**CSCI 8820 Computer Vision and Pattern RecognitionAlex Darwiche HW3**

**3/28/2025**

**Original grayscale images**A computer on a desk

AI-generated content may be incorrect.A room with a shelf and a door

AI-generated content may be incorrect.A white desk with a pile of books and a chair

AI-generated content may be incorrect.**:**

**Question 1: Peakiness Detection**

**Explanation:** To determine the peaks and valley chosen in this algorithm, I used a figure of merit that weighs 4 different things against one another. First, I was looking to maximize the 1) separation between the peaks and 2) the height of the peaks. Next, I wanted to minimize the 3) the height of the valley and 4) the difference between the middle point of the peaks and the index of the valley. Each of these constituted a “term” in my figure of merit, with each getting its own constant to control “how much we care” about that specific term. The terms to maximize were put in the numerator and the terms to minimize were put in the denominator, and I added +1 to the denominators to ensure they were never 0. Below you will see the Python code that shows how I built this figure of merit, using the same important 1, for each term in this instance. Once you determine the 2 peaks and valley that maximize this figure of merit, you can use the valley to threshold between foreground and background.

A computer screen shot of text

AI-generated content may be incorrect.

A black and white image of a graph

AI-generated content may be incorrect.

A graph with red lines

AI-generated content may be incorrect.

**Question 2: Iterative Thresholding**

**Explanation:** For this question, I simply started by finding the mean of the grayscales of the image. I call this T\_init. Values below T\_init are in region 1 and values above T\_init are in region 2. I then find the mean of each region, then t\_new if the average of these 2 means. I continue this process iteratively, until the t\_new stops changing. The code below will show the process of finding these thresholds for each of the 3 input images.

A screen shot of a computer program

AI-generated content may be incorrect.

A graph showing a number of objects

AI-generated content may be incorrect.

**Question 3: Dual Thresholding**

**Explanation:** For this problem, we first need to come up with an “automatic” way to find initial thresholds from the histogram of gray levels. There are 2 ways that I tested to do this. The first, I believe we talked about this in class, is to find the top 2 “modes” of the distribution. If we make the assumption that the gray scale levels are bi-modal, then we can treat these 2 modes as the 100%-foreground and background pixels. Anything below the first peak is in region 1 (foreground). Anything above the second peak is in region 3 (background). The pixels in the middle are then considered region 2 (unsure). I then looped through all the pixels in r1 and grew them outwards if they had 4-neighbors in region 2. If a neighbor of region 1 pixels is in region 2, I added those to the end of region 1 array. This mimics a recursive solution where we continue spreading out as long as there are “unsure” pixels, growing region 1. Once we can no longer grow region 1, we then make all the remaining pixels join region 3 (background). **This approach did not seem to be successful for this** **set of images, likely due to the modes being relatively far to the edges of the grayscale histogram. The image with green below, shows the “unsure” region in one of the images. This illustrates how few pixels can get initially grouped as r1 or r3 in the initial pass. When most pixels are in r2, the algorithm will simply grow into the green space as far as it can, leaving a completely mono-color image (as seen below). The second approach achieves much better results (p-tile method).**A computer screen with text on it

AI-generated content may be incorrect.

**Bad Initial Results Below:**

A graph with a red line

AI-generated content may be incorrect.**A green and black rectangular object

AI-generated content may be incorrect.**

A black and white image of a desk

AI-generated content may be incorrect.

A graph with numbers and lines

AI-generated content may be incorrect.

**(Better Results: Method 2) Explanation:** The second approach you can use to come up with an automatic threshold is to use p-tile or percentiles. Essentially, we can choose percentiles to form our region 1 and region 3 thresholds, then grow from there. In the below example, I used percentiles 40 and 60. These provided the bounds on the initial pixels, and then I grew the region 1 until it could not grow anymore. This appears to do a good job of separating the foreground and backgrounds from each other.

A black and white image of a room

AI-generated content may be incorrect.

A graph with lines and numbers

AI-generated content may be incorrect.A graph with red lines

AI-generated content may be incorrect.A graph with red lines

AI-generated content may be incorrect.

**Individual Images (in case the ones above are difficult to see):**

**Question 1 Images:**

A black and white image of a graph

AI-generated content may be incorrect.A computer on a desk

AI-generated content may be incorrect.A black and white image of a room

AI-generated content may be incorrect.

**Question 2 Images:**

A computer on a desk

AI-generated content may be incorrect.

A black and white image of a graph

AI-generated content may be incorrect.

A black and white image of a room

AI-generated content may be incorrect.

**Question 3 Method 1 Images:**

A black and white image of a room

AI-generated content may be incorrect.

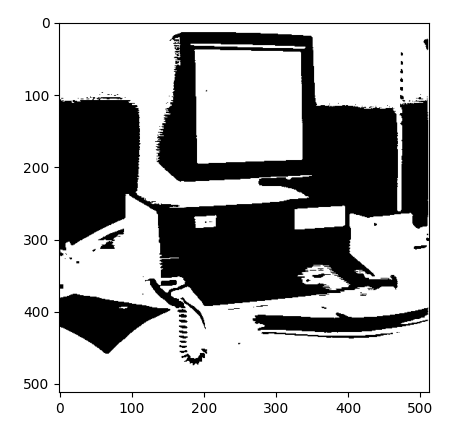
A black and white image of a desk

AI-generated content may be incorrect.

A black and white graph

AI-generated content may be incorrect.

**Question 3 Method 2 Images:**



A black and white image of a graph

AI-generated content may be incorrect.

A black and white image of a door

AI-generated content may be incorrect.

**CODE BELOW:**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import pandas as pd**

**import math**

**def show\_image(img,cmap\_str='gray\_r'):**

**norm = plt.Normalize(vmin=0, vmax=1)  # Normalize so that only positive values are highlighted**

**plt.imshow(img, cmap=cmap\_str,norm=norm)**

**plt.show()**

**def show\_image\_2(img,cmap\_str='gray\_r'):**

**norm = plt.Normalize(vmin=0, vmax=255)  # Normalize so that only positive values are highlighted**

**plt.imshow(img, cmap=cmap\_str,norm=norm)**

**plt.show()**

**def find\_peaks(counts, bins):**

**peaks = []**

**indices = []**

**for i in range(len(counts)):**

**if i == 0:**

**if counts[i] >= counts[i+1]:**

**peaks.append(counts[i])**

**indices.append(bins[i])**

**elif i == len(counts)-1:**

**if counts[i] >= counts[i-1]:**

**peaks.append(counts[i])**

**indices.append(bins[i])**

**elif counts[i] >= counts[i+1] and counts[i] >= counts[i-1]:**

**peaks.append(counts[i])**

**indices.append(bins[i])**

**return peaks, indices**

**def find\_valleys(counts, bins):**

**valleys = []**

**indices = []**

**for i in range(len(counts)):**

**if i == 0:**

**if counts[i] < counts[i+1]:**

**valleys.append(counts[i])**

**indices.append(bins[i])**

**elif i == len(counts)-1:**

**if counts[i] <= counts[i-1]:**

**valleys.append(counts[i])**

**indices.append(bins[i])**

**elif counts[i] <= counts[i+1] and counts[i] <= counts[i-1]:**

**valleys.append(counts[i])**

**indices.append(bins[i])**

**return valleys, indices**

**def find\_best\_combo(peak\_indices, peak\_counts, valley\_indices, valley\_counts):**

**best\_FOM = -10000**

**for index\_i, i in enumerate(peak\_indices):**

**for index\_j, j in enumerate(peak\_indices):**

**# Find peak information**

**peak1\_index, peak2\_index = i, j**

**peak1\_count, peak2\_count = peak\_counts[index\_i], peak\_counts[index\_j]**

**# Confirm that these peaks are correctly located around each other**

**if peak1\_index != peak2\_index and peak1\_index < peak2\_index:**

**for index\_k, k in enumerate(valley\_indices):**

**# Find Valley information**

**valley\_index, valley\_count = k, valley\_counts[index\_k]**

**if valley\_index > peak1\_index and valley\_index < peak2\_index and peak1\_count > valley\_count and peak2\_count > valley\_count:**

**# Compute each term in the FOM equation**

**a, b, c, d = 1,1,1,1**

**term1 = a \* abs(peak1\_index - peak2\_index)  # Peak index distance**

**term2 = b \* (peak1\_count + peak2\_count)    # Sum of peak heights**

**term3 = ((c \* (valley\_count)) + 1)              # valley height**

**term4 = ((d \* abs(((peak1\_index + peak2\_index) / 2) - valley\_index))+1)  # valley distance to middlepoint**

**# Compute final FOM**

**current\_FOM = (term1 \* term2) / (term3 \* term4)**

**if current\_FOM > best\_FOM:**

**best\_FOM = current\_FOM**

**best\_combo = [peak1\_index.copy(), valley\_index.copy(), peak2\_index.copy()]**

**# print(f"Best FOM: {best\_FOM:.10f}, Peak 1 Index: {best\_combo[0]:.2f}, Valley Index: {best\_combo[1]:.2f}, Peak 2 Index: {best\_combo[2]:.2f}")**

**# print(f"Part 1 (a \* term1): {term1:.2f}")**

**# print(f"Part 2 (b \* term2): {term2:.2f}")**

**# print(f"Part 3 (c \* (term3 + 1)): {term3:.2f}")**

**# print(f"Part 4 (d \* (term4 + 1)): {term4:.2f}")**

**# print(f"Valley Count: {valley\_count:.1f}")**

**# print(f"Peak1 Count: {peak1\_count:.1f}")**

**# print(f"Peak2 Count: {peak2\_count:.1f}")**

**# print(f"Current FOM: {current\_FOM:.10f}")**

**return best\_combo**

**#### Run Code ####**

**# Parameter Definitions**

**file\_path1 = "img/test1.img"**

**file\_path2 = "img/test2.img"**

**file\_path3 = "img/test3.img"**

**files = [file\_path1,file\_path2,file\_path3]**

**width, height = 512, 512**

**header\_size = 512**

**image\_data = []**

**combos = []**

**# Read the file(s)**

**for i in range(len(files)):**

**with open(files[i], "rb") as f:**

**f.seek(header\_size)**

**image\_data.append(list(f.read()))**

**# # Reshape into 2D array**

**# import pdb;pdb.set\_trace()**

**# image\_array = [image\_data[i].reshape((height, width)) for i in range(len(files))]**

**# image\_array = [[item for sublist in image\_array[i] for item in sublist] for i in range(len(image\_array))]**

**for i in range(len(image\_data)):**

**image = image\_data[i]**

**bins\_count = 255**

**# Plotting a basic histogram**

**counts, bins, patches = plt.hist(image, bins=bins\_count, color='skyblue', edgecolor='black')**

**# plt.show()**

**peak\_counts, peak\_indices = find\_peaks(counts, bins)**

**valley\_counts, valley\_indices = find\_valleys(counts, bins)**

**combo = find\_best\_combo(peak\_indices,peak\_counts,valley\_indices,valley\_counts)**

**for val in combo:**

**plt.axvline(val, color='red', linestyle='dashed', linewidth=2)**

**plt.savefig('img/img\_'+str(i)+'.png')**

**# plt.show()**

**plt.close()**

**combos.append(combo)**

**# print(len(combos))**

**# for i in combos:**

**#     print(i)**

**import copy**

**image\_data\_binary = copy.deepcopy(image\_data)**

**for i in range(len(image\_data)):**

**for j in range(len(image\_data[i])):**

**if image\_data[i][j] >= combos[i][1]:**

**image\_data\_binary[i][j] = 1**

**else:**

**image\_data\_binary[i][j] = 0**

**# Reshape into 2D array**

**image\_data\_binary = [np.array(image\_data\_binary[i]).reshape((height, width)) for i in range(len(files))]**

**image\_data\_binary = [[item for sublist in image\_data\_binary[i] for item in sublist] for i in range(len(image\_data\_binary))]**

**for i in image\_data\_binary:**

**show\_image(np.array(i).reshape((height,width)))**

**# # Create a subplot with the number of images you have (assuming 3 images for this example)**

**# num\_images = len(image\_data\_binary)**

**# fig, axes = plt.subplots(1, num\_images, figsize=(num\_images \* 3, 3))  # Adjust size as needed**

**# # Ensure axes is iterable if there's only one image**

**# if num\_images == 1:**

**#     axes = [axes]**

**# # Loop over each image and display it**

**# for i, ax in enumerate(axes):**

**#     ax.imshow(np.array(image\_data\_binary[i]).reshape(height,width), cmap='gray\_r')**

**# plt.tight\_layout()  # Adjust layout to make sure images are spaced nicely**

**# plt.show()**

**import matplotlib.pyplot as plt**

**import matplotlib.image as mpimg**

**# # Load images**

**# img1 = mpimg.imread("img/img\_0.png")**

**# img2 = mpimg.imread("img/img\_1.png")**

**# img3 = mpimg.imread("img/img\_2.png")**

**# # Create figure and subplots**

**# fig, axes = plt.subplots(1, 3, figsize=(15, 5))**

**# # Display first image**

**# axes[0].imshow(img1)**

**# axes[0].axis("off")  # Hide axes**

**# axes[0].set\_title("Image 1")**

**# # Display second image**

**# axes[1].imshow(img2)**

**# axes[1].axis("off")**

**# axes[1].set\_title("Image 2")**

**# # Display third image**

**# axes[2].imshow(img3)**

**# axes[2].axis("off")**

**# axes[2].set\_title("Image 3")**

**# # Adjust layout and show**

**# plt.tight\_layout()**

**# plt.show()**

**adaptive\_thresholds = []**

**for i in range(len(image\_data)):**

**t\_init = np.mean(image\_data[i]) # starting threshold**

**t\_new = t\_init # new threshold to be updated each cycle**

**t\_old = -10 # threshold from previous iteration**

**while t\_old != t\_new:**

**t\_old = t\_new**

**m1 = np.mean([value for value in image\_data[i] if value > t\_old]) # mean of region 1**

**m2 = np.mean([value for value in image\_data[i] if value <= t\_old]) # mean of region 2**

**t\_new = (m1+m2)/2 # average of means**

**adaptive\_thresholds.append(t\_new)**

**image\_data\_binary2 = copy.deepcopy(image\_data)**

**for i in range(len(image\_data)):**

**for j in range(len(image\_data[i])):**

**if image\_data[i][j] >= adaptive\_thresholds[i]:**

**image\_data\_binary2[i][j] = 1**

**else:**

**image\_data\_binary2[i][j] = 0**

**# # Reshape into 2D array**

**# image\_data\_binary2 = [np.array(image\_data\_binary2[i]).reshape((height, width)) for i in range(len(files))]**

**# image\_data\_binary2 = [[item for sublist in image\_data\_binary2[i] for item in sublist] for i in range(len(image\_data\_binary2))]**

**for i in image\_data\_binary2:**

**show\_image(np.array(i).reshape((height,width)))**

**# # Create a subplot with the number of images you have (assuming 3 images for this example)**

**# num\_images = len(image\_data\_binary)**

**# fig, axes = plt.subplots(1, num\_images, figsize=(num\_images \* 3, 3))  # Adjust size as needed**

**# # Ensure axes is iterable if there's only one image**

**# if num\_images == 1:**

**#     axes = [axes]**

**# # Loop over each image and display it**

**# for i, ax in enumerate(axes):**

**#     ax.imshow(np.array(image\_data\_binary2[i]).reshape(height,width), cmap='gray\_r')**

**plt.tight\_layout()  # Adjust layout to make sure images are spaced nicely**

**plt.show()**

**def find\_2\_highest\_peaks(peaks\_counts,peak\_indices):**

**total\_values = np.sort(peaks\_counts)[-2:]**

**top\_peaks = [[peak\_indices[ind],i] for ind,i in enumerate(peaks\_counts) if i in total\_values]**

**return top\_peaks**

**dual\_thresholds = []**

**image\_data3 = copy.deepcopy(image\_data)**

**image\_data4 = [np.array(image\_data3[i]).reshape((height, width)) for i in range(len(files))]**

**for i in range(len(image\_data3)):**

**bins\_count = 255**

**# Plotting a basic histogram**

**counts, bins, patches = plt.hist(image\_data3[i], bins=bins\_count, color='skyblue', edgecolor='black')**

**# plt.show()**

**peaks\_counts, peak\_indices = find\_peaks(counts, bins)**

**import os**

**top\_peaks = find\_2\_highest\_peaks(peaks\_counts,peak\_indices)**

**t1 = top\_peaks[0][0]**

**t2 = top\_peaks[1][0]**

**import time**

**vals = [t1,t2]**

**for val in vals:**

**plt.axvline(val, color='red', linestyle='dashed', linewidth=2)**

**plt.show()**

**# Make the program wait for 1 second**

**time.sleep(1)**

**plt.close()**

**img = copy.deepcopy(image\_data4[i])**

**r1 = []**

**r2 = []**

**r3 = []**

**for ind1, row in enumerate(img):**

**for ind2, col in enumerate(row):**

**if img[ind1][ind2] <= t1:**

**img[ind1][ind2] = 0**

**r1.append([ind1,ind2])**

**elif img[ind1][ind2] >= t2:**

**img[ind1][ind2] = 1**

**r3.append([ind1,ind2])**

**else:**

**img[ind1][ind2] = -1**

**r2.append([ind1,ind2])**

**count = 0**

**import matplotlib.colors as mcolors**

**# Define a colormap for the 3 values**

**cmap = mcolors.ListedColormap(["green", "black", "white"])**

**bounds = [-1.5, -0.5, 0.5, 1.5]  # Define the bounds for each value**

**norm = mcolors.BoundaryNorm(bounds, cmap.N)  # Normalize values to colormap**

**# Plot the image**

**plt.imshow(img, cmap=cmap, norm=norm)**

**plt.colorbar(ticks=[-1, 0, 1], label="Pixel Value")  # Add colorbar for reference**

**plt.axis('off')  # Hide axis for better visualization**

**plt.title("Custom Image with Values -1, 0, 1")**

**plt.show()**

**# loop through all points in region 1 (foreground)**

**while count < len(r3):**

**point = r3[count]**

**ind1 = point[0] # y**

**ind2 = point[1] # x**

**count += 1**

**# Left Neighbor**

**if ind2 != 0:**

**if img[ind1][ind2-1] == -1: # Check if left neighbor is in region 2 (unsure)**

**img[ind1][ind2-1] = 1 # Set this neighbor to region 1 (foreground)**

**r3.append([ind1,ind2-1]) # Add this coordinate to region 1's list**

**# Top**

**if ind1 != 0:**

**if img[ind1-1][ind2] == -1:**

**img[ind1-1][ind2] = 1**

**r3.append([ind1-1,ind2])**

**# Right**

**if ind2 != 511:**

**if img[ind1][ind2+1] == -1:**

**img[ind1][ind2+1] = 1**

**r3.append([ind1,ind2+1])**

**# Right**

**if ind1 != 511:**

**if img[ind1+1][ind2] == -1:**

**img[ind1+1][ind2] = 1**

**r3.append([ind1+1,ind2])**

**for ind1, row in enumerate(img):**

**for ind2, col in enumerate(row):**

**if img[ind1][ind2] == -1:**

**img[ind1][ind2] = 0**

**image\_data4[i] = img**

**for i in image\_data4:**

**show\_image(np.array(i))**

**# Create a subplot with the number of images you have (assuming 3 images for this example)**

**num\_images = len(image\_data4)**

**fig, axes = plt.subplots(1, num\_images, figsize=(num\_images \* 3, 3))  # Adjust size as needed**

**# Ensure axes is iterable if there's only one image**

**if num\_images == 1:**

**axes = [axes]**

**# Loop over each image and display it**

**for i, ax in enumerate(axes):**

**ax.imshow(np.array(image\_data4[i]).reshape(height,width), cmap='gray\_r')**

**plt.tight\_layout()  # Adjust layout to make sure images are spaced nicely**

**plt.show()**

**# Create a subplot with the number of images you have (assuming 3 images for this example)**

**num\_images = len(image\_data4)**

**fig, axes = plt.subplots(1, num\_images, figsize=(num\_images \* 3, 3))  # Adjust size as needed**

**# Ensure axes is iterable if there's only one image**

**if num\_images == 1:**

**axes = [axes]**

**# Loop over each image and display it**

**for i, ax in enumerate(axes):**

**ax.hist(image\_data3[i], bins=bins\_count, color='skyblue', edgecolor='black')**

**plt.tight\_layout()  # Adjust layout to make sure images are spaced nicely**

**plt.show()**

**image\_data5 = copy.deepcopy(image\_data)**

**image\_data6 = [np.array(image\_data5[i]).reshape((height, width)) for i in range(len(files))]**

**for i in range(len(image\_data5)):**

**bins\_count = 255**

**# Plotting a basic histogram**

**counts, bins, patches = plt.hist(image\_data5[i], bins=bins\_count, color='skyblue', edgecolor='black')**

**# plt.show()**

**peaks\_counts, peak\_indices = find\_peaks(counts, bins)**

**import os**

**top\_peaks = find\_2\_highest\_peaks(peaks\_counts,peak\_indices)**

**t1 = np.percentile(image\_data5[i], 40)**

**t2 = np.percentile(image\_data5[i], 60)**

**import time**

**vals = [t1,t2]**

**for val in vals:**

**plt.axvline(val, color='red', linestyle='dashed', linewidth=2)**

**plt.show()**

**# Make the program wait for 1 second**

**time.sleep(1)**

**plt.close()**

**img2 = copy.deepcopy(image\_data6[i])**

**r1 = []**

**r2 = []**

**r3 = []**

**for ind1, row in enumerate(img2):**

**for ind2, col in enumerate(row):**

**if img2[ind1][ind2] <= t1:**

**img2[ind1][ind2] = 0**

**r1.append([ind1,ind2])**

**elif img2[ind1][ind2] >= t2:**

**img2[ind1][ind2] = 1**

**r3.append([ind1,ind2])**

**else:**

**img2[ind1][ind2] = -1**

**r2.append([ind1,ind2])**

**count = 0**

**# loop through all points in region 1 (foreground)**

**while count < len(r1):**

**point = r1[count]**

**ind1 = point[0] # y**

**ind2 = point[1] # x**

**count += 1**

**# Left Neighbor**

**if ind2 != 0:**

**if img2[ind1][ind2-1] == -1: # Check if left neighbor is in region 2 (unsure)**

**img2[ind1][ind2-1] = 0 # Set this neighbor to region 1 (foreground)**

**r1.append([ind1,ind2-1]) # Add this coordinate to region 1's list**

**# Top**

**if ind1 != 0:**

**if img2[ind1-1][ind2] == -1:**

**img2[ind1-1][ind2] = 0**

**r1.append([ind1-1,ind2])**

**# Right**

**if ind2 != 511:**

**if img2[ind1][ind2+1] == -1:**

**img2[ind1][ind2+1] = 0**

**r1.append([ind1,ind2+1])**

**# Right**

**if ind1 != 511:**

**if img2[ind1+1][ind2] == -1:**

**img2[ind1+1][ind2] = 0**

**r1.append([ind1+1,ind2])**

**for ind1, row in enumerate(img2):**

**for ind2, col in enumerate(row):**

**if img2[ind1][ind2] == -1:**

**img2[ind1][ind2] = 1**

**image\_data6[i] = img2**

**for i in image\_data6:**

**show\_image(np.array(i))**

**# Create a subplot with the number of images you have (assuming 3 images for this example)**

**num\_images = len(image\_data6)**

**fig, axes = plt.subplots(1, num\_images, figsize=(num\_images \* 3, 3))  # Adjust size as needed**

**# Ensure axes is iterable if there's only one image**

**if num\_images == 1:**

**axes = [axes]**

**# Loop over each image and display it**

**for i, ax in enumerate(axes):**

**ax.imshow(np.array(image\_data6[i]).reshape(height,width), cmap='gray\_r')**

**plt.tight\_layout()  # Adjust layout to make sure images are spaced nicely**

**plt.show()**