CALEB J. GROHMANN

Assignment 1

February 1, 2022

CMP\_SC 8650 Computer Vision

Professor Ye Duan

INTRODUCTION

The objective of this assignment was to perform image binarization using thresholding A) before and B) after visual assessment of a histogram of image pixel values and C) image segmentation using the K-means algorithm for unsupervised clustering. Below, parts A, B, and C will outline methodology and results of each of these objectives. All programs were written in Python (v3.9.7) and OpenCV4 was used for basic image processing tasks. The below images will be the test images used for this report [one grayscale (left; Test Image 1) and one RGB color (right; Test Image 2)].



Figure 1. Test Image 1 (left) and 2 (right).

PART A: IMAGE BINARIZATION USING NAÏVE THRESHOLDING

The objective of this program, *threshold.py*, was binarization of grayscale image pixel values (0 to 255) based on a user-defined threshold, *t*. Pixel values that were less than *t* were converted to 0 (black) and pixel values that were greater than or equal to t were converted to 255 (white). Helper information for the command line usage of the program is shown below in Figure 1.

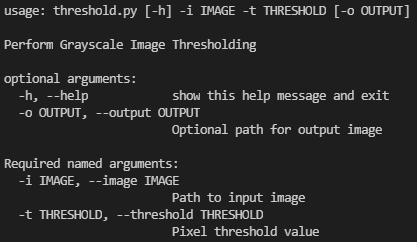


Figure 2. Usage information for program 'threshold.py'.

Required arguments are the path to the input image (-i or --image) and threshold value (-t or --threshold). In addition, the flag ‘-o’ or ‘--output’ allows the user to specify a path to save the binarized image. Results of implementing this program on Test Image 1 with the middle threshold value, 128, are shown below.



Figure 3. Output from 'threshold.py' using a threshold value of 128.

From this result, we see that the image was successfully binarized. When bumping the threshold value up and down 64 pixel values (right and left pictures, respectively), we see that there are less and more black pixels, respectively, as expected (Figure 4).

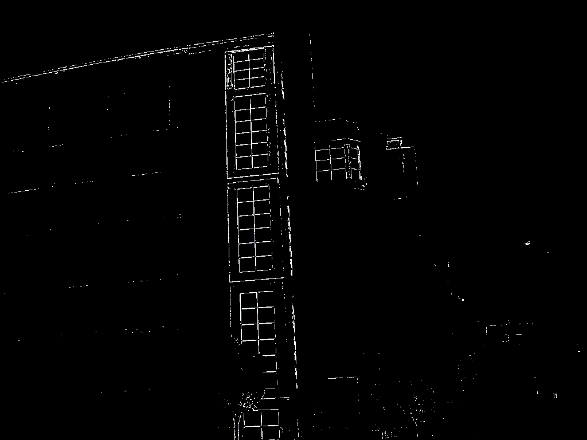
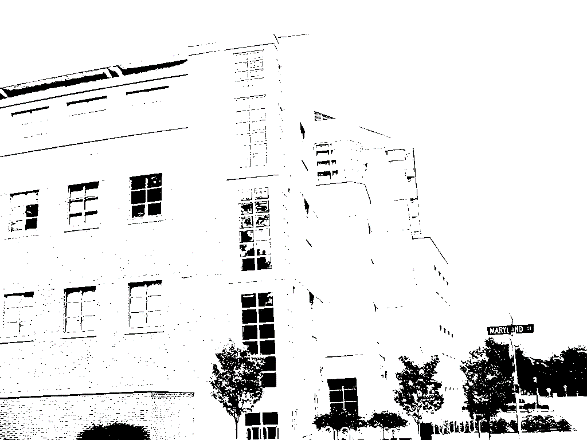


Figure 4. Results from 'threshold.py' using 64 (left) and 192 (right) as threshold pixel values.

PART B: IMAGE BINARIZATION USING NAÏVE THRESHOLDING AFTER VISUAL APPRAISAL OF HISTOGRAM

The program from Part A, *threshold.py*, was modified to produce the program for Part B, *histogram.py*. Below is the command line usage for the *histogram.py* program.

Text

Description automatically generated

As shown above, there is no requirement to specify a threshold pixel value until after the histogram and image are visually appraised. Once the below figures are evaluated, then the program will accept a threshold pixel value via a command line input.

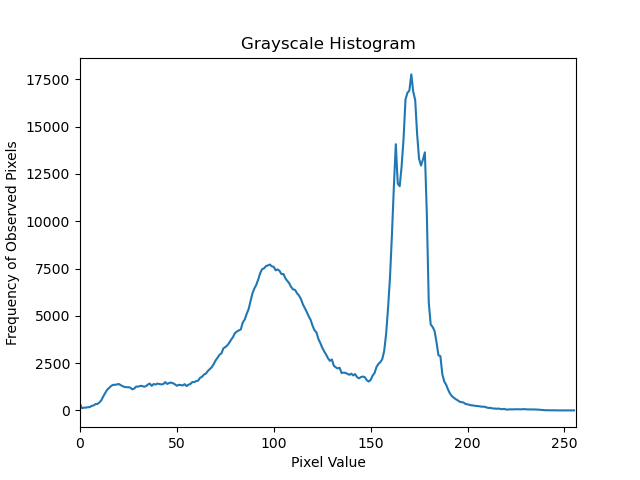
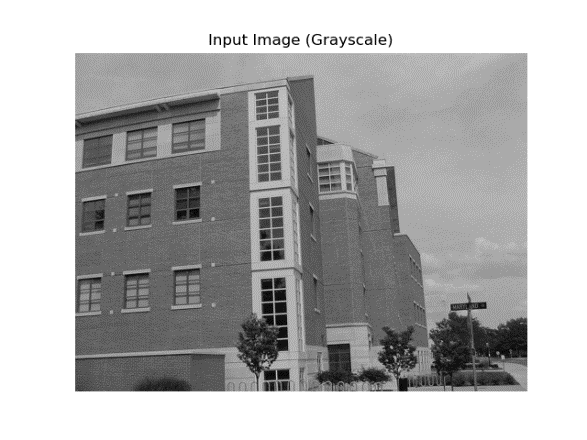


Figure 5. Test Image 1 (top left) and its histogram of pixel values (top right). A threshold value of 150 was selected upon visual appraisal (bottom).

There are two modes in the distribution of pixel values for Test Image 1 (Figure 5). To optimally split these modes, we selected a threshold pixel value of 150 (Figure 5). The binarized image results are shown below.

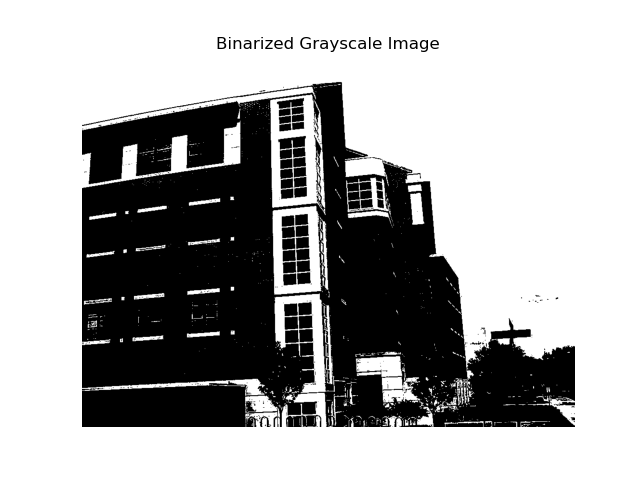


Figure 6. Binarized Test Image 1 using a threshold pixel value of 150.

Above, we see that 150 is likely the optimum threshold to separate the building (foreground) from the sky (background). Selection of this value was only possible after visual appraisal of the histogram of grayscale pixel values.

PART C: Color Image Segmentation Using K-Means Clustering

Image segmentation was performed using a from scratch implementation of the K-means unsupervised clustering algorithm. Test Image 2 was used for this part of the assignment. The goal of the K-means algorithm is to separate similar pixels in an image into *k* clusters that share high similarity scores based off of Euclidean distance between each pixel in the cluster and the centroid of the cluster. A class named KMeansClustering was built in Python that contained relevant user-defined functions to the KMeansClustering method. An example workflow for Test Image 2 is shown in the figures below. The image was resized to 666 × 1000 pixels for computational efficiency (Figure 7). Next, the image required reshaping into a 2D array (Figure 7).

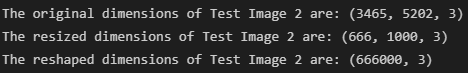


Figure . Test Image 2 resized and reshaped dimensions.

Three different K values were tested (3, 5, and 7), and that code is shown below. Source code for the KMeansClustering class is shown in depth in *kmeans.py*.

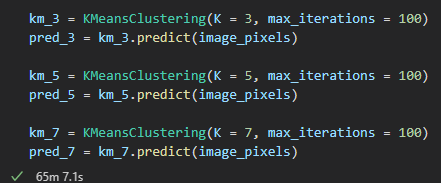


Figure . Code used to run K-means clustering on Test Image 2.

Results from the output were manipulated to create each clustered image using the below code.

A screenshot of a computer

Description automatically generated with medium confidence

Lastly, the resulting clustered images are shown in Figure 9.

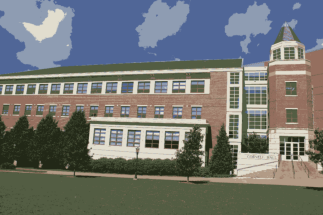
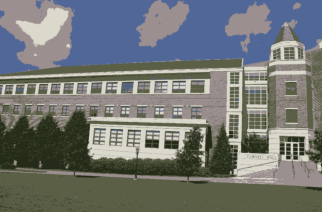


Figure . Results of K-means clustering using 3 (top), 5 (middle), and 7 (bottom) clusters.

We see that as we increase the number of potential clusters from 3 to 7, the image gets closer to its original form. In addition, the algorithm works well in pairing pixel values that are similar in color into their respective groups.

CONCLUSIONS

The programs presented in this report allow users to perform image threshold binarization (with and without a histogram) and image segmentation with the K-means clustering algorithm. In addition, many learning objectives were personally met. For example, writing the K-means algorithm from scratch allowed me to truly understand what the algorithm is doing to classify pixel values. Knowledge gained from this assignment will be used to employ future computer vision related analyses in my PhD program.