NATIONAL UNIVERSITY OF SINGAPORE

ME5405 MACHINE VISION

Computing Team Project

Assignment – AY21/22 Semester 1

Members	Matrix No.
Jin Hoontae	A0243155L
Wan Yi	A0242941H
Xu Shihao	A0243135N

Table of Contents

Page No.

1. Introduction

1.1. Backgrounds

In the history of machine vision, there have been numerous approaches, methods, and algorithms developed in order to perform better imaging-based analysis and inspection for various applications, especially in industry. The technology covers both industrial and non-industrial applications such academical/educational, governmental/military applications in which constrains are different for each application. Throughout the project given in this module, we were able to utilize basic image processing methods taught in the lectures and fully comprehend the principles of the methods.

1.2. Given Tasks

The project was divided into two main parts where each part contained sub-parts with the given images. For both parts, images whose resolution was 64x64 with the gray level of 32 was given. The images were comprised of characters 0-9 and A-V and contained different objects. The raw/unprocessed images can be found in **Figure 1**.

The project was completed in such a manner in which multiple different algorithms could be constructed sequentially. The details of the algorithm construction will be explained in **Section 2**, and the processed images will be shown and compared with the MATLAB built-in functions. The tasks to be performed for each part are as follows:

Part 1

1. Display the original image on screen.
2. Threshold the image and convert it into binary image.
3. Determine a one-pixel thin image of the objects.
4. Determine the outline(s).
5. Label the different objects. Discuss and compare 4-connectivity and 8-connectivity in your report.

Part 2

1. Display the original image on screen.
2. Create a binary image using thresholding.
3. Determine a one-pixel thin image of the characters.
4. Determine the outline(s) of characters of the image.
5. Segment the image to separate and label the different characters.
6. Arrange the characters in one line with the sequence: **AB123C**

7. Rotate the output image from Step 6 about its center by 30 degrees.

8. Using the training dataset provided on *LumiNUS (p_dataset_26.zip)*, train the (conventional) classification method of your choice (i.e., self-ordered maps (SOM), k-nearest neighbors (kNN), or support vector machine (SVM)) to recognize the 6 characters ("1", "2", "3", "A", "B", or "C"). You should use 75% of the dataset to train your classifier, and the remaining 25% for validation (testing). Then, test your trained classifier on each character in image 2, reporting the final classification results. **Do not use the characters in image 2 as training data for your classifier**

9. Throughout step 8 (training of the classifier), also experiment with pre-processing of the data (e.g., padding/resizing input images) as well as with hyperparameter tuning. In your report, discuss how sensitive your approach is to these changes.

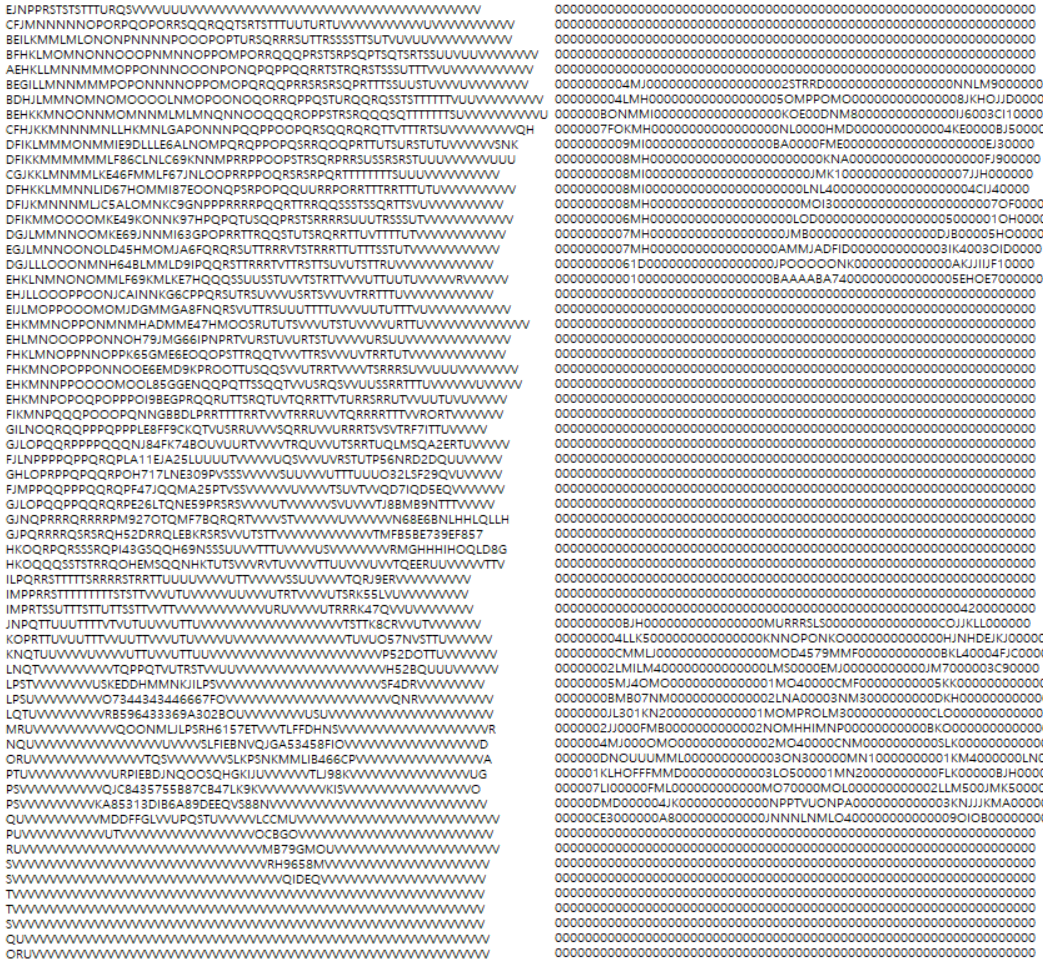

Figure 1. Raw/Unprocessed images (left: chromo for part 1, right: charact1 for part 2)

The tasks were performed with the minimal use of MATLAB built-in functions to compare each algorithm subsequently. The algorithm flowcharts are described in the appendix section.

2. Methodology and Algorithms

2.1. Original Image Display

Images are organized and stored in multiple image file formats. Data in images can be stored in representatively three format types; that is, an uncompressed format, a compressed format, and a vector format. Digital data are contained in images in one of the formats, and the images can be displayed after rasterizing the data. Rasterization is the process where the image data are converted into a grid of pixels that determines its color according to a number of bits. For the images given in this project, each character was designated corresponding pixels to perform the image display successfully. The alphabets, **A-V** in the images, were properly mapped to corresponding pixel values to fit within the range of the gray level 32 as advised in the project sheet.

2.2. Thresholding

Thresholding is a method that produces a binary image based on the thresholding value. Image binarization is considered as the simplest way of segmenting images to reset the pixel value with only two values, 0 and 1 for objects and backgrounds. The pixel values (0 and 1) are determined based on a threshold value. Pixels below the threshold value are usually converted to black (0), and pixels above it are converted to white (1). The histograms for both images are shown in **Figure 2**. Generally, the method is utilized to extract the desired objects/foreground from the background where threshold values differ depending on the images.

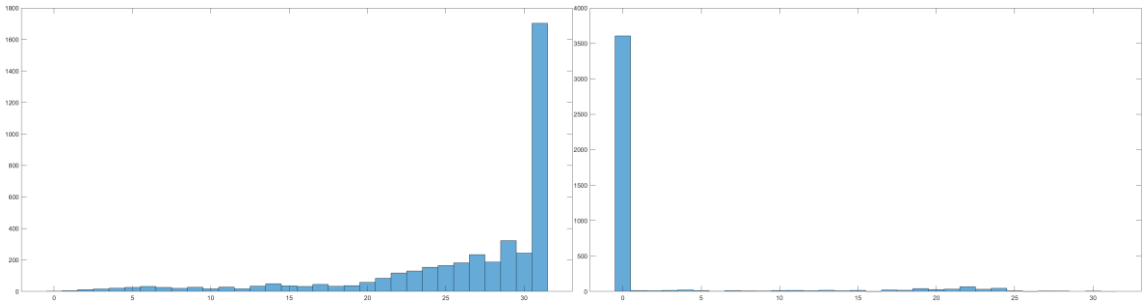

Figure 2. Histograms of the image 1 (left) and the image 2 (right)

Four thresholding algorithms were attempted to check their practicality in both images.

• ***Basic Global Thresholding***

This thresholding method performs an iterative process to compute a new thresholding value, T, which is smaller than the predefined thresholding value, T_0. The process is as follows:

1. Randomly determine an initial estimate for T_0.

2. Segment the image according to T thereby producing two groups of pixels: G1 containing all pixels with gray level values equal to or less than T, and G2 containing all pixels with gray level values greater than T.

3. Compute the average values $\mu 1$ and $\mu 2$ for the pixel values in G1 and G2.

4. Obtain a new threshold value: $T = \frac{1}{2}[\mu 1 + \mu 2]$

5. Iterate step 2 through 4 continuously until the change of T becomes significantly small.

• ***Optimum Thresholding (Otsu's Method)***

This algorithm divides pixels into two classes – background and foreground, based on a single intensity threshold. The threshold computation involves an iterative process of examining all the threshold values available in the image and measuring an extent of the contract between background and foreground. When the contract is minimized, meaning that the intra-class variance is minimized, the thresholding value is determined. The process is as follows:

1. Process the input image.

2. Obtain the histogram and probabilities of each pixel in the image.

3. Define two variables: $w_i(0)$ and $\mu_i(0)$ for the class probability and class mean of i respectively.

$$w_0\,(t) = \sum_{t=0}^{t-1} p(i) \qquad u_0(t) = \frac{\sum_{i=0}^{t-1} ip(i)}{w_0(t)}$$

$$w_1\,(t) = \sum_{t=t}^{L-1} p(i) \qquad u_0(t) = \frac{\sum_{i=t}^{L-1} ip(i)}{w_1(t)}$$

Where t = possible threshold values, L= bins of the histogram, and i=intensity

4. Iterate through all the possible threshold values (T). During the step, update w_i and μ_i consecutively and compute the between-class variance $\sigma_b^2(T)$.

$$\sigma_b^2(T) = w_0(t) w_1\,(t)[u_0(t) - u_1(t)]^2$$

5. Select the threshold that corresponds to the maximum of $\sigma_b^2(T)$.

• ***Adaptive Gaussian-c Thresholding***

Unlike global thresholding methods where a single threshold value is applied to all the pixels in the image, adaptive thresholding methods compute the threshold value for smaller regions, resulting in different threshold values for each region. To each image, $n \times n$ Gaussian kernel is employed to scan and compute a threshold value for each fractional region. The process for the adaptive Gaussian thresholding method is as follows for the image 1 and 2:

1. Construct the 3×3 *Gaussian* kernel used to scan the images.

$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-[((x-(k+1))^2 + ((y-(k+1))^2]}{2\sigma^2}}$ where σ is 1 by default, and k is obtained from the kernel size $(2k+1) \times (2k+1)$

2. Convolve the image with the kernel and obtain threshold values for each region by summing the convolved values and locating them in an appropriate manner.

3. The image is binarized under the condition; that is, (convolved image– original image) > C where C is the constant value. Usually, 3 is used by default but different values can be used.

4. Invert the binarized image to capture the final objects.

• ***Adaptive Mean-c Thresholding***

The process for the adaptive mean-c thresholding is performed in a similar way to the Adaptive *Gaussian* thresholding, but a different kernel is utilized.

1. Convolve the image with a suitable kernel operator. In this process, a 3×3 kernel composed of a single value, 1, is used. During the process, mean values need to be computed for each region.

2. Subtract the original image from the convolve image.

3. Thresh the subtracted image with C where the value, 3, is used by default.

4. Invert the binarized image.

2.3. Thinning

The thinning method, also known as skeletonization, is the reduction process where a digital image is reduced to the minimum size while preserving the major points. By utilizing the method, processing time can be saved, and redundant features and image noise can be eliminated. There are multiple algorithms available to implement the thinning process such as Zhang Suen Thinning algorithm, Canny Edge detection, Edge Based Thinning algorithm, and Guo and Hall's parallel Thinning algorithm.

In this report, Zhang Suen Thinning algorithm was utilized. The algorithm works on skeletonizing binary images, and it operates on all black pixels, **P1**, surrounded by eight neighbors as shown in **Figure 3**.

P9	P2	P3
P8	**P1**	P4
P7	P6	P5

Figure 3. 3x3 Kernel with the black pixel P1 and 8 neighboring pixels (P2,P3,...,P9)

Since the boundary pixels cannot have the full eight neighbors, the P1 starts at the pixel point

of (2,2) and ends at (m-1, n-1) where m and n are the row and column lengths, respectively. Two variables, N(P1) and S(P1), need to be defined before performing the main process

A(P1)=The number of transitions from white (0) to black (1) in the sequence P2, P3, … P9, P2.

B(P1)=The number of non-zero/black pixel neighbors (=sum(P2, P3,…, P9).

The main process works on two iterative steps as below.

Step 1: All pixels are examined and pixels that meet all the following conditions are just recognized at this stage.

• Condition 0: The object pixel is black (1) and contains eight neighbors (P2-P9).

• Condition 1: 2 <= B(P1) <= 6

• Condition 2: A(P1) = 1

• Condition 3: At least one of P2, P4 and P6 is white (= P2×P4×P6 = 0).

• Condition 4: At least one of P4, P6 and P8 is white (= P4×P6×P8 = 0).

After examining all the pixels in the image, the recognized pixels are converted to 0.

Step 2 iterates the same process as ***Step 1***. If any pixel is converted to 0 in either step 1 or step 2, all steps will be repeated until there is no further change of pixels in the image. This iterative process is performed to ensure an effective thinning.

2.4. Outlining

Outlining, also called edge detection, is an effective technique of image processing used to segment an image into regions of sharp changes or discontinuities. The main purpose of this method is to capture important features in an image and erode unimportant pixels. The ideally outlined image may contain a set of connected curves that represent the boundaries of objects, and curves that correspond to discontinuities. The processed image contains less amount of data while preserving the important structural properties of the image.

In this report, Sobel operator was used. It is a discrete differentiation operator that creates emphasizing edges in an image by computing the gradient approximation of the image intensity function. The operator calculates the vertical and horizontal derivative approximation by using two 3x3 kernels, which are convolved with the original image. The traditional 3x3 kernels are shown in **Figure 4**.

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \qquad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Figure 4. 3x3 kernels for Sobel operator

The two processed images in x- and y-directions are then combined together to create a new image that is comprised of the sum of x and y edges of the image. Hence, the resulting magnitude at every coordinate can be computed as:

$$G = \sqrt{G_x + G_y}$$

However, before performing the operator, the original images are convolved with a 3×3 *Gaussian* kernel for this particular project and after the operator procedure, an assumption is made, which is that the value of a center pixel within the 3×3 neighboring matrix remains unchanged when it is greater than the other ones and is suppressed otherwise. This assumption successfully yielded excellent visual results, as compared to solely using the Sobel operator.

2.5. Labeling Objects by Connected Component Labeling

Binary Connected Component Labeling (CCL) methods are commonly employed for image processing because they can effectively recognize objects in a digital pattern. The algorithms can locate groups of connected black pixels by scanning an image row by row. Every black pixel is given certain pixel intensity values and subsequently labeled with a grey level value. Hence, this process plays a central role in detecting and labeling objects. In this report, 4-connected and 8-connected component labeling were used to identify the objects in the images given. The 4-CCL and 8-CCL methods check different neighbors as shown in **Figure 5**. The gray-colored pixels are the ones that are checked when a black pixel, *P*, is scanned.

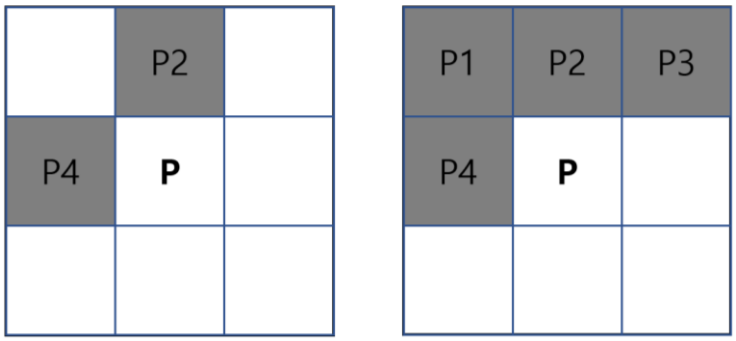

Figure 5. Locations of pixels that are examined for labeling (Left: 4-CCL, Right: 8-CCL)

The CCL operators scan a binary image row by row until they reach to a point *P* whose intensity value is 1 (black), and examines the grey-colored neighbors in the Fig (P1, P2,P3, and P4), which have already been scanned previously. Depending on the conditions, a grey level value for labeling *P* differs. For example, in the 4-CCL case, if the values of P2 and P4 are zero, the P value is automatically labeled with a new number or if the P2 has a positive value, the P is labelled with the same value to be grouped with the pixel. This consecutive process eventually identifies and groups the objects with unique values in the image.

2.6. Re-arrangement of the objects

To re-arrange specific objects in the image, each object needs to be labeled with unique values beforehand. Hence, the above-mentioned method, CCL, is an essential process that needs to be done before re-arranging the objects. In the image 2 given in this project, the shapes of the characters were relatively simple, thus the bounding box approach was attempted. The method basically returns the smallest rectangle bounds that enclose the desired objects appropriately. The coordinates are obtained by manually recognizing the locations and given CCL values of the objects and by utilizing the *IF* statement to pinpoint them. The specific coordinates obtained from the method are used to crop out the objects and create new variables. Finally, the rearrangement is completed by combining each matrix of the variables in the sequence given (A, B, 1, 2, 3, C).

2.7. Rotation

To rotate an image about its center, one thing that needs to be considered is different array dimensions created by different rotational angles. Hence, to prevent an image from being truncated, additional arrays are required to fit the output image appropriately. The optimal array dimensions are computed as below. The following equations dynamically produce a desirable background with regards to an angle input.

$$row_{new} = round\big(row_{original} \times \cos(rad) + + col_{original} \times \sin(rad)\big)$$

$$col_{new} = round\big(row_{original} \times \sin(rad) + + col_{original} \times \cos(rad)\big)$$

After creating the new background, centroids of the original and final images need to be calculated. Subsequently, to rotate about the center of the image, the following steps are required:

1. Define two variables (x, y) where each pixel dimension (i, j) is subtracted by the centroids of the output image

2. Employ the rotation matrix and multiply the results obtained in the step 1 with the matrix

$$[x_{rotate} \quad y_{rotate}] = [x \quad y] \begin{bmatrix} \cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$$

3. Change the coordinate system by adding the centroid points of the original image into each variable (x_{rotate}, y_{rotate}). This process helps the output image shift towards the center point of the array dimensions thereby showing whole objects in the background.

2.8. Number Recognition

K-nearest number (KNN) method is used to realize number recognition.

KNN determines the category of the test picture by comparing the gap between the test picture and the training picture. The gap refers to the sum of the difference of the gray value of each pixel between test image and training image. Among the top K training pictures with the smallest gap with the test picture, the most frequently appeared category is most likely to be the category of the test picture. Please refer to **Figure 25** in the appendix for detailed procedure description.

The performance depends on the value of K. And K is manually selected based on experimental results. Please refer to section 3.7 for experimental results.

3. Performance Analysis

3.1. Original Image Display

Both image 1 and image 2 were originally in a text file format. The txt files were comprised of numbers (0-9) and capital letters (A-V), which were read as characters in MATLAB. To satisfy the given requirements in which the characters correspond to 32 levels of grey, the alphanumeric characters (A-V) needed to be converted to the appropriate numbers with A being 10 and V being 31. By creating a dictionary table, each character could be assigned to specific numbers. When displaying the images using *Imshow()* function, *'[0 32]'* needs to be put inside the function as it always reads images with the grey level of 255 as default.

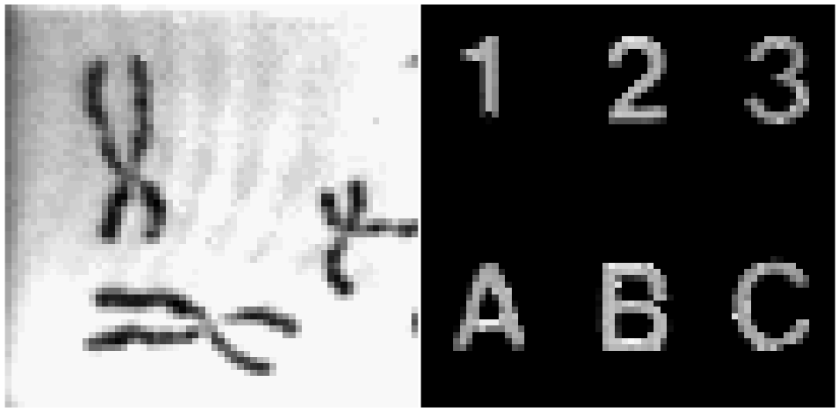

Figure 6. Original Image Display (Left: Image 1, Right: Image 2)

3.2. Thresholding

As described in **Section 2.2**, four threshold methods were employed to visualize binary images for the images given. Different threshold values were obtained through each algorithm and the processed images were compared with the original images to identify regions of interest while neglecting the unimportant parts as shown in **Figure 7**.

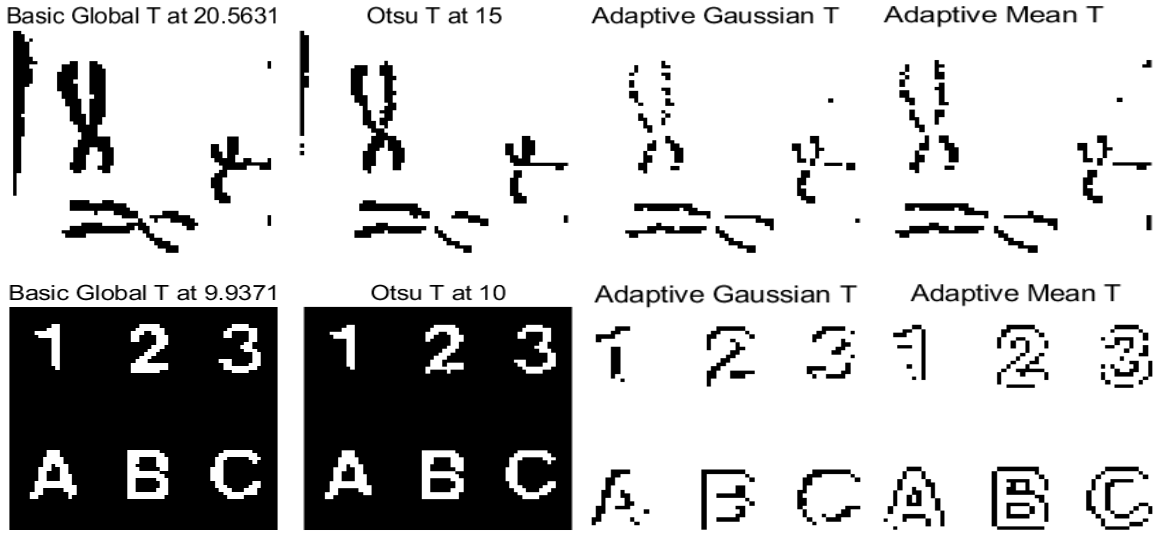

Figure 7. Binary images produced by thresholding methods

The processed binary images by the basic global thresholding and Otsu's methods both preserve important features of the objects. Similarly, the other binary images obtained by the adaptive *Gaussian-c* and Mean-c thresholding methods also preserve the important information. However, as seen in the adaptively-thresholded chromosome images, the objects are not clearly connected for the image 1, which could affect the subsequent tasks such as thinning, and labeling due to their disconnected lines, despite the significantly reduced noises in the background. For the image 2, the adaptively-thresholded images clearly display the better visual representation with smoother shapes. However, when the two images were used for the thinning and connected component labeling parts, they produced dissatisfying results making the characters look rather fatter as compared to the ones in the original image. Hence, the adaptively-thresholded images were excluded and not considered for the next parts.

Figure 8 shows a specific object in the chromosome images. In the original image, the chromosome is seen as one complete entity. The same chromosome in another image (Basic Global T) is not fully connected but seems to maintain the important features of the object. For the last one in the other image (Otsu T), the object is completely disconnected by four separate lines, which could fatally impact the CCL performance resulting in four separate groups of pixels. Hence, the basic global thresholding method was employed for further performance.

Figure 8. Magnified regions of the image 1

For the image 2, Otsu's method is chosen to be the optimal method. This is because the method computes the same threshold value whereas the basic global one produces different threshold values ranging from approximately 20.5 to 21.5 because it randomly chooses an initial threshold value. The different values significantly were deemed to complicate the rearrangement part as the object dimensions were continuously changed by the different threshold values.

3.3. Thinning

To optimize and simplify the margin determination, the methods of 'Basic Global Thresholding' and 'Otsu's method' were employed to produce the binary image for the image 1 and the image 2, respectively. Subsequently, Zhang-Suen thinning algorithm was utilized to perform the given task.

3.3.1. Thinning on Image 1

Since the objects were initially black with the intensity value of 0 in the binary image, in this process, the pixel values of the objects needed to be converted to 1 and the background pixels to 0 to successfully perform the Zhang-Suen thinning algorithm. **Figure 9** shows the original binary image and the thinned image processed by the algorithm.

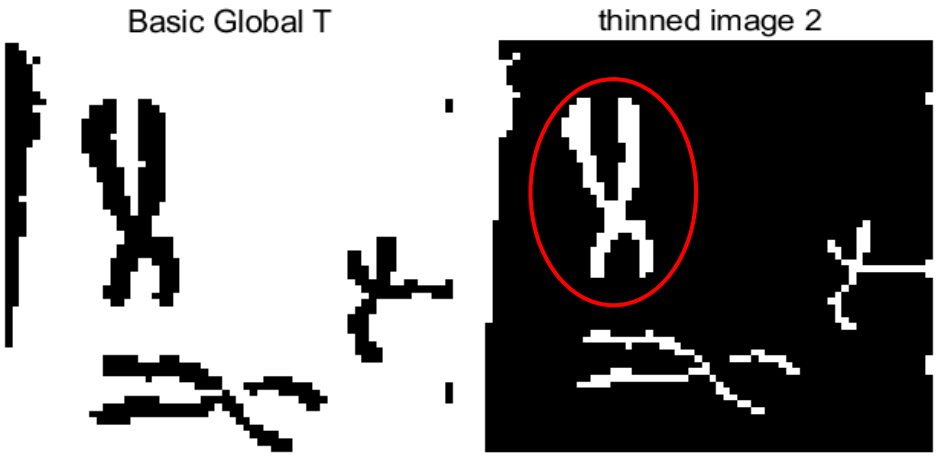

Figure 9. The original binary image and the thinned image processed by Zhang-Suen thinning algorithm

It can be seen from the figure that the algorithm does skeletonize the targeted objects to some extent, although the chromosome in the red circle does not show a significant change as compared to the one in the left image. In order to produce a better thinned image, perhaps, better threshold method needs to be applied.

3.3.2. Thinning on Image 2

The Zhang-Suen algorithm works properly on the image as shown in **Figure 10**. No disadvantages such as noise and distortion were found as all the characters seemed to preserve the main features appropriately. However, a confusion could occur from the character, *B*, because, in the thinned image, it can be interpreted as a number, 8. The phenomenon arises supposedly because the feature is not well defined in the original image.

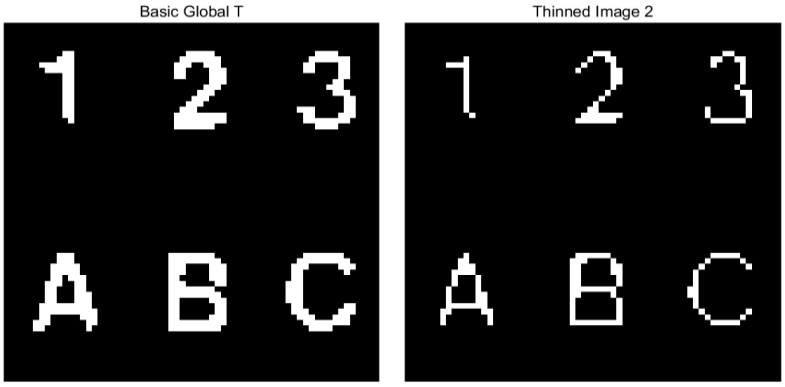

Figure 10. Images produced by Basic Global Thresholding (left) and Zhang-Suen thinning (right)

3.4. Outlining

To perform the outlining process, Sobel operator with additional procedures was utilized as described in **Section 2.4**.

3.4.1. Outlining on image 1

Figure 11 illustrates the processed images by our written algorithm. Even though the initial algorithm was based on the correct equations and procedures, the lines were thick and the black pixels inside the lines were relatively discontinuous, unsmooth and thin. Moreover, diagonal direction black pixels within the objects were not properly preserved. It was observed that the primary reason for this occurrence was due to the inherently small thickness of the objects and the small size of the image, which made the algorithm hard to process the task as required. In order to improve the image quality, the *Gaussian* filtering initially was performed to smoothen the whole image. After applying the Sobel operator to the filtered image, a new approach similar to the non-maximum suppression method was utilized. Instead of utilizing the *arctan* angles to compute the center pixels, they were determined depending on the 4-neighbouring pixel values. The new approach could effectively rectify the issue.

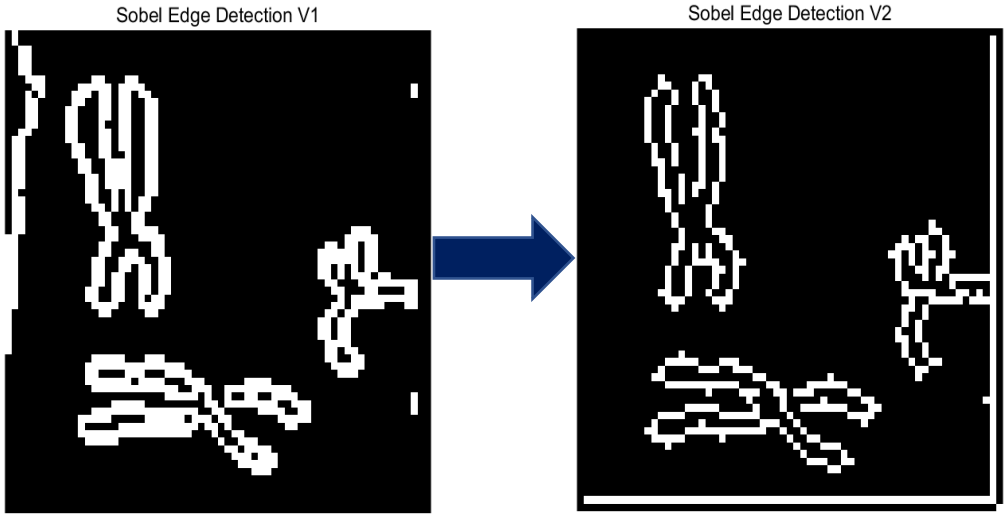

Figure 11. Outlined binary images produced by original and improved Sobel operators (left: Initial Sobel Operator, right: Improved Sobel Operator)

3.4.2. Outlining on image 2

The same approach as the one mentioned in **Section 3.4.1** was taken. Similarly, the initial Sobel operator produced three undesirable results: Thick lines, discontinuous black pixels within the objects, and thick object lines. The improved Sobel approach indeed improved the quality of the image, better-maintaining important features without disconnected lines as shown in **Figure 12**.

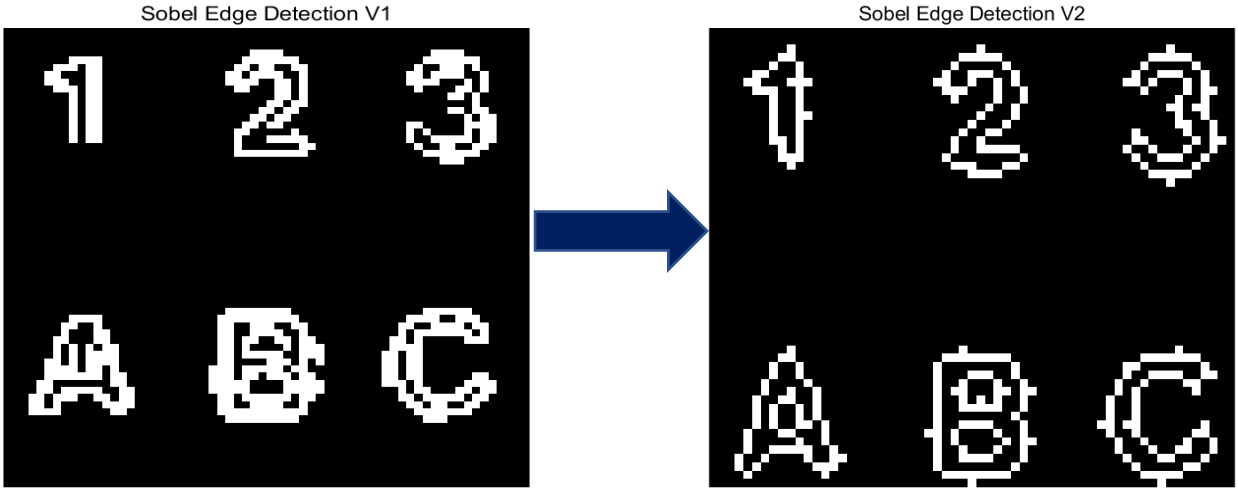

Figure 12. Outlined binary images produced by original and improved Sobel operators (left: Initial Sobel Operator, right: Improved Sobel Operator)

3.5. Labeling Objects based on Connected Component Labeling (CCL)

3.5.1. 4- and 8-CCL on Image 1

4 and 8 connected component labeling (CCL) methods were employed to label the objects in the image. As can be seen from **Figure 13**, the 8-CCL algorithm works more efficiently on identifying and labeling the objects. The 4-CCL algorithm failed to label the objects accurately and identified single objects as multiple ones. The reason why the 8-CCL algorithm performs better is because it is capable of scanning more neighbors, which are diagonal pixels. This approach offers better access to the information of neighbor pixels and produces more precise results. However, even with the 8-CCL algorithm, the labeling performance could not be 100% accurate. For instance, the two objects (one in light blue and the other in blue) at the bottom in the processed image are supposed to be labeled as one object. However, due to the un-purposely disconnected lines created by either the blurry display of the initial original image or the inappropriate thresholding approach, labeling them as a single object was not feasible. In order to rectify the issue, all the thresholding values (0-31) for the global thresholding and adaptive thresholding approaches were attempted but we could not connect the lines, thus concluded that changing the values of the specific pixels manually would solve the issue.

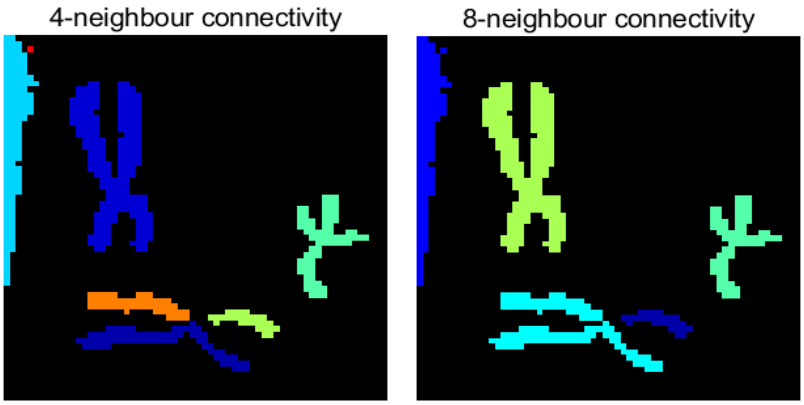

Figure 13. Labeled images created by 4-CCL (left) and 8-CCL (right) methods

3.5.2. 4- and 8-CCL on Image 2

The segmentation based on CCL algorithms was performed successfully labeling six individual objects (1, 2, 3, A, B, and C) as shown in **Figure 14**. As the object lines in the binary image were all grouped properly without disconnected lines, the algorithms were able to identify correct number of the objects as opposed to the one in the image 1. The labeled objects were then used to perform the subsequent tasks that will be discussed in the following sections.

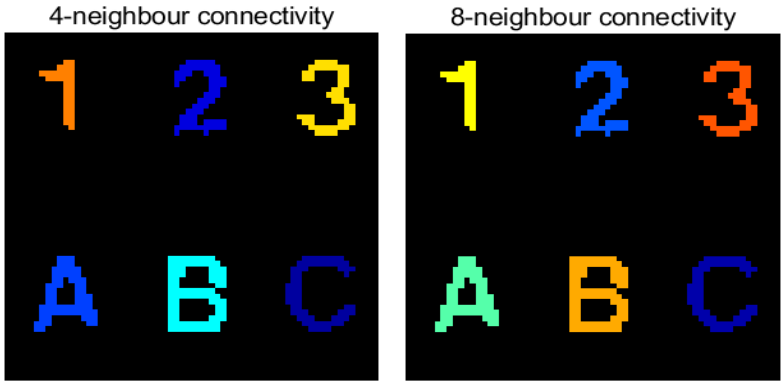

Figure 14. Labeled images created by 4-CCL (left) and 8-CCL (right) methods

3.6. Rearrangement of Image 2

The height and width of each character was measured based on the labeled values obtained in **Section 3.5**. The bounding box approach was successfully utilized to segment the objects to separate. However, when they were initially arranged in the appropriate order (A, B, 1, 2, 3, C), they looked rather compact. Hence a pixel thick column was added between each character for a better visualization as shown in **Figure 15**. The bounding box code written by our group presented several disadvantages. Since most of the coordinates were found manually and only a few of them could be computed automatically, it was not possible to apply the code to other images. A better approach would have produced a more efficient and clean code.

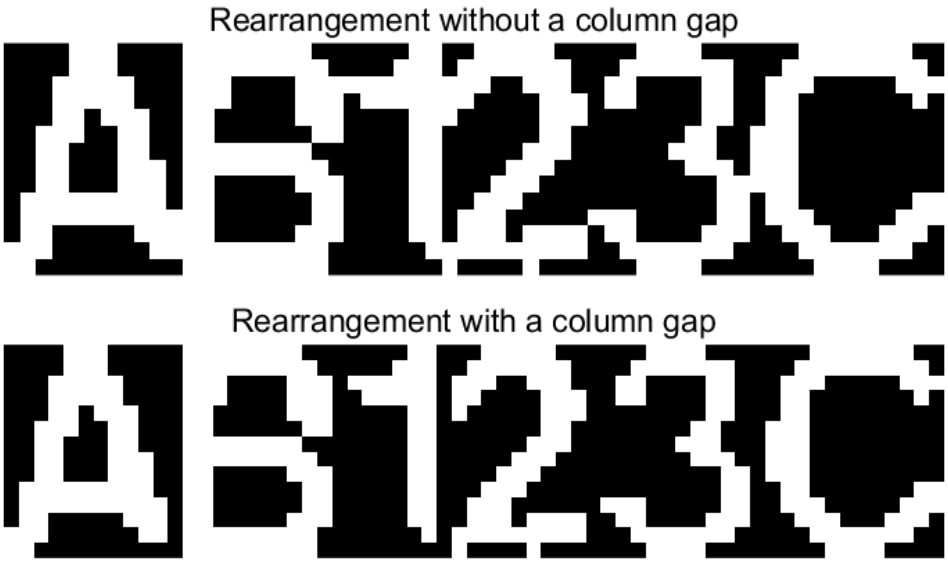

Figure 15. Rearranged Images for the image 2

3.7. Rotation of Image 2 by 30 degrees

Figure 16 shows the output images rotated about its center of the previous image. Although the objects look rather distorted, they still seem to hold their main features, and we concluded that the image is visually representable and does not need further refining process. This is because, when compared with the use of the built-in function "*Imrotate()"*, our image did not present any inferior features.

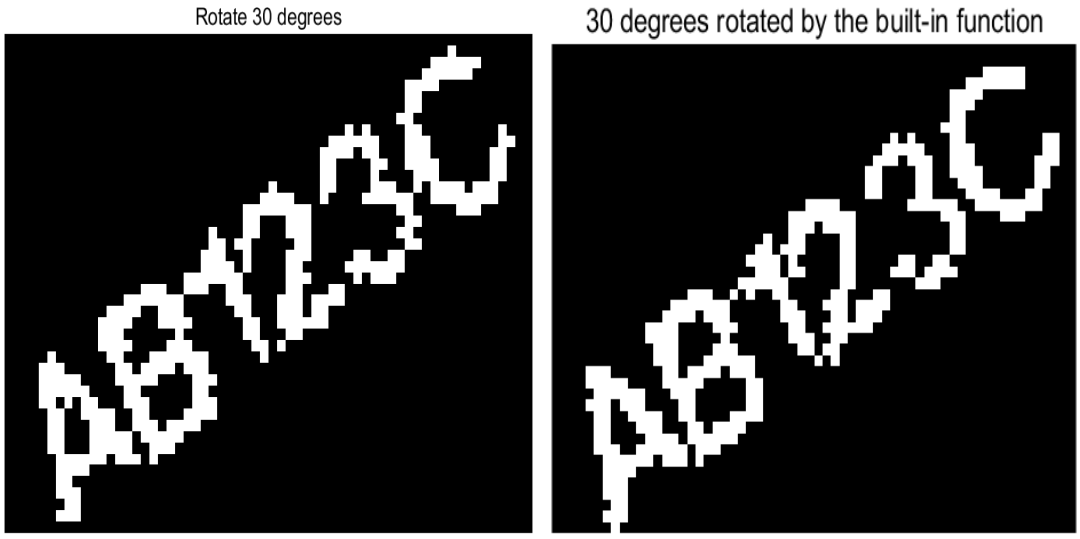

Figure 16. 30° Rotated Images by our code (left) and built-in function (right)

3.7. Number Recognition

Accuracy varies according to the change of K value. Accuracy of the prediction is defined as

$$accuracy = \frac{The\ number\ of\ predictions\ that\ are\ consistent\ with\ the\ real\ results}{Total\ number\ of\ prediction}$$

K is the top K training pictures with the smallest gap with the test picture.

The relationship between K and accuracy is shown in the following graph. In this scenario, th e accuracy is highest when K is 1.

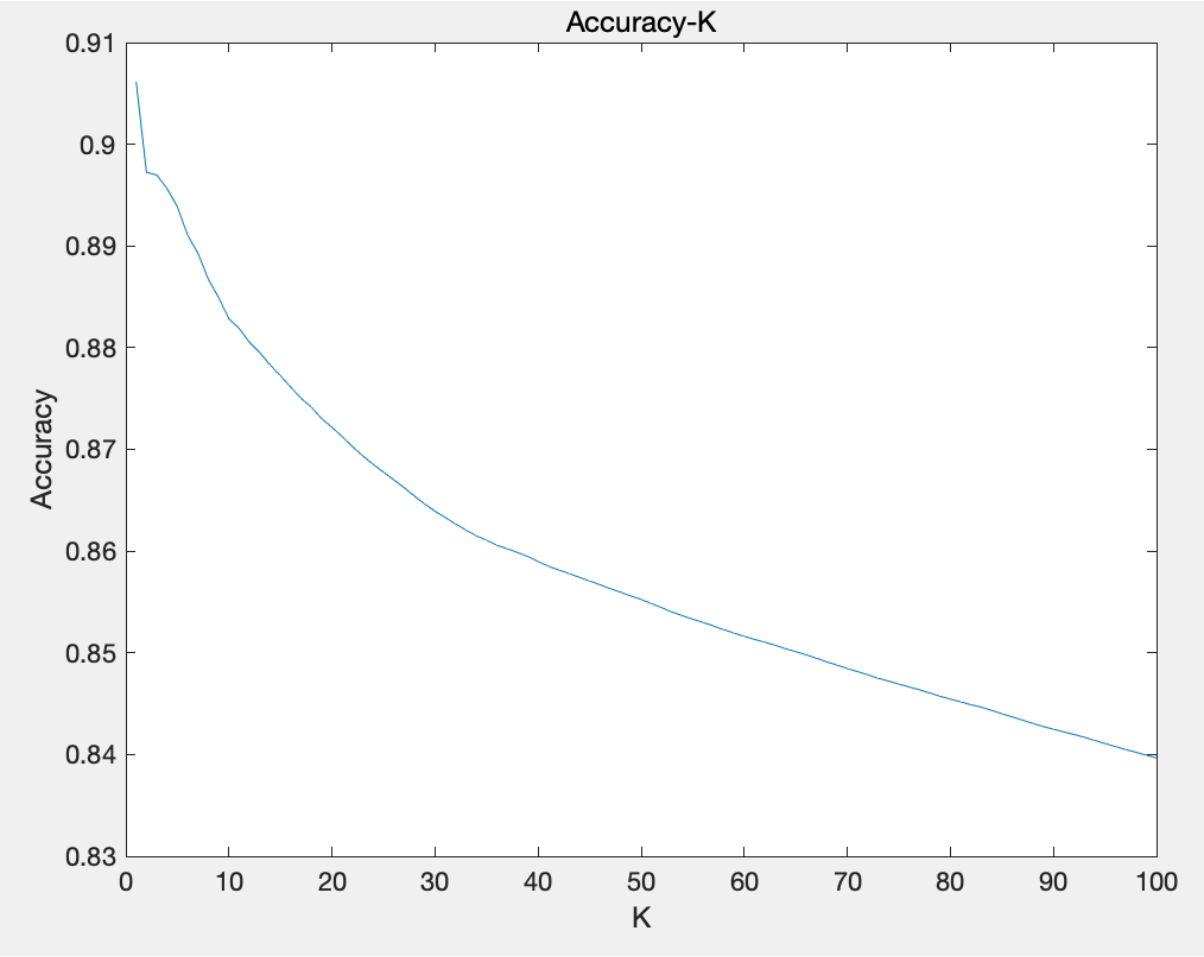

Figure 17. Relationship between K and accuracy

4. Conclusion

In this project, multiple technical operations and basic knowledges of image processing were implemented to complete a series of tasks related to machine vision applications. As can be seen in the introduction part, most of the procedures were performed in the same manner for both images, except for the image segmentation, rotation and dataset training parts, which were only conducted for the image 2.

For the opening part, the txt files were comprised of numbers (0-9) and capital letters (A-V) and each character was designated corresponding pixels to perform the image display successfully. Determining a threshold value was a technical process because, depending on the value, it could result in noise or loss of information. Four threshold methods were employed to produce binary images. By optimizing the thresholding value, the binary images with best quality were produced. As for the outline part, the *Gaussian* filtering was initially applied on the images before the Sobel operator. Then a new approach similar to the non-maximum suppression method was utilized. As a result, the images were created with better outlines as compared to solely using the Sobel operator. Compared to the 4-CCL method, the 8-CCL algorithm worked more efficiently on identifying and labeling the objects. For the segmentation of image 2, the bonding box approach was utilized according to the heights and widths of characters measured previously, and the segmented objects were re-arranged accordingly. The rotation was realized based on the transform matrix. Some features looked rather imperfect when the image was rotated. However, we considered it to be still visually representable since it did not look too different from the other rotated image created by the built-in function. For the machine learning part, KNN method was employed for the object recognition performance, which showed an excellent test accuracy ranging from approximately 84 to 90 percent.

Throughout the project, the understanding of the course content has been deepened and the interest in a further investigation about machine vision was stimulated. We all appreciate having the opportunity to take this lecture.

5. Appendix

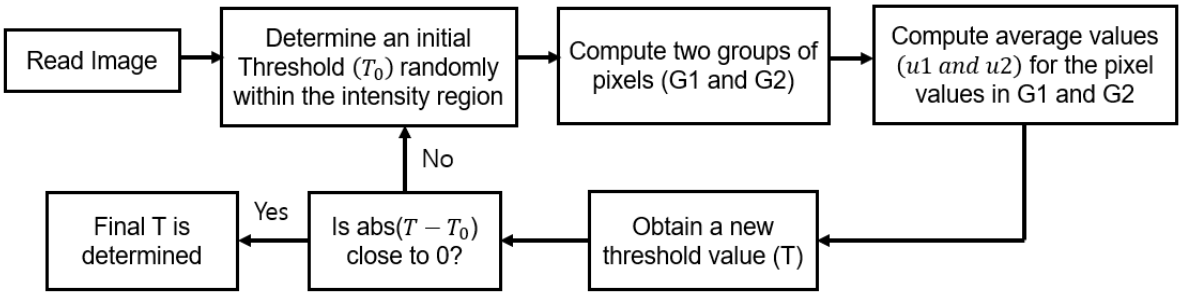

Figure 18. Basic Global Thresholding Flowchart

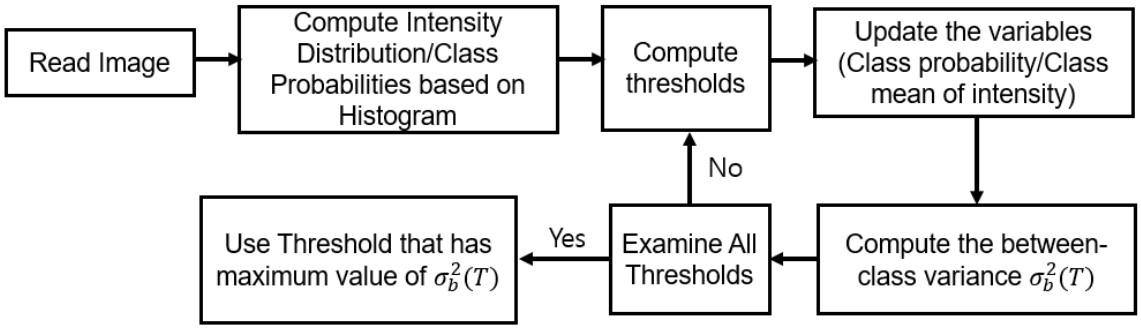

Figure 19. Otsu's Method Flowchart

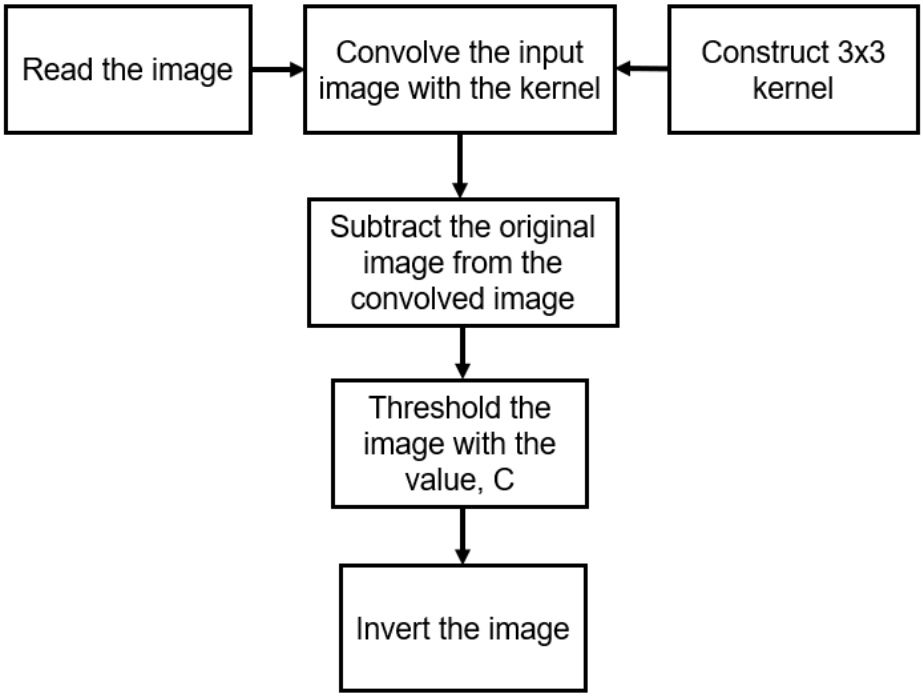

Figure 20. Adaptive Thresholding Flowchart

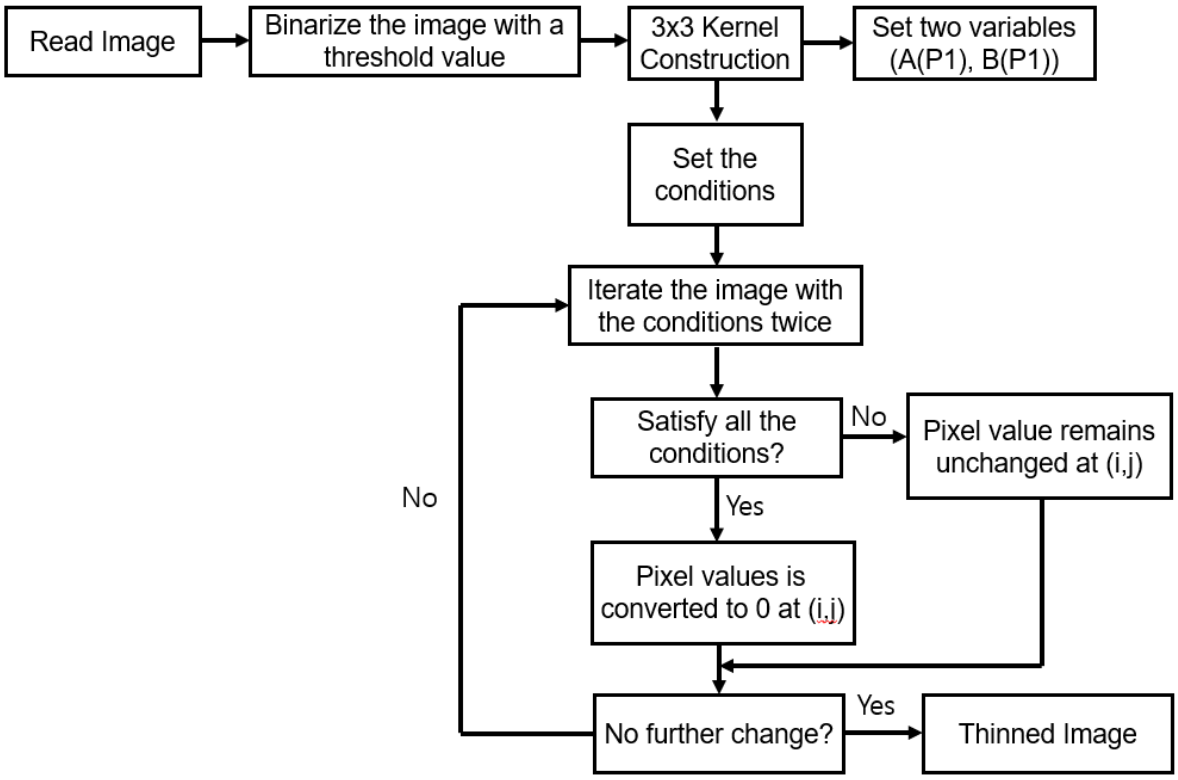

Figure 21. Zhang-Suen Thinning Algorithm Flowchart

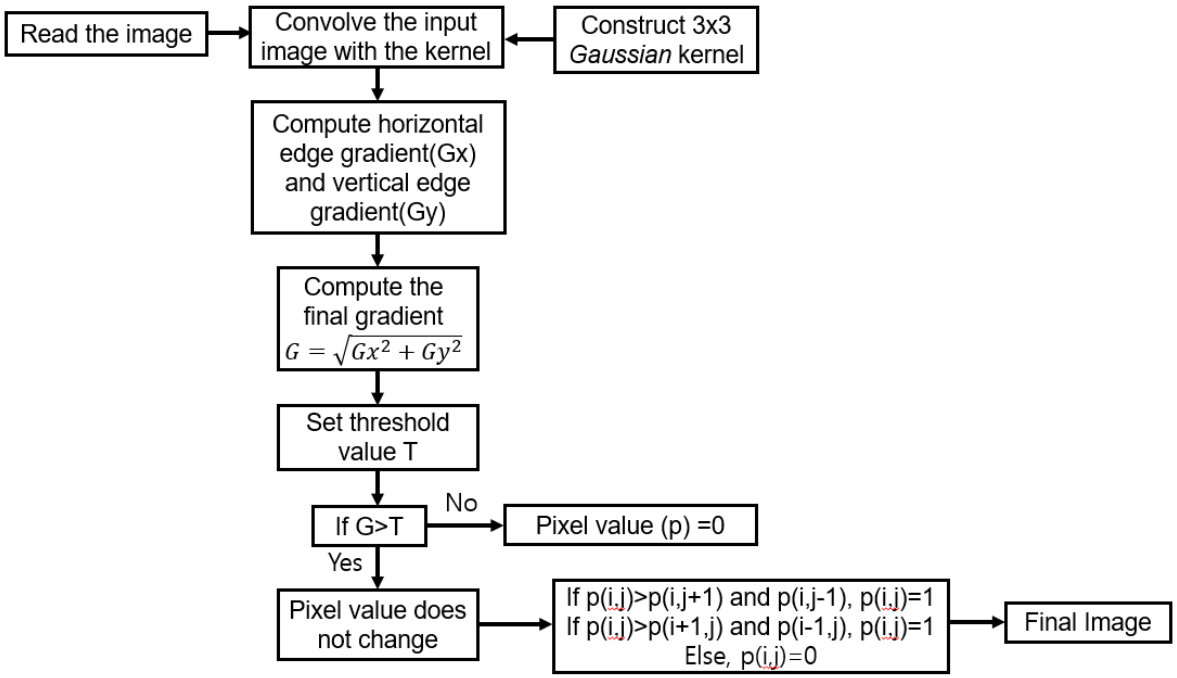

Figure 22. Sobel Operator with additional approaches Flowchart

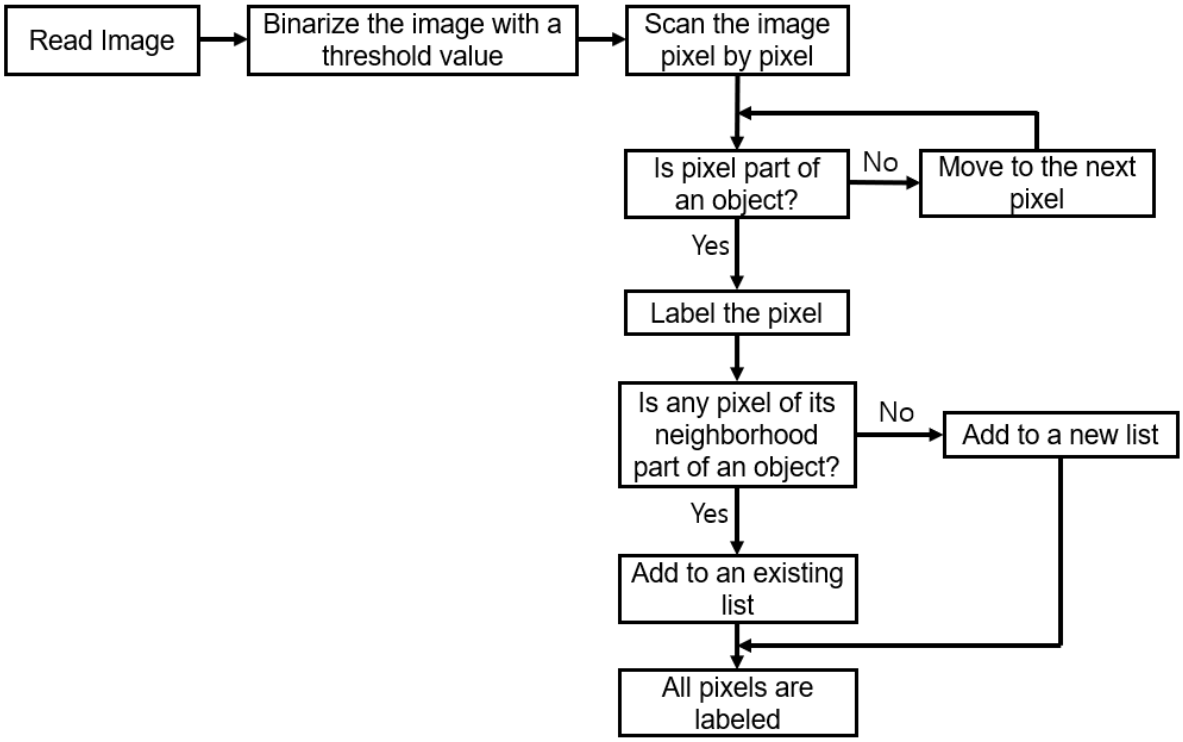

Figure 23. Connected Component Labeling Flowchart

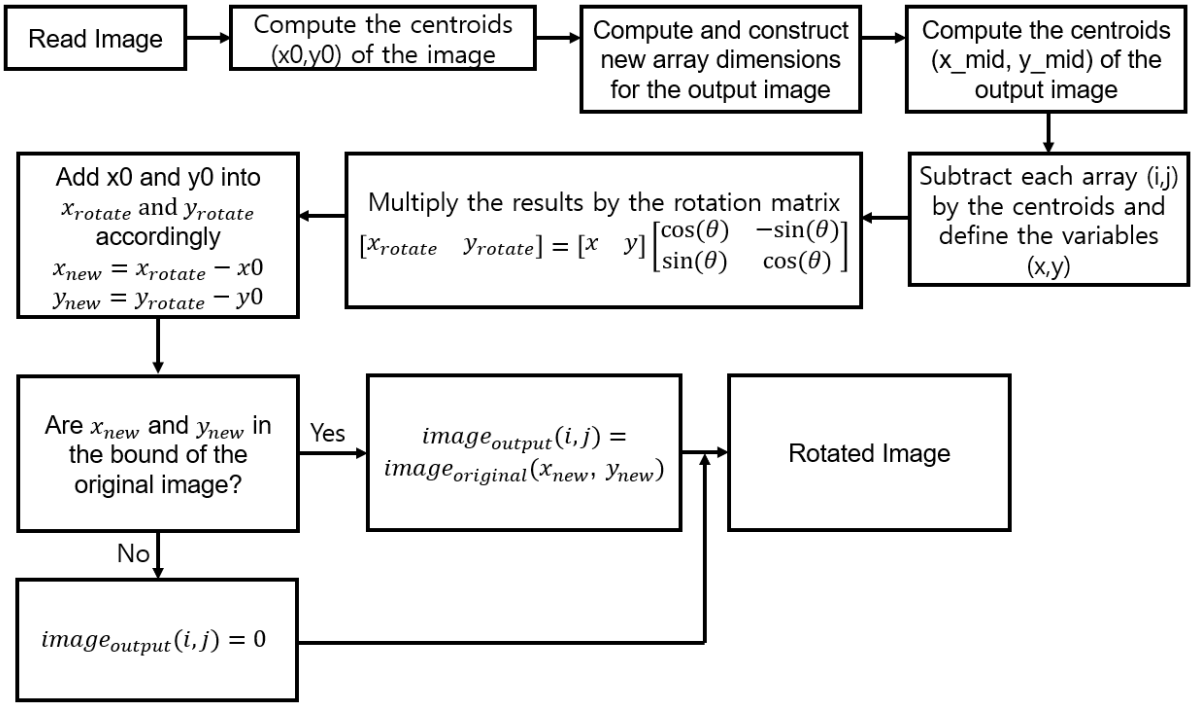

Figure 24. Rotation Flowchart

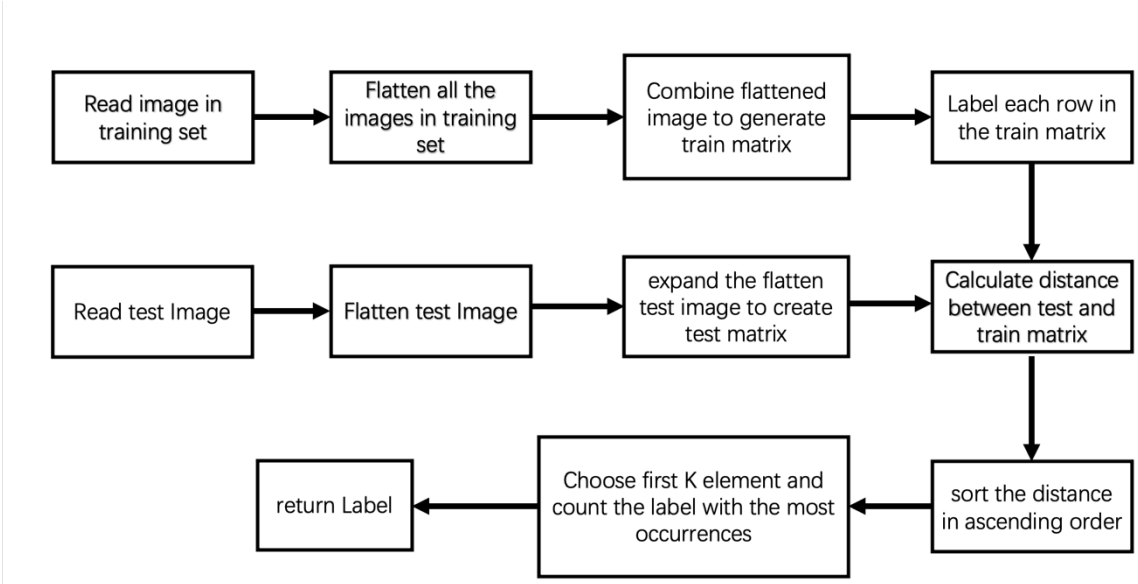

Figure 25. KNN Flowchart