Fundamentals Of Image Processing
Take Home Exam 3

1st Can Karagedik
Middle East Technical University
Ankara, Turkey
can.karagedik@metu.edu.tr

***Abstract*—This document is a report that explains the methodology and includes the analysis of the results about the assignment which we implementing fundamental morphological image processing techniques for object counting, Mean Shift and Normalized-Cut image segmentation algorithms for separating water, land and sky in the given images.**

I. Introduction

This Document has two main sections. In the first section we implemented mathematical morphology operations for object counting using mainly trial-error methodology. Second part we used Mean Shift and Normalized-Cut algorithms to segment the given images.

II. Object Counting

The counting problem is the estimation of the number of objects in a still image or video frame. To this end we have mainly two options, counting by detection and counting by regression. Counting by detection assumes the use of a visual object detector, that localizes individual object instances in the image. Given the locations of all instances, counting becomes trivial except some cases especially for overlapping instances. In the other hand, counting by regression methods avoids solving the detection problem. Instead, a direct mapping from some global image characteristics (mainly histograms) to the number of objects is used. In our implementation we will be doing counting by regression with basic tools.

Our first goal is to separate the foreground(where the objects are) and the background. The first way that comes to mind is the color segmentation. However, there are problems that we may encounter depending on the images. In most cases we need to resolve the problem of uneven illumination and appereance of variaty of colors in the background(we want sky or water to be same color or tones of it, but for example in sunset sky has multiple colors). To avoid this, we converted our RGB images to grayscale images and then we applied adaptive thresholding with the Otsu algorithm. Yet, the Otsu algorithm failed to separate the background in one piece. The reason for this may be that the grayscale histogram does not fit exactly into the gaussian mixtures. Therefore, we manually examined the grayscale histograms of the images, and determined thresholds by trial and error. Hence we got the binarized images but this binary image comes with a lot of artificial objects with a little squares (around 1-10 pixels). So we removed them using operations of mathematical morphology.

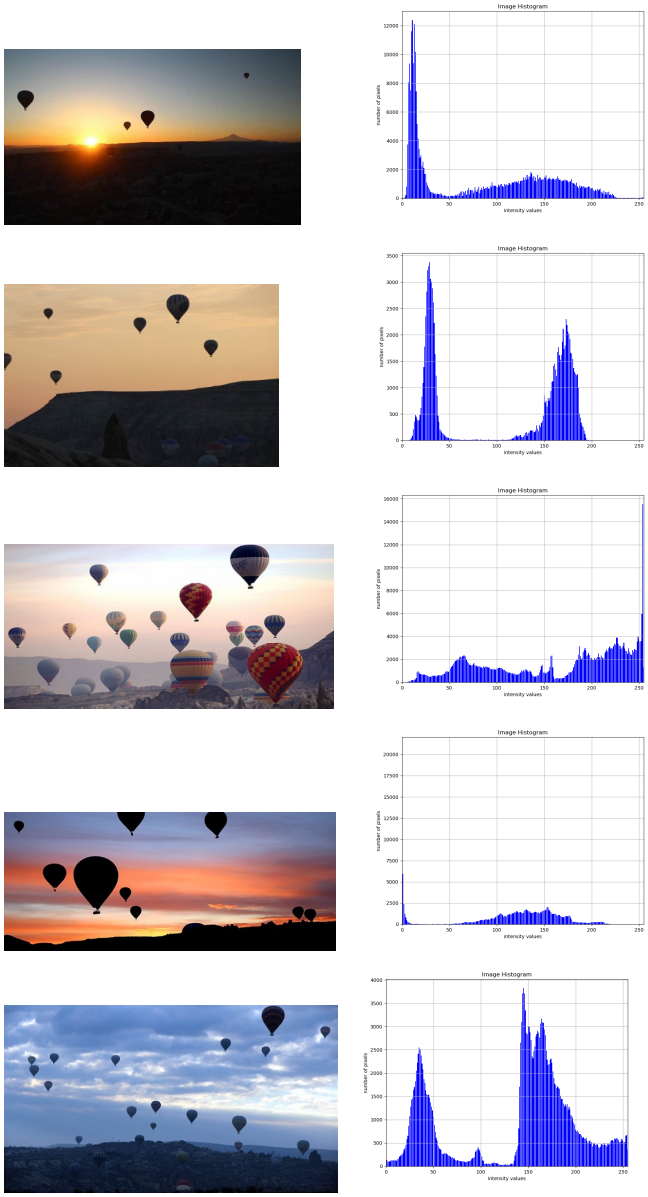

Fig. 1. Given Images and the Grayscale Histograms of the Images

According to these histograms we pick the suitable threshold values and binarized each images so that while the background pixels having 0 values, object pixels got value 1.

```
thresholds = [50,100,160,50,80]

for i in range (1,6):
    # Read the input images in order
    img = cv2.imread("./partA/A{}.png".format(i), 1)
        #0(cv2.IMREAD_GRAYSCALE) stands for
        GRAYSCALE
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    number_of_rows = img.shape[0]
    number_of_columns = img.shape[1]
    ret,binary = cv2.threshold(gray,thresholds[i
        -1],255,cv2.THRESH_BINARY_INV)
```

To see the importance of the threshold values, here the binarized and post processed images with threshold values 156 and 160 with very small gap:

Fig. 2. On the Left Thresholded Images with T_1 = 156, T_2 = 160, On the Right Morphology Operated Images

And contribution of thresholding to the final result:

Fig. 3. Number of Balloons with T_1 = 156, T_2 = 160

The artifacts that appear in the above image can be destroyed by dilation operation as we will discuss later. However, by doing this, some of the balloons we have already detected may combine and cause us to count those balloons as a single balloon. Thus the threshold values are crucial.

Afterwards we applied morphological operations to get rid of the unwanted artifacts. First we decided our structuring elements suited to corresponding images. All of them being matrices with full of once, their sizes are differed. We decided on the size of the kernels to be the minimum size to cover the artifacts. Also, in some cases, the kernel would cause two balloons to merge or separate. We tried to make the optimal choice by taking this into account.

In mathematical morphology, a structuring element is defined as a point set that is translated across the image using logical operations to restructure the object in the image. The structuring element used to restructure the input picture is an important aspect of the morphological dilation and erosion procedures. A structuring element is a matrix that specifies the neighborhood utilized in the processing of each pixel in the picture being processed. One may pick a structural element in any shape and size corresponding to one's needs.

After defining our structuring elements, we applied dilation and erosion in the required order.

Mathematically, dilation and erosion of binary image A by structuring element B is defined respectively as:

$$A \oplus B = \{z|[B_z \cap A] \subseteq A\} = \bigcup_{a \in A} B_a \tag{1}$$

$$A \ominus B = \{z|B_z \subseteq A \quad \} = \{a|a + b \in A, \forall b \in B\} \tag{2}$$

,where B_z is the structure element originated at z.

Fig. 4. Morphological Operations

In words:

Dilation of a set A by a structuring element B is the set of all displacements z such that the intersection of structuring element, B_z and A is not an empty set, that is they overlap by at least one element which resides in A.

The erosion of an image A by a structuring element B is the set of all points z of A such that the structuring element B_z is fully contained in A.

In our implementation, instead of using logical operations, we used min/max operations that bring us to the same conclusion. In dilation, while traversing the image pixels, if any entry of the structuring element(kernel) is encountered with an image pixel with value 1 then the whole image block at the location of the kernel must become 1. Equally, assuming the same scenario, when we multiply kernel with the image block at the location of the kernel and we take the maximum value of this multiplication, it will return 1. So we assign maximum value to every entry of the image block at the location of the

kernel. The erosion operation is similar but this time we assign the minimum value.

Here dilation and erosion operations in our implementation:

```
def morphology(image, kernel, operation = "erote"):
    k_x, k_y = kernel.shape
    number_of_rows = image.shape[0]
    number_of_columns = image.shape[1]
    result = np.zeros((number_of_rows,
        number_of_columns),np.uint8 )
    for x in range ((k_x-1)//2,number_of_rows-(k_x
        -1)//2):
        for y in range ((k_y-1)//2,number_of_columns
            -(k_y-1)//2):
            img_block = image[x-(k_x-1)//2: x+1+(k_x
                -1)//2  ,y-(k_y-1)//2: y+1+(k_y-1)
                //2]
            temp = img_block*kernel
            if(operation == "erote"):
                result[x][y] =  np.min(temp)
            elif (operation == "dilate"):
                result[x][y] = np.max(temp)
    return result
```

In all images except the third we applied opening that is erosion followed by dilation. In the third image, eroting first resulted with bigger unwanted artifacts so we inverted the order of operations hence we applied closing operation. After all these operations, we resulted with binary image where the balloons and terrain have value 1 and the sky has 0. Thus, to find the number of the balloons, we found the number contours in the image by using OpenCV function cv2.findContours() and subtracted 1 to not count the terrain.

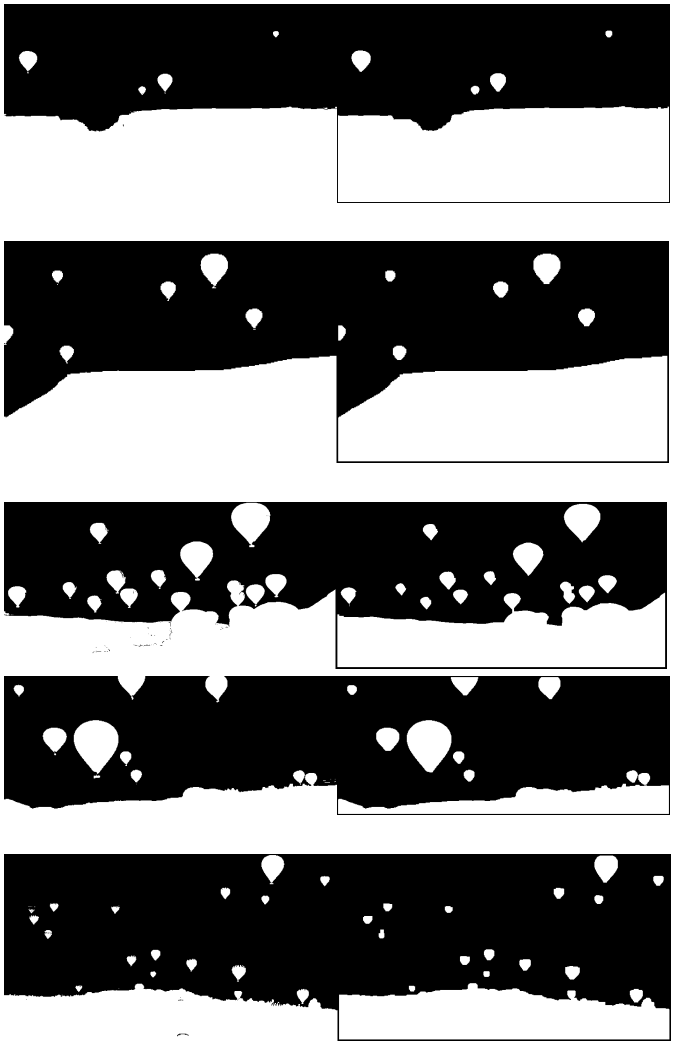

Fig. 5. Thresholded and Opened/Closed Images

```
thresholds = [50,100,160,50,80]
structuring_elements_erotion = [(3,3),(5, 5),(9, 9),(5, 5),(5, 5)]
structuring_elements_dilation = [(5, 5),(5, 5),(3, 3),(5, 5),(7, 7)]
for i in range (1,6):
    # Read the input images in order
    img = cv2.imread("./partA/A{}.png".format(i), 1)
    gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    number_of_rows = img.shape[0]
    number_of_columns = img.shape[1]
    ret,binary = cv2.threshold(gray,thresholds[i-1],255,cv2.THRESH_BINARY_INV)
    if (i == 3): #for the third image we apply opening instead of closing
        structuring_element = np.ones(structuring_elements_dilation[i-1], np.uint8
            )
        dilated = morphology(binary,structuring_element,"dilate")
        structuring_element = np.ones(structuring_elements_erotion[2], np.uint8 )
        eroted = morphology(dilated,structuring_element,"erote")
        detected = eroted
    else:
        structuring_element = np.ones(structuring_elements_erotion[i-1], np.uint8 )
        eroted = morphology(binary,structuring_element,"erote")
        structuring_element = np.ones(structuring_elements_dilation[i-1], np.uint8
            )
        dilated = morphology(eroted,structuring_element,"dilate")
        detected = dilated
    contours, hierarchy = cv2.findContours(detected, cv2.RETR_TREE, cv2.
        CHAIN_APPROX_SIMPLE)
    print("The number of flying balloons in image A{} is {}".format(i,len(contours)
        -1))
    cv2.imwrite("part1_A{}.png".format(i), detected)
```

As can be seen in the pictures, we could not manage to count the overlapping balloons separately, and we counted them as one. To detect overlapping balloons one may try to obtain the skeletons of the objects which defined as the set of connected pixels, which are equidistant to object boundaries. Another method may work for detection of overlapping objects may be the concave point extraction method. Unfortunately we were unable to implement these methods due to lack of time. Also we could not count the balloons with a background, where color tones of the balloons are close to the colors of the terrain. We have already lost those balloons at the thresholding stage and we could not detect them.

Results:

Fig. 6. Number of Balloons

III. Segmentation

In this section we will implement unsupervised segmentation techniques. These groups of algorithms are context independent. In other words, the segmentation algorithm blindly groups the pixels without considering the meaningful labels for the pixel groups. The output of these group of algorithms are called partial segmentation, which does not always correspond to meaningful regions.

Fig. 7. Given Images for Segmentation

A. *Mean-Shift*

The Mean-Shift segmentation algorithm is a straightforward extension of the mean shift clustering algorithm, which we have pixel intensity values as the data.

Mean-shift clustering is an unsupervised, nearly a non-parametric clustering algorithm that is mostly data-driven, where the bandwidth is the only parameter to tune the output of the algorithm. Mean shift clustering seeks for the modes of the histogram of the data. In this case mode is the intensity value x at which the probability mass function takes its maximum value. To this end, we define the bandwidth, a window, centered around each intensity value in the color space. The size and shape of the window is defined by us in accordance with the image we have and the result that we are expecting. The effect of changing window is that number of clusters will change. Larger window tends to lower number of clusters while smaller window tends to more number of clusters as expected. To find the modes, we estimate the mean value of the window centered around each value for all the intensity values of the image in the color space. Then, we shift the centroid (center of the sliding window) to the mean value, The stopping criteria is when the iterations comes to an end such that there are no more shifts in the window. Thus, the trajectories of all the mean shift vectors converges to the modes of the color

histogram, which estimates the mean vectors of the Gaussian mixture density. When compared with the other clustering algorithms, here clustering is done independent of cluster shape and number of clusters. So we do not need to define number of clusters in advance. The algorithm itself will find suitable value for number of clusters. In addition, Mean-shift algorithm provides clusters with irregular shapes which may be beneficial in some cases.

Here the Mean-Shift algorithm:

- Initialize random seeds to start the clustering process, and define the size and shape of the window W.
- Compute the mean of the window W.
- Shift the search window to the computed mean.
- Repeat from the step 2 until the convergence.

When multiple sliding windows overlap, the window containing the most points is preserved, and the others are deleted. At the end, we map intensity values to the mean of the sliding window in which they reside. Hence we get our clusters.

With direct implementation of segmentation, we observed that the regions we expected to be in the same segment were in different segments or reverse due to different lightning conditions. Therefore as a pre-processing we changed the contrast of the images, which caused smaller number of clusters and so more compact segments for some images.

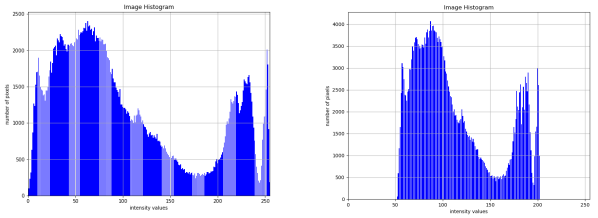

Fig. 8. Histogram of Image B1 with contrast 127 and 75 respectively

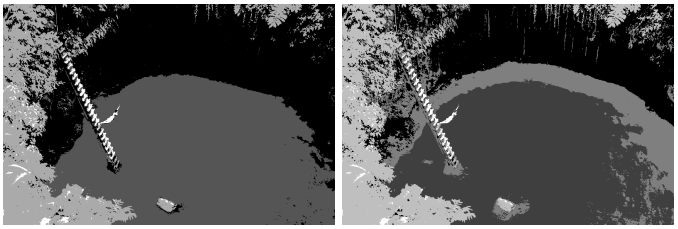

Fig. 9. Mean-Shift Segmentation with Contrast Values 127 and 75 respectively

In programming, we a function for brightness adjustment and stretching and narrowing the contrast of the images.

```
def enhance(image, brightness = 127, contrast = 127
    ):
    image = np.int16(image)
    enhanced = image * (contrast/127) + brightness -
        contrast
    normalized = np.clip(enhanced, 0, 255)
    enhanced = np.uint8(normalized)
    return enhanced
```

For implementation of the Mean-Shift algorithm, we used sklearn library functions estimatebandwidth() and MeanShift(). For initialization of the bandwidth, we decide parameters quantile and number of samples for the estimation of bandwidth. Quantile is the number, where a histogram is divided into equal-sized subgroups. Quantile times number of intensity values in these subgroups gives us the average number of pixel values in the initial clusters of Mean-Shift. Then, bandwidth will be calculated based on the average pairwise distances between the pixels that are in the same cluster. So estimated bandwidth increases with increase in quantile resulting in less number of clusters at the output of the algorithm(see figure (10)).

For the MeanShift() function, the most significant parameter is the bandwidth that we calculated just before. MeanShift() function returns us the clusters so on cluster labels by using clusters.labels. Since some of the clusters are gathered in the same region (converged to same value) their labels are the same. Therefore, we only take one of the labels per converged value to know the number of clusters/segments. We did this by using np.unique() function to extract unique labels. Finally we colored the given image with respect to labeling with different colors for each segment(0 to 255 in grayscale divided by number of clusters).

Fig. 10. Mean-Shift Segmentation with Quantile Values 0.2 and 0.3 respectively

```
def segmentation_function_meanshift(image,quantile):
    shape = image.shape
    result = np.zeros((shape[0], shape[1]), np.uint8
        )
    reshape_img = np.reshape(image, [-1, 3])

    bandwidth = estimate_bandwidth(reshape_img,
        quantile= quantile, n_samples=500)
    clusters = MeanShift(bandwidth=bandwidth,
        cluster_all=True, bin_seeding=True)
    clusters.fit(reshape_img)
    labels = clusters.labels_
    labels_unique = np.unique(labels)
    number_of_clusters = len(labels_unique)
    segmented = np.reshape(labels, shape[:2])
    if len(labels_unique) == 1:
        print("only 1 segment")
    else:
        for i in range (0,number_of_clusters):
            segment_row, segment_column = np.where(
                segmented == i)
            result[segment_row, segment_column] = i
                *255//(number_of_clusters-1)
    return result
```

Contrast values for image pre-processing and quantile parameters of estimate bandwith() function is decided by trial

and error for each images. With these parameters we applied Mean-Shift segmentation to all images.

```
mean_contrast = [127,50,75]
mean_parameters = [0.45,0.1,0.13] #[0.4,0.1,0.2]
for i in range (1,4):
    # Read the input image
    img = cv2.imread("./partB/B{}.png".format(i), 1)
    enhanced_img_1 = enhance(img, 127, mean_contrast
        [i-1])
    mean_segmented = segmentation_function_meanshift
        (enhanced_img_1, mean_parameters[i-1])
    cv2.imwrite("the3_B{}output_meanshift.png".
        format(i), mean_segmented)
```

Results:

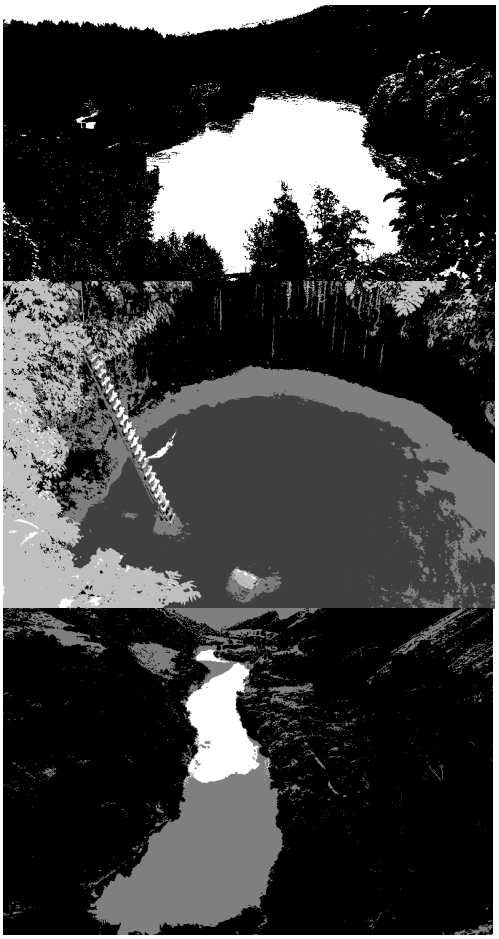

Fig. 11. Mean-Shift Segmentation Results

B. Normalized-Cut

An arbitrary image can be represented as a weighted undirected complete graph G = (V, E), where the nodes of the graph are the pixel poins. The weight w_{ij} of an edge $e(i,j) \in E$ is a function of the similarity between the nodes i and j. In this context, we can formulate the image segmentation problem as a graph partitioning problem that asks for a partition $V_1, ..., V_k$ of the vertex set V, where, according to some measure, the vertices in any set V_i have high similarity, and the vertices in two different sets V_i, V_j have low similarity.

A graph G = (V, E) can be partitioned into two disjoint sets A, B such that $A \cup B = V$ and $A \cap B = \emptyset$, by simply removing edges connecting the two parts. The degree of dissimilarity between these two pieces can be computed as total weight of the edges that have been removed. In graph language, it is called the cut:

$$cut(A,B) = \sum_{v_i \in A, v_j \in B} w(i,j) \tag{3}$$

The optimal bipartitioning of a graph is the one that minimizes this cut value. The minimum cut criteria favors cutting small sets of isolated nodes in the graph. This is not surprising since the cut defined in equation (3) increases with the number of edges going across the two partitioned parts. We can see that the cut that divides out node n1, n2 will have a very small value if the edge weights are inversely proportional to the distance between the two nodes. In fact, any cut that partitions out individual nodes on the right half will have smaller cut value than the cut that partitions the nodes into the left and right halves.

To avoid this unnatural bias for partitioning out small sets of points, Shi and Malik proposed a new measure of disassociation between two groups. Instead of looking at the value of total edge weight connecting the two partitions, their measure computes the cut cost as a fraction of the total edge connections to all the nodes in the graph. This disassociation measure is called the normalized cut (Ncut):

$$Ncut(A,B) = \frac{cut(A,B)}{Asso(A,G)} + \frac{cut(A,B)}{Asso(B,G)} \tag{4}$$

Association of a subgraph, A, where $A \subset G$ is defined as the total weights, node degrees of the nodes in A:

$$Asso(A,G) = \sum_{v_i \in A} d(i) \tag{5}$$

Dividing the $cut(A,B)$ by $Asso(A,G)$ and $Asso(B,G)$ makes the normalized cut independent of the degree of connectivity of the subgraphs A and B to the graph G. Consequently, Ncut avoids partitioning the image graph into a very small and a very large subgraph.

Here the N-cut segmentation algorithm:

- Given an image or image sequence, set up a weighted graph G = (V, E) and set the weight on the edge connecting two nodes to be a measure of the similarity between the two nodes.
- Let x be an Nx1 dimensional indicator vector, $x_i = 1$ if node i is in A and $x_i = -1$, otherwise. Also let D be an NxN diagonal matrix with $d(i) = \sum_j w(i,j)$ on its diagonals. d(i) be the total connection from node i to all other nodes. W be an NxN symmetrical matrix with $W(i,j) = w_{ij}$ Thus, Solve $(D-W)x = \lambda Dx$ for eigenvectors with the smallest eigenvalues.
- Use the eigenvector with the second smallest eigenvalue to bipartition the graph
- Decide if the current partition should be subdivided and recursively repartition the segmented parts if necessary.

In programming we used skimage library for implementation of N-cut algorithm. Graph similarity matrix with respect to colors is created by the function graph.ragmeancolor() with given image and its initial segmentation as parameters. Initial segmentation of the image is done by segmentation.slic() function using k-means clustering in color space. Here in the initial segmentation we have two parameters, compactness and number of segments. Compactness balances color proximity and space proximity. Higher values of compactness give more weight to space proximity, making superpixel shapes more square/cubic. Number of segments clearly determines number of segments in the initial segmentation. Therefore, its contribution to output is quite significant. Finally, the function graph.cutnormalized() with parameters labels(initial segmentation) and graph similarity matrix, returns us the final segmentation covering all the steps 2-4 of the algorithm above. At last, we colored the output segments with color.label2rgb() function.

```
def segmentation_function_ncut(image,
    initial_number_of_segments = 400):
    labels = segmentation.slic(image, compactness =
        30, n_segments = initial_number_of_segments,
         start_label = 1)
    graph_similarity_matrix = graph.rag_mean_color(
        image, labels, mode='similarity')
    labels = graph.cut_normalized(labels,
        graph_similarity_matrix)
    result = color.label2rgb(labels, image, bg_label
        = 0, kind = 'avg')
    return result


ncut_contrast = [75,127,75]
ncut_parameters = [400,50,100] #
    initial_number_of_segments for segmentation.slic
    ()
for i in range (1,4):
    img = cv2.imread("./partB/B{}.png".format(i), 1)
    enhanced_img_2 = enhance(img, 127, ncut_contrast
        [i-1])
    ncut_segmented = segmentation_function_ncut(
        enhanced_img_2, ncut_parameters[i-1])
    cv2.imwrite("the3_B{}_output_ncut.png".format(i)
        , ncut_segmented)
```

As discussed in the Mean-Shift part, we changed the contrast of some of the images for better segmentation.

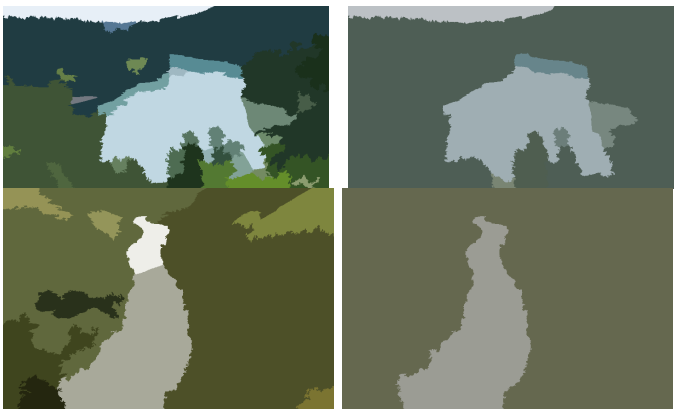

Fig. 12. N-Cut Segmentation with Contrast Values 127 and 75 respectively

For optimal segmentation, we experimented with the initial number of segments.

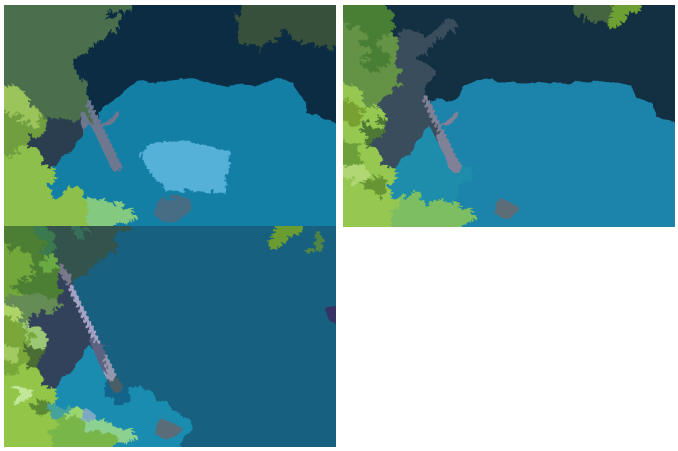

Fig. 13. N-Cut Segmentation with Initial Number of Segments 50, 200 and 400 respectively

Here are the segmentation outputs that we think best summarize the images:

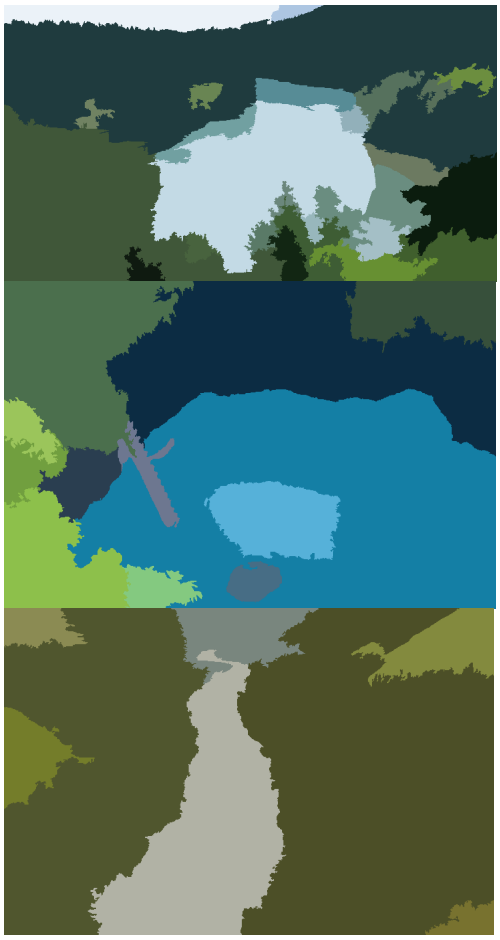

Fig. 14. N-Cut Segmentation Results

If we compare the Mean-Shift and N-Cut algorithms, it is

obvious that the N-Cut algorithm is much more successful in segmenting the regions as a single piece. But at the same time, it is necessary to play with the parameters to achieve successful results with N-Cut. On the other hand, the Mean-Shift algorithm often gives sufficient results if its only parameter, bandwidth, is chosen appropriately. However, post-processing is required for some regions to be segmented in one piece. While the Mean-Shift can easily distinguish small regions with large differences, N-Cut is unable to do so due to the bias of separating segments of equal size. In terms of time complexity, while Mean Shift with O(n^2) is more complex than N-Cut with O(n), N-Cut demands much more storage space when compared this two. As a conclusion, although both algorithms give sufficient results, we think that N-Cut is more successful in extracting segments in one piece, since it does not require any post processing.

REFERENCES

[1] Muhammet Bolat. *Image segmentation using K-means clustering algorithm and mean-shift clustering algorithm*. June 2020. URL: https://medium.com/@muhammetbolat/image - segmentation - using - k - means - clustering - algorithm - and - mean - shift - clustering - algorithm - fb6ebe4cb761.

[2] Damir Demirović. "An implementation of the mean shift algorithm". In: *Image Processing On Line* 9 (2019), pp. 251–268. DOI: 10.5201/ipol.2019.255.

[3] *Image processing for object counting*. Jan. 2022. URL: https://www.abtosoftware.com/blog/image-processing-for-object-counting.

[4] Edouard TETOUHE KILIMOU. *Image processing: Segmentation and objects counting with python and opencv*. Oct. 2019. URL: https://medium.com/analytics-vidhya/images-processing-segmentation-and-objects-counting-in-an-image-with-python-and-opencv-216cd38aca8e.

[5] Jianbo Shi and Jitendra Malik. "Normalized cuts and image segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.8 (Aug. 2000), pp. 888–905. DOI: 10.1109/34.868688.

[6] Tong Zou et al. "Recognition of overlapping elliptical objects in a binary image". In: *Pattern Analysis and Applications* 24 (Aug. 2021). DOI: 10.1007/s10044-020-00951-z.

[5] [2] [1] [6] [4] [3]