ECE661 Fall 2024: Homework 6

By – Aayush Bhadani

abhadani@purdue.edu

2. Theory Question:

Otsu's algorithm is an effective and simple technique used for image segmentation. It automatically identifies the optimal threshold to separate foreground and background by analyzing the image histogram. While it performs well with bimodal histograms that exhibit clear intensity peaks, it faces challenges with multimodal histograms containing multiple peaks. Additionally, the algorithm tends to struggle under conditions of heavy noise, with small/multiple objects in the scene, and in uneven lighting situations.

The watershed method is able to handle complex shapes and performs well for segmenting objects with intricate boundaries. It uses spatial information which makes it robust against noise, unlike Otsu's algorithm. The watershed method can however result in over-segmentation, thereby producing many small regions. In such complex scenes, it can be adapted with markers for more controlled segmentation. This method is also sensitive to the choice of parameters and can be slow due to heavy computation, particularly when dealing with large images that require markers.

3. Implementation

Otsu's Algorithm:

It finds the threshold to separate foreground and background. It does so by minimizing in-class variance/maximizing between-class variance. This between-class variance is given as:

$$\sigma_B^2 = P_1 P_2 (m_1 - m_2)^2$$

For a set $\{0,1,2, \dots, L-1\}$ of L distinct integer intensity levels in a digital image with MxN pixels,

The normalized histogram components $p_i = \frac{n_i}{MN}$ which follows $\sum_{i=0}^{L-1} p_i = 1 \ with \ p_i \geq 0$.

With threshold $\mathrm{T(k)} = \mathrm{k}, \ 0 < \mathrm{k} < \mathrm{L} - 1$ we can threshold the input image into two classes:

c_1 with pixel intensity range $[0, k]$ and c_2 with range $[k+1, L-1]$.

The probability that a pixel is assigned to the class c_1 is $P_1(k) = \sum_{i=0}^{k} p_i$

and that of class c_2 is $P_2(k) = \sum_{i=k+1}^{L-1} p_i = 1 - P_1(k)$

The mean intensity values of pixels in c_1 is $m_1(k) = \frac{\sum_{i=0}^{k} i p_i}{\sum_{i=0}^{k} p_i}$ and c_2 is $m_2(k) = \frac{\sum_{i=k+1}^{L-1} i p_i}{\sum_{i=k+1}^{L-1} p_i}$

The average intensity of the entire image is given as $m_G = \sum_{i=0}^{L-1} i p_i$

With this, we can observe that $P_1 m_1 + P_2 m_2 = m_G \ and \ P_1 + P_2 = 1$

The between-class variance is given by $\sigma_B^2 = P_1(m_1 - m_G)^2 + P_2(m_2 - m_G)^2$ which can be reduced to

$$\sigma_B^2 = P_1 P_2 (m_1 - m_2)^2$$

Otsu's algorithm is to maximize σ_B^2 .

Image Segmentation with Otsu using RGB values:

- Split the input image into different channels: B, G, R.
- For each channel find the segmentation mask using the threshold from Otsu's algorithm.
- Combine the results from all channels using logical 'AND' operation.

Image Segmentation with Otsu using texture features:

- Start by converting the colored image to grayscale.
- Extract texture-based features using a sliding window approach where we place an NxN window at each pixel and compute intensity variance within. The generated feature map is then passed to Otsu's algorithm to obtain the segmentation mask. The image is padded with 0 at N//2 distance before we start processing.
- The abovementioned step is repeated with different window sizes (N). 3 different sizes were used here.
- Finally, the masks obtained from different window sizes are combined using logical 'AND' operation.

Contour Extraction:

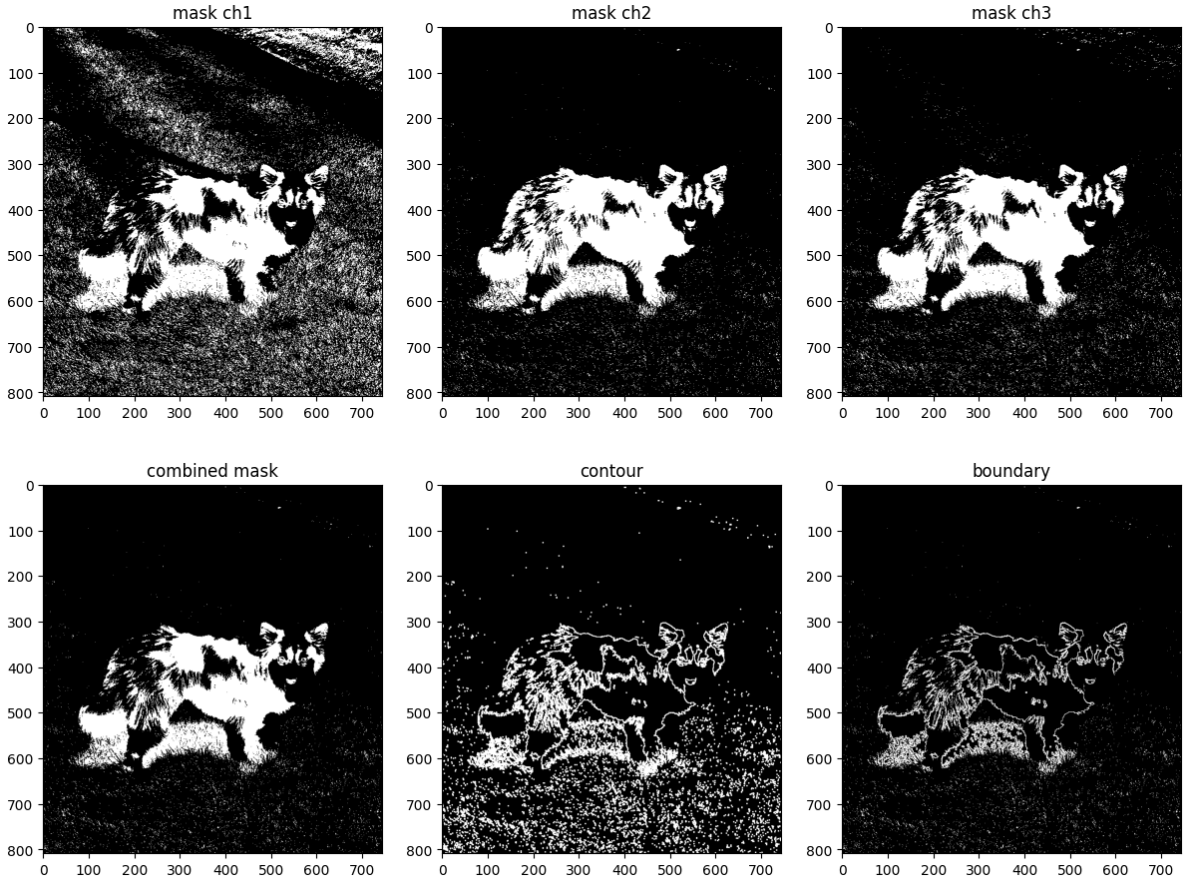
- Using a 3x3 sliding window on the combined mask, a pixel with value 1 is identified as being on the contour if at least one of its eight neighbours has a value 0. If so, that pixel is copied to the contour mask. (labelled as contour mask)
- In another approach, the boundary is extracted by taking the difference between the input image and the eroded image. (labelled as boundary mask)

3.1 TASK 1

Image 1:

Image Segmentation using RGB values & Contour extraction:

(channel order: BGR)

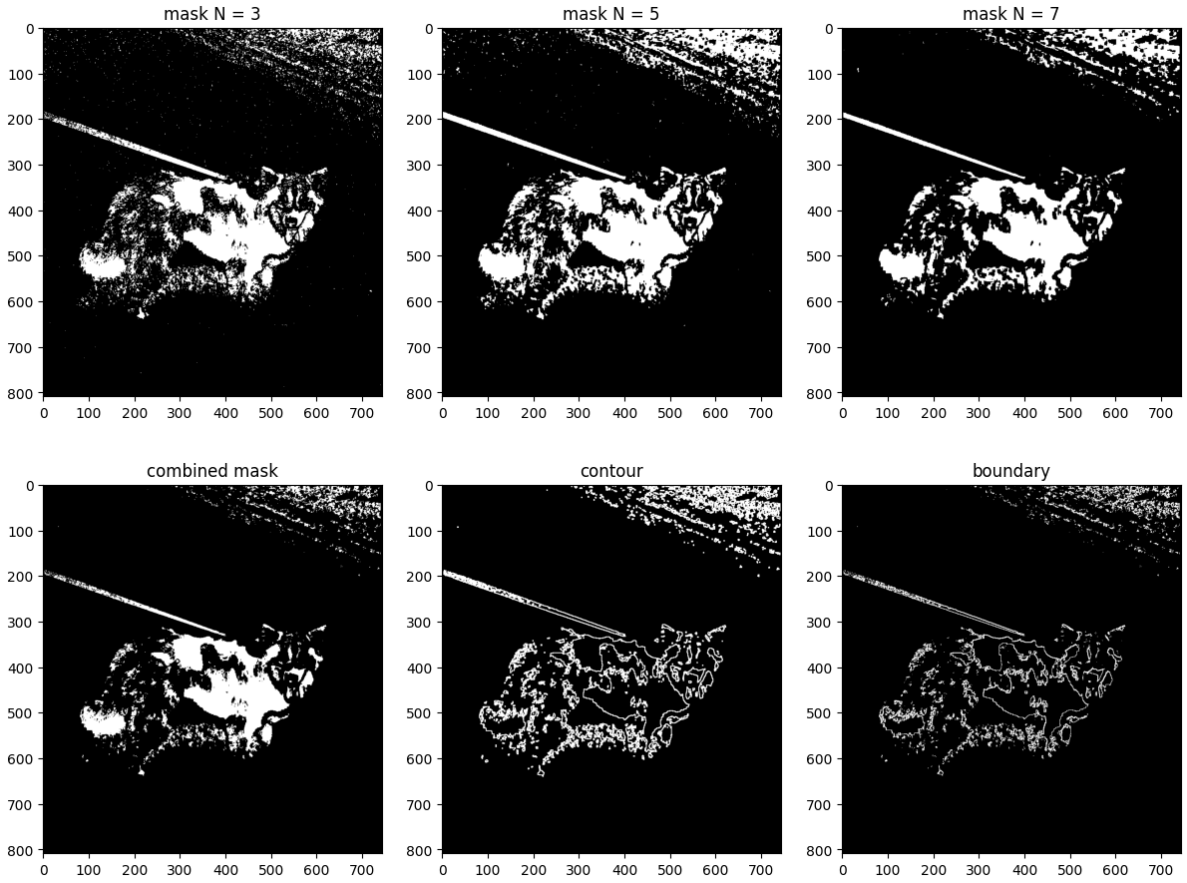


Parameters: (channel-wise with order BGR)

iter:[2,2,2]

inv: [1,1,1]

Texture-based segmentation & Contour extraction:



Parameters: (as per window size)

Window: [3,5,7]

iter: [5,4,4]

inv: [1,1,1]

Image 2:

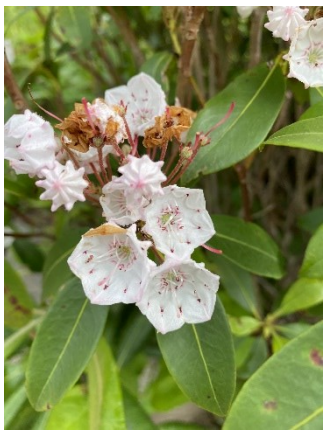
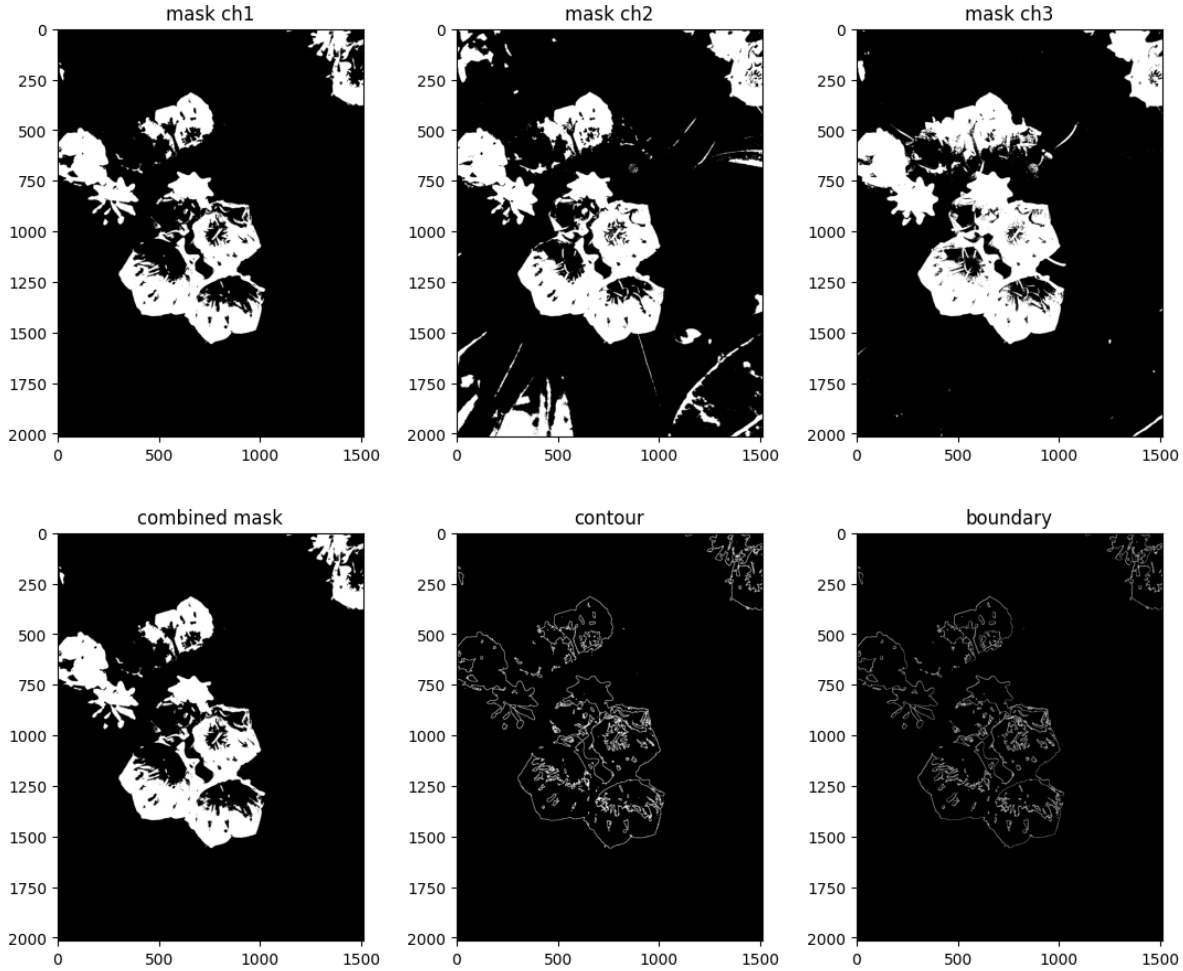


Image Segmentation using RGB values & Contour extraction:

(channel order: BGR)

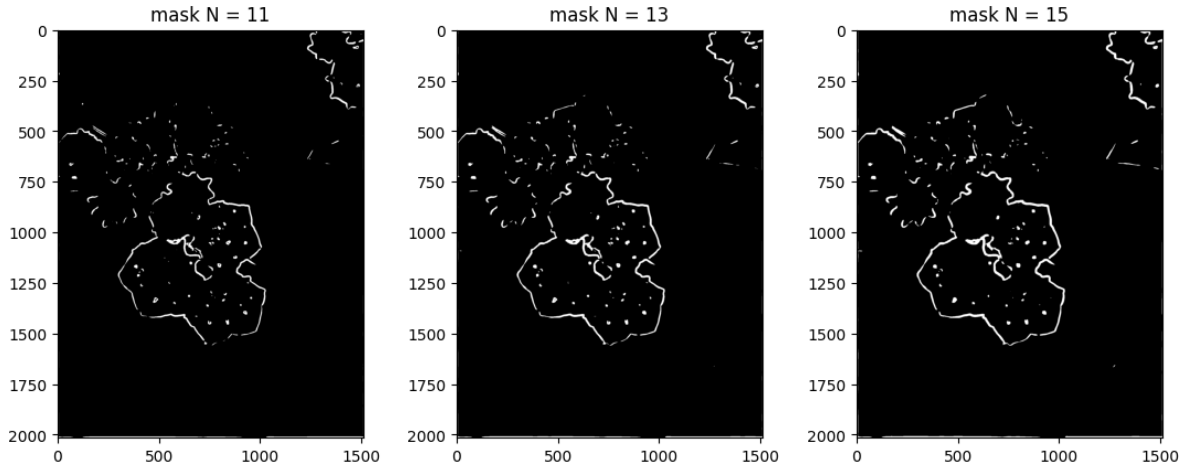


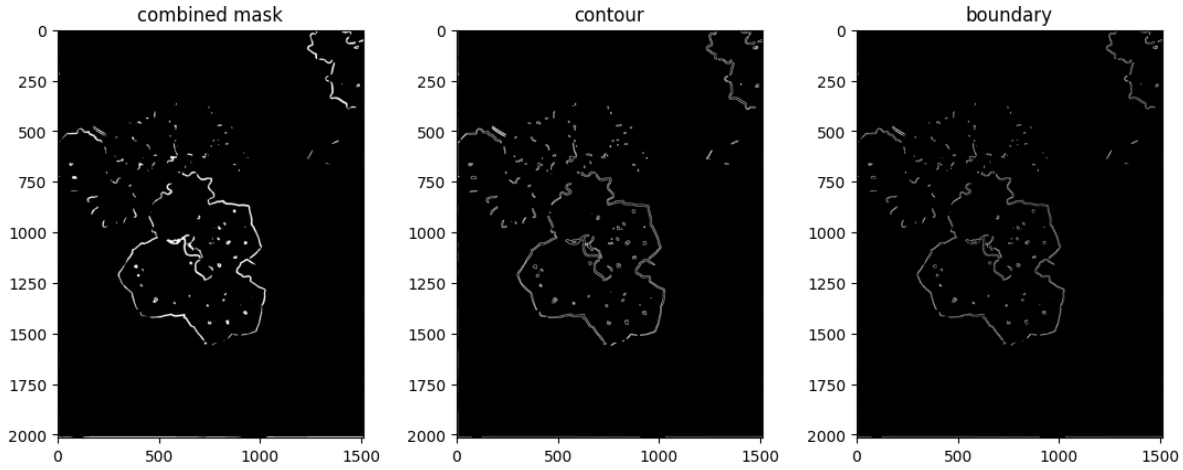
Parameters: (channel-wise with order BGR)

iter:[2,2,2]

inv: [0,0,0]

Texture-based segmentation & Contour extraction:





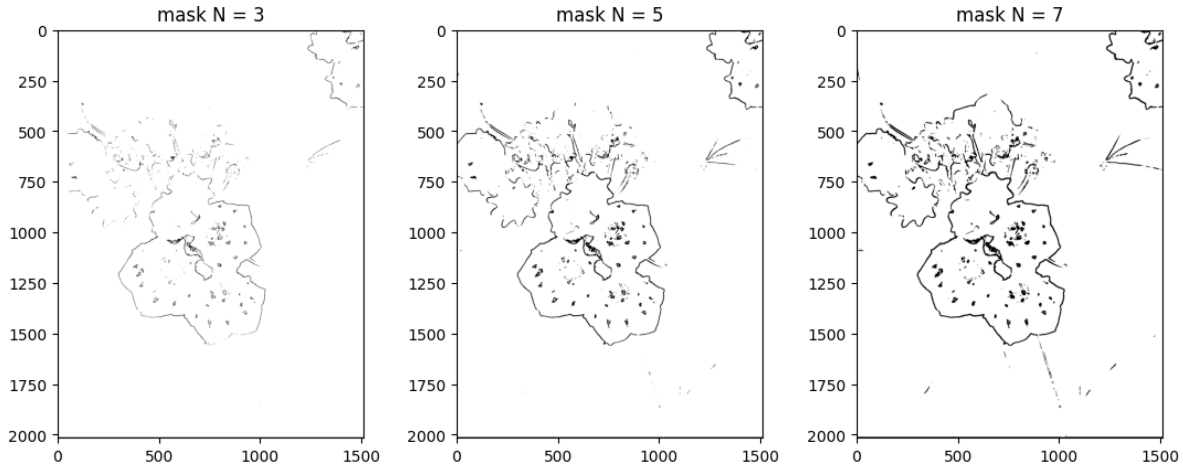
Parameters: (as per window size)

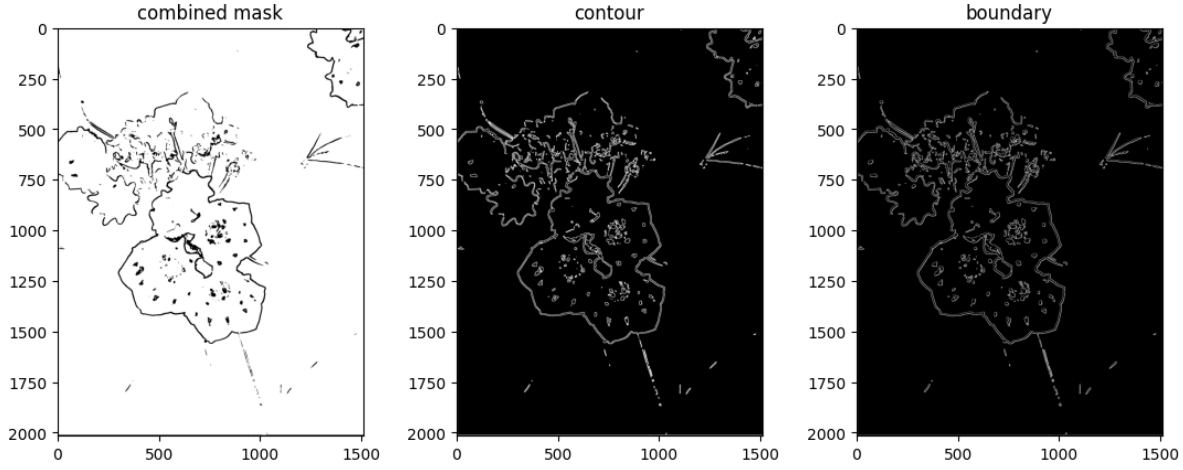
Window: [11,13,15]

iter: [1,1,1]

inv: [0,0,0]

<u>Texture-based segmentation & Contour extraction:</u> (with different set of parameters)





Parameters: (as per window size)

Window: [3,5,7]

iter: [2,2,2]

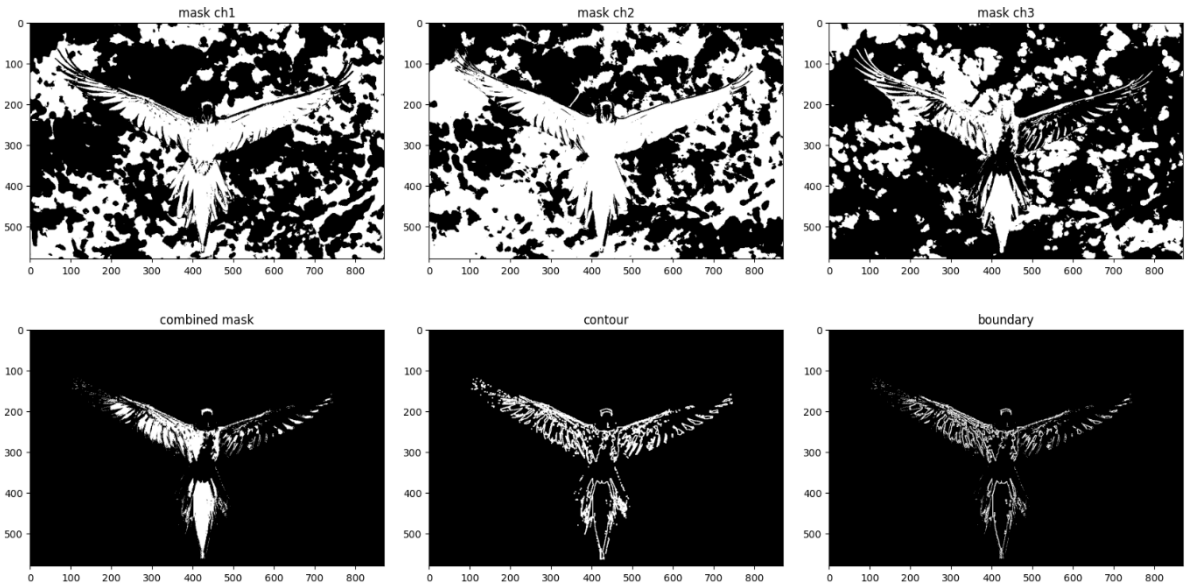
inv: [1,1,1]

3.3 TASK 2

Image 3: Bird with a tree in the background

Image Segmentation using RGB values & Contour extraction:

(channel order: BGR)

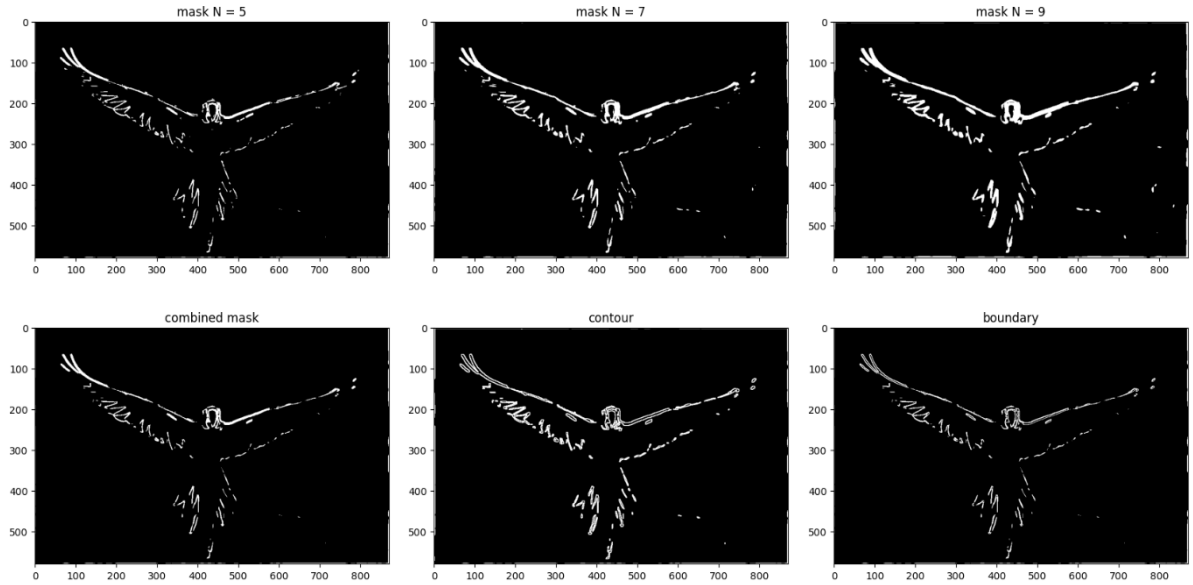


Parameters: (channel-wise with order BGR)

iter:[2,1,1]

inv: [1,1,0]

Texture-based segmentation & Contour extraction:



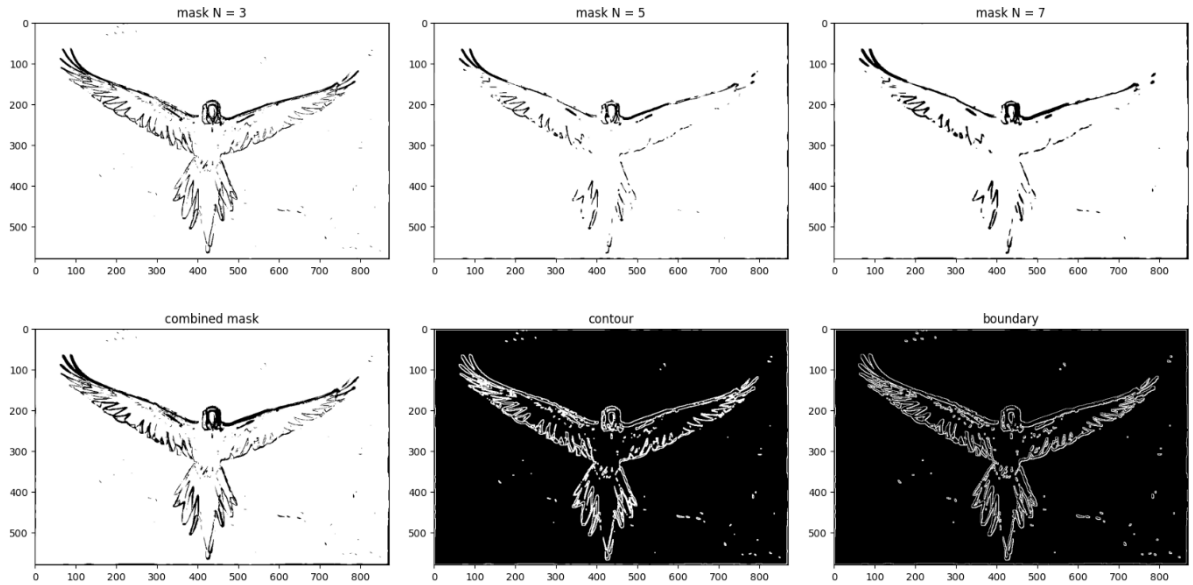
Parameters: (as per window size)

Window: [5,7,9]

iter: [1,1,1]

inv: [0,0,0]

Texture-based segmentation & Contour extraction: (with different set of parameters)



Parameters: (as per window size)

Window: [3,5,7]

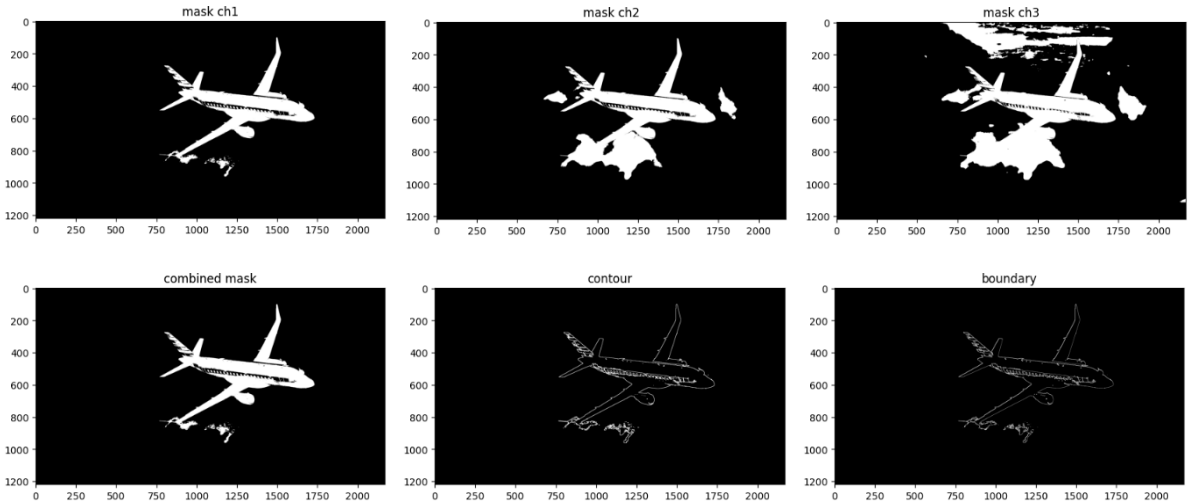
iter: [2,1,1]

inv: [1,1,1]

Image 4: plane with clouds in the background

Image Segmentation using RGB values & Contour extraction:

(channel order: BGR)

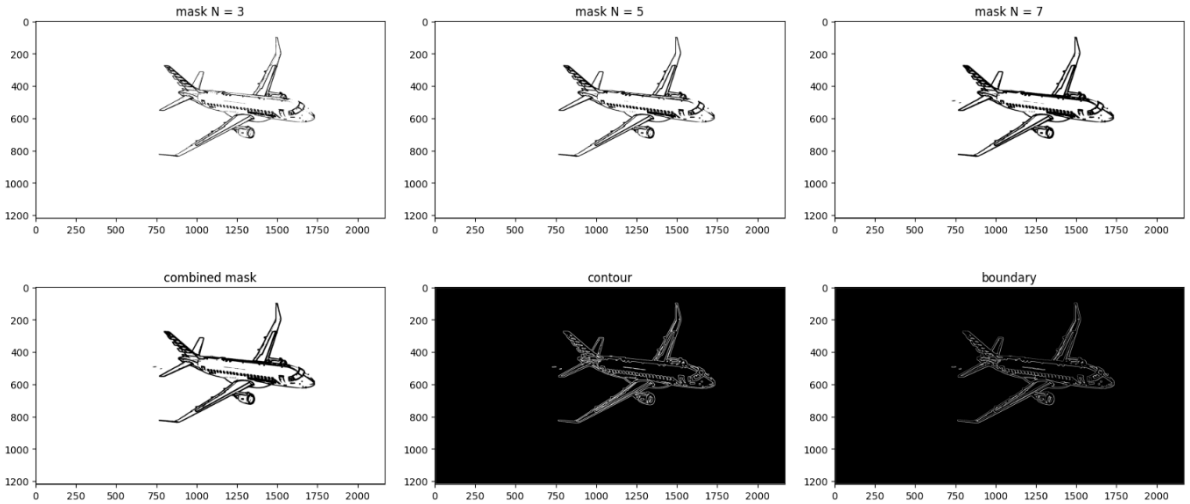


Parameters: (channel-wise with order BGR)

iter:[1,1,1]

inv: [1,1,1]

Texture-based segmentation & Contour extraction:



Parameters: (as per window size)

Window: [3,5,7]

iter: [4,4,4]

inv: [1,1,1]

Observations:

RGB-based and texture-based image segmentation approaches were explored. The performance of these methods is based on the characteristics of the image, the objectives of the segmentation task, and the choice of parameters. While RGB-based approach works well in images with distinct colors in well-defined regions, it struggles in complex scenes having overlapping objects and similar colors. On the other hand texture-based approach tends to be better where textures are the main differentiator, suggesting that it can be helpful with objects that have similar colors but different textures. As this method focuses on structural patterns rather than color, it is robust to lighting conditions as well.

- In image1 where we have a picture of a dog, RGB based approach works very well in segmenting out the dog from its green grass background resulting in a good smooth outline. It even took care of its blue-colored collar strap, although the shadow remains noticeable. The texture-based approach also performed well, capturing fine details of the dog's face in the output.
- For image2, both RGB-based and texture-based approaches were able to separate flowers from green leaves. The RGB-based approach worked as there's a color contrast between flowers and leaves. The texture-based approach was able to capture finer bud structures as well.
- In image3, the RGB-based method is able to extract most of the red-colored bird outline from its green background. However, the beak and part of its belly are missing. This exclusion is likely due to the similar colors of these areas and their surroundings. The texture-based approach is able to capture these missing parts as well, but we end up with more noisy points in the background.
- In image4, the colors of the plane and the background clouds are quite similar. This creates some challenge for the RGB-based approach, which ends up capturing both the plane and a small portion of the cloud background. In contrast, the texture-based approach provides a distinct outline of the plane, isolating it effectively.

CODE

```
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv
from skimage import io, draw
from scipy import optimize


# Otsu's algo -> get threshold
def Otsu_thres(img):

    hist, bin_edg = np.histogram(img,bins=256,range=(0,256))
    hist = hist/sum(hist)

    max_sig = 0
    threshold_opt = 0
    for t in range(256):
        P1, P2 = np.sum(hist[:t]), np.sum(hist[t:])
        if P1 and P2:
            m1, m2 = np.sum(bin_edg[:t]*hist[:t])/P1, np.sum(bin_edg[t:-1]*hist[t:])/P2
            sig = P1*P2*((m1-m2)**2)
            if sig>max_sig:
                max_sig = sig
                threshold_opt = t

    return threshold_opt



#image segmentation using RGB. I/P param: img-image ; iter - #iterations (list of length 3); inv - boolean list len 3 for inverting
def ImgSegment_RGB(img,iter,inv):
    mask_list = []
    for c in range(img.shape[2]):
        img_data = img[:,:,c]
        #img_data = cv.GaussianBlur(img_data, (5, 5), 0)
        for i in range(iter[c]):
            thres = Otsu_thres(img_data)
            if inv[c]:
                mask = img[:,:,c] < thres
            else:
                mask = img[:,:,c] >= thres
            img_data = img[:,:,c][mask]
        mask_list.append(mask)
    comb_mask = np.logical_and.reduce(mask_list)

    plt.imshow(mask_list[0]*255,cmap='gray')
    plt.title('mask ch1')
    plt.show()
    plt.imshow(mask_list[1]*255,cmap='gray')
    plt.title('mask ch2')
    plt.show()
    plt.imshow(mask_list[2]*255,cmap='gray')
    plt.title('mask ch3')
    plt.show()
    plt.imshow(comb_mask*255,cmap='gray')
    plt.title('combined mask')
    plt.show()

    # cv.imwrite('test1.jpeg',comb_mask*255)

    return comb_mask



#extract textual feature. Param N is window size
def textual_feature(gray_img,N):
    var_map = np.zeros(gray_img.shape,dtype=float)
    padded_img = np.pad(gray_img,N//2,constant_values=0)
    for i in range(gray_img.shape[0]):
        for j in range(gray_img.shape[1]):
            window = padded_img[i:i+N,j:j+N]
            window_var = np.var(window)
            var_map[i,j] = window_var
    var_map = (((var_map - np.min(var_map)) / (np.max(var_map)-np.min(var_map)))*255).astype(np.uint8)
    return var_map
```

```
#Texture-based segmentation. Param window_size is list of window sizes (i.e N) ; iter-#iterations ; inv-invert
def ImgSegment_features(img,window_size,iter,inv):
    gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    feature_maps = [textual_feature(gray_img,N) for N in window_size]
    mask_list=[]
    for i in range(len(feature_maps)):
        img_data = feature_maps[i]
        for j in range(iter[i]):
            thres = Otsu_thres(img_data)
            if inv[i]:
                mask = feature_maps[i] < thres
            else:
                mask = feature_maps[i] >= thres
            img_data = feature_maps[i][mask]
        mask_list.append(mask)
    comb_mask = np.logical_and.reduce(mask_list)

    plt.imshow(mask_list[0]*255,cmap='gray')
    plt.title(f'mask N = {window_size[0]}')
    plt.show()
    plt.imshow(mask_list[1]*255,cmap='gray')
    plt.title(f'mask N = {window_size[1]}')
    plt.show()
    plt.imshow(mask_list[2]*255,cmap='gray')
    plt.title(f'mask N = {window_size[2]}')
    plt.show()
    plt.imshow(comb_mask*255,cmap='gray')
    plt.title('combined mask')
    plt.show()

    # cv.imwrite('test1.jpeg',comb_mask*255)

    return comb_mask



#extract contour based on neighbours
def extract_contour(img_mask):
    h,w = img_mask.shape
    contour_img = np.zeros((h,w),dtype=np.uint8)
    padded_img_mask = np.pad(img_mask,1,constant_values=0)
    for i in range(h):
        for j in range(w):
            window = padded_img_mask[i-1:i+2,j-1:j+2]
            if 0<np.sum(window)<9:
                contour_img[i,j]=1

    plt.imshow(contour_img*255,cmap='gray')
    plt.title('contour')
    plt.show()


#contour using diff. of image and eroded image
def extract_boundary(img_mask):
    k = 4
    kernel = np.ones((k,k),dtype=np.uint8)
    eroded_img_mask = cv.erode(np.float32(img_mask),kernel,iterations=1)
    contour_img = img_mask - eroded_img_mask

    plt.imshow(contour_img*255,cmap='gray')
    plt.title('boundary')
    plt.show()


def ImgSegment_RGB_wContours(img,iter,inv):
  comb_mask = ImgSegment_RGB(img,iter,inv)
  extract_contour(comb_mask)
  extract_boundary(comb_mask)



def ImgSegment_features_wContours(img,window,iter,inv):
  comb_mask = ImgSegment_features(img,window,iter,inv)
  extract_contour(comb_mask)
  extract_boundary(comb_mask)




image1 = cv.imread('/content/dog_small.jpg')
image2 = cv.imread('/content/flower_small.jpg')
image3 = cv.imread('/content/bird1.jpg')
```

```
image4 = cv.imread('/content/plane.jpg')
image5 = cv.imread('/content/cat.jpg')


ImgSegment_RGB_wContours(image1,[2,2,2],[1,1,1])
ImgSegment_features_wContours(image1,[3,5,7],[5,4,4],[1,1,1])
ImgSegment_RGB_wContours(image2,[2,2,2],[0,0,0])
ImgSegment_features_wContours(image2,[3,5,7],[2,2,2],[1,1,1])
#ImgSegment_features_wContours(image2,[11,13,15],[1,1,1],[0,0,0])
ImgSegment_RGB_wContours(image3,[2,1,1],[1,1,0])
ImgSegment_features_wContours(image3,[3,5,7],[2,1,1],[1,1,1])
#ImgSegment_features_wContours(image3,[5,7,9],[1,1,1],[0,0,0])
ImgSegment_RGB_wContours(image4,[1,1,1],[1,1,1])
ImgSegment_features_wContours(image4,[3,5,7],[4,4,4],[1,1,1])
```