image.

2. (8 points) Explain why Hough transform for lines cannot be carried out in the Cartesian (x, y)
coordinate system. Give details on how Hough transform for line searching is done on a set
of n points.

Spatial relationships can be analyzed through the use of the Cartesian coordinate system, denoted by the (x, y) axes. However, when applied directly to line detection tasks using the Hough transform, challenges arise. Traditional methods for expressing lines in Cartesian coordinates, such as the point-slope form (y - y1 = mx - x1) or slope-intercept form (y = mx + b), face limitations. These forms struggle with representing vertical lines (with an infinite slope) and lines crossing the origin with arbitrary slopes.

To overcome these constraints and efficiently detect lines in images, the Hough transform employs a parameterized approach, converting the Cartesian coordinate space into a parameter space. This parameter space is often depicted as a 3D accumulator array. The three dimensions of this array correspond to the parameters defining a line:

ρ (rho): The distance between the line and the origin.
θ (theta): The angle between the line and the x-axis.
b (beta): The y-axis intercept for lines with a non-zero slope.
The Hough Transform for line detection involves representing each point in polar coordinate space and using the accumulator array to detect lines. The process includes:

Parameterization of Lines:

Represent each point (x, y) as a line in polar coordinates (ρ, θ), where ρ is the distance from the origin to the closest point on the line, and θ is the angle between the x-axis and the line perpendicular to the detected line.

Accumulator Array Initialization:

Create an accumulator array with dimensions corresponding to the range of ρ and θ values.
Each cell in the accumulator array represents a potential line in Cartesian space.

Voting:

For each point (x, y), calculate multiple ρ and θ values representing lines passing through that point.
Increment the corresponding cells in the accumulator array for each calculated (ρ, θ) pair.

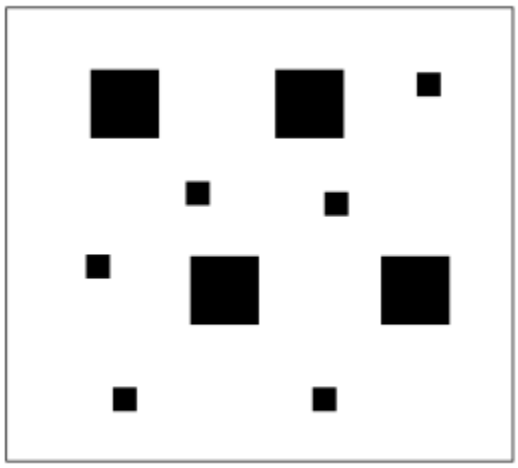
Peak Detection:

Identify peaks in the accumulator array, indicating potential lines in Cartesian space.
Threshold the accumulator array to detect significant peaks.

Conversion to Cartesian Space:

Convert the detected peaks back to Cartesian space to obtain the parameters (slope and intercept) of the detected lines.

3. (8 points) Here is a binary image of squares of two different sizes. Assume that the sizes of
the squares are known, can you devise an algorithm using the knowledge you have learnt to
obtain the number of the bigger squares automatically?



Algorithm:

1. Image Loading:

Load the binary image containing squares of two different sizes into the program.
2. Hough Transform Setup:
a. Set Square Parameters:
- Define the parameters of the squares, including the side lengths of the smaller and larger squares.
b. Initialize Accumulator:
- Create a 3D accumulator array to store the Hough transform votes for each possible square configuration.

3. Hough Transform Calculation:
a. Edge Detection:
- Apply edge detection to the image to identify potential square boundaries. Techniques like Canny edge detection can be used.
b. Hough Transform Voting:

- For each edge point, consider all possible squares that can pass through it. Calculate the Hough transform parameters (ρ, θ, β) for each square and increment the corresponding cell in the accumulator array.

4. Peak Detection:
a. Thresholding:
- Apply a threshold to the accumulator array to identify local maxima, representing the most likely square positions.
b. Square Parameter Extraction:
- Extract the Hough transform parameters (ρ, θ, β) for each peak in the accumulator array. These parameters correspond to the position, orientation, and size of the detected square.

5. Square Classification:
a. Size Comparison:
- Compare the extracted square size to the given sizes of the smaller and larger squares.
b. Square Grouping:
- Assign each detected square to the smaller or larger square group based on the size comparison.

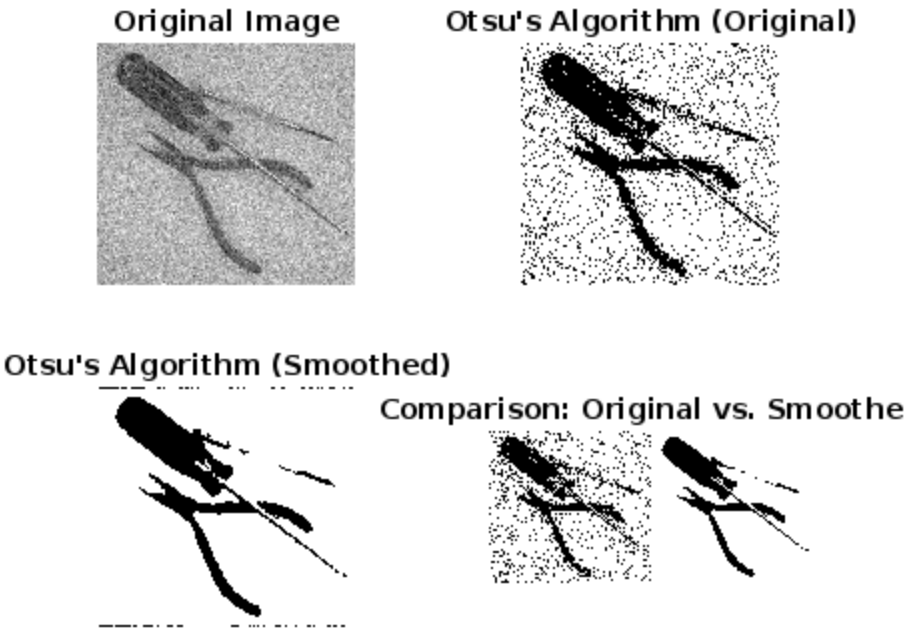
6. Bigger Square Count:
a. Selection:
- Identify the group corresponding to the larger squares.
b. Counting:
- Count the number of squares in the selected group. This provides the final count of bigger squares in the image.

Part II: Programming questions (26 points)

1. (10 points) Download the image (tools_noisy.png) from the course webpage. This image
contains noise.
a) (5 points) Implement and apply Otsu's algorithm to the image and show the result.
b) (5 points) Smooth the image by a 5 by 5 averaging filter, then apply Otsu's algorithm and
show the result. Compare the results in a) and b).

```
% Load the image
originalImage = imread('tools_noisy.png');

% Display the original image
figure;
```

```
subplot(2, 2, 1);
imshow(originalImage);
title('Original Image');

% a) Apply Otsu's algorithm to the original image
thresholdOriginal = graythresh(originalImage);
binaryImageOriginal = imbinarize(originalImage, thresholdOriginal);

% Display the result of Otsu's algorithm on the original image
subplot(2, 2, 2);
imshow(binaryImageOriginal);
title('Otsu''s Algorithm (Original)');

% b) Smooth the image with a 5 by 5 averaging filter and apply Otsu's
algorithm
smoothedImage = imfilter(originalImage, fspecial('average', [5 5]));
thresholdSmoothed = graythresh(smoothedImage);
binaryImageSmoothed = imbinarize(smoothedImage, thresholdSmoothed);

% Display the result of Otsu's algorithm on the smoothed image
subplot(2, 2, 3);
imshow(binaryImageSmoothed);
title('Otsu''s Algorithm (Smoothed)');

% Compare the results
subplot(2, 2, 4);
imshowpair(binaryImageOriginal, binaryImageSmoothed, 'montage');
title('Comparison: Original vs. Smoothed');
```



Comparison between the results in a) and b):

Smoothing Effect:

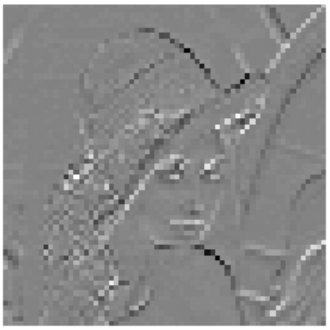
Original Image:
Otsu's algorithm, applied directly to the original image, successfully preserves finer structures and noise. The binary representation exhibits detailed features.

Smoothed Image:
To reduce noise and enhance smoother representation, the image is smoothed using a 5x5 averaging filter before applying Otsu's thresholding. The structures in the resulting binary image are more consistent and appear smoother.

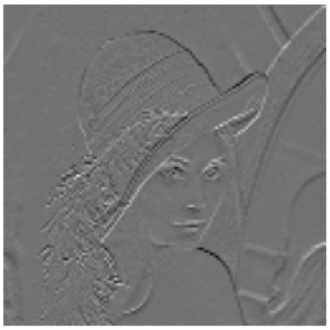
2. Download the image lena.tif from the course webpage, then do the following with the MATLAB function (if you are able to find any Python library for wavelet processing, please
feel free to use it), wavedec2():
a) (5 points) Perform wavelet transform of the Lena image up to and including level 3 by using Haar wavelet, and show your results.

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

# Load the Lena image
lena = cv2.imread('lena.tif', cv2.IMREAD_GRAYSCALE)

def perform_haar_wavelet_transform(image, wavelet_level=3):
    # Haar wavelet transform
    coeffs_haar = pywt.wavedec2(image, 'haar', level=wavelet_level)

    # Reconstruct at level 3 approximation
    reconstructed_haar = pywt.waverec2(coeffs_haar, 'haar')

    # Visualize Haar wavelet transform results
    plt.figure(figsize=(8, 8))
    for i in range(1, len(coeffs_haar)):
        plt.subplot(2, 2, i)
        plt.imshow(coeffs_haar[i][0], cmap='gray')  # Use [0] to access the approximation
coefficient
        plt.title(f'Haar Detail Coefficients - Level {i}')
        plt.axis('off')

    plt.subplot(2, 2, 4)
    plt.imshow(reconstructed_haar, cmap='gray')
    plt.title('Reconstructed Image - Haar Approximation at Level 3')
    plt.axis('off')
    plt.subplots_adjust(wspace=1)
    plt.show()

# Perform Haar wavelet transform on the Lena image
perform_haar_wavelet_transform(lena)
```

Haar Detail Coefficients - Level 1



Haar Detail Coefficients - Level 2



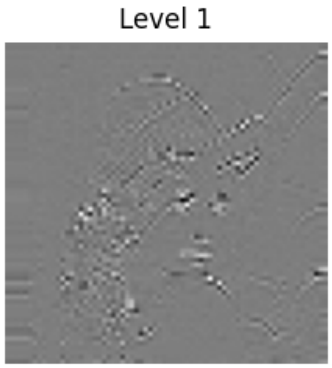
Haar Detail Coefficients - Level 3

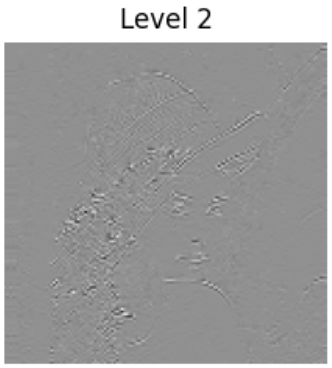
Reconstructed Image - Haar Approximation at Le

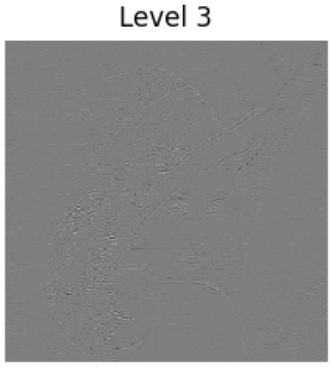
b) (5 points) Perform wavelet transform of the Lena image up to and including level 3 by using Daubechies-4 wavelet (check out the 'wname' flag for available wavelets), and show
your results.

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

# Load the Lena image
```

```
lena = cv2.imread('lena.tif', cv2.IMREAD_GRAYSCALE)


def perform_db4_wavelet_transform(image, wavelet_level=3):
    # Daubechies-4 wavelet transform
    coeffs_db4 = pywt.wavedec2(image, 'db4', level=wavelet_level)

    # Reconstruct at level 3 approximation
    reconstructed_db4 = pywt.waverec2(coeffs_db4, 'db4')

    # Visualize Daubechies-4 wavelet transform results
    plt.figure(figsize=(8, 8))
    for i in range(1, len(coeffs_db4)):
        plt.subplot(2, 2, i)
        plt.imshow(coeffs_db4[i][0], cmap='gray')  # Use [0] to access the approximation coefficient
        plt.title(f'Level {i}')
        plt.axis('off')

    plt.subplot(2, 2, 4)
    plt.imshow(reconstructed_db4, cmap='gray')
    plt.title('Reconstructed Image - Level 3')
    plt.axis('off')
    plt.subplots_adjust(wspace=1)
    plt.show()


# Perform Daubechies-4 wavelet transform on the Lena image
perform_db4_wavelet_transform(lena)
```



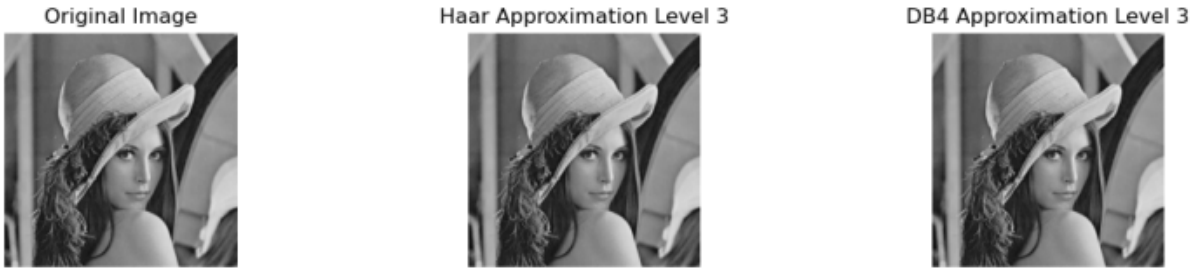




c) (6 points) Visually compare the quality of the approximation images at level 3 of the two
cases and give your comments.

```
# Visual comparison of the approximation images at level 3
plt.figure(figsize=(12, 12))

# Original Image
plt.subplot(1, 3, 1)
plt.imshow(lena, cmap='gray')
```

```
plt.title('Original Image')
plt.axis('off')

# Haar Approximation
plt.subplot(1, 3, 2)
plt.imshow(reconstructed_haar, cmap='gray')
plt.title('Haar Approximation Level 3')
plt.axis('off')

# DB4 Approximation
plt.subplot(1, 3, 3)
plt.imshow(reconstructed_db4, cmap='gray')
plt.title('DB4 Approximation Level 3')
plt.axis('off')

plt.subplots_adjust(wspace=1)
plt.show()
```



Comparison Commentary:

Although the differences may appear subtle at first glance, detailed observations highlight distinctive characteristics between the Haar and DB4 wavelet approximations at Level 3:

Smoothness:

Haar Approximation: The Haar wavelet produced a sharper image, emphasizing finer details.
DB4 Approximation: In contrast, the DB4 wavelet resulted in a smoother image with reduced emphasis on sharp edges.
Contrast:

Haar Wavelet: Preserved contrast more effectively, resulting in a natural appearance for features like eyes, nose, and lips.
DB4 Wavelet: Showed a smoother face appearance, impacting the distinctiveness of facial features.

Color Variation:

Both wavelets maintained color accuracy reasonably well.
DB4: Exhibited subtle color variations, contributing to a more blended appearance.
Haar: Displayed slightly more contrasting color variations, enhancing visual appeal.
Consistency:

Haar Wavelet: Demonstrated less consistency due to its discontinuous nature, featuring sharp jumps between values.
DB4 Wavelet: Offered more consistency with smoother transitions between values, attributed to its continuous wavelet characteristics.
In summary, the choice between Haar and DB4 wavelets depends on the desired visual outcomes. Haar tends to highlight sharp features and maintain contrast, while DB4 provides smoother transitions and a more blended appearance.