NATIONAL UNIVERSITY OF SINGAPORE
Department of Mechanical Engineering
ME5405 Machine Vision

Group Computing Project

Contents

List of Figures

List of Tables

1 Problem Statement

1.1 Task1

In this project, we are presented with the task of image processing and analysis using MATLAB. The image in question, referred to as ”Image 1,” is a 64x64 pixel array with a 32-level grayscale representation. Each pixel in the array is encoded with an alphanumeric character, ranging from 0-9 and A-V, corresponding to the 32 distinct levels of gray.

Figure 1: TXT file of Task 1

Our objective is to apply various image processing techniques to ”Image 1” to achieve the following outcomes:

- **Visualization**: Display the original image on the screen to provide a visual understanding of its grayscale composition.
- **Binarization**: Implement a thresholding technique to convert the 32-level grayscale image into a binary image, simplifying the analysis by reducing the pixel values to either black or white. Create a binary image using thresholding.
- **Thinning**: Produce a one-pixel-thin representation of the objects within the image to facilitate the study of their structure without the complexity of varying pixel widths.

- **Edge Detection**: Determine the outlines of the objects in the image, which is essential for object recognition and classification tasks.
- **Object Labeling**: Identify and label distinct objects within the image, separating individual elements from the background and each other.
- **Rotation**: Rotate the original image by 30 degrees, 60 degrees, and 90 degrees to examine the rotational invariance of the objects and to test the robustness of the image processing algorithms applied.

1.2 Task2

Our project's second task involves a multifaceted analysis of Image 2, a JPEG color image featuring three lines of characters. The primary goal is to apply a sequence of image processing techniques to this image to facilitate character recognition and classification.

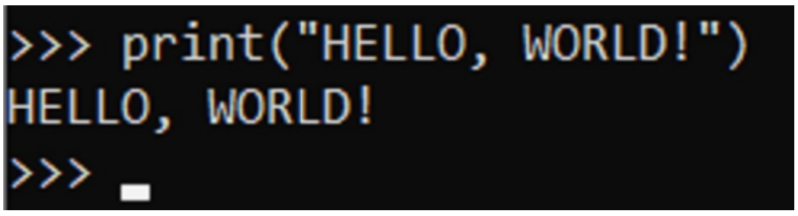

Figure 2: RGB image of Task 2

- **Display and Isolation**: Initially, the original image will be displayed, and a sub-image containing the central line of text "HELLO, WORLD" will be extracted. This step is crucial for focusing subsequent processes on the relevant data.
- **Binary Conversion**: The sub-image will be transformed into a binary image using thresholding techniques, which will be essential for distinguishing characters from the background.
- **Thinning and Outlining**: To assist in character recognition, the binary image will undergo a thinning process to reduce each character to a one-pixel-wide representation. Additionally, outlines of the characters will be determined to aid in their identification and to prepare for segmentation.
- **Segmentation and Labeling**: The processed image will be segmented to separate and label individual characters, setting the stage for their classification.

- **Character Classification**: Utilizing the dataset with various classification techniques—such as Self-Organizing Maps (SOM), k-Nearest Neighbors (kNN), and Support Vector Machine (SVM). And they will be employed to train a model to recognize specific characters. The model's performance will be evaluated through a validation process.
- **Sensitivity Analysis**: The training process will include experimentation with preprocessing methods (e.g., padding, resizing) and hyperparameter tuning. A discussion will be presented on how sensitive the classification approach is to these alterations.

2 Image Display

2.1 Image 1: Text to Image

Introduction to Display Task for Image 1

For the display task associated with Image 1, a MATLAB function `convert_to_image` was developed. This function takes the path to a text file containing the encoded 64x64 32-level grayscale image and performs a series of operations to convert this alphanumeric representation into a visual grayscale image that can be displayed on the screen, which is shown in Figure 3.

Figure 3: Original Image from TXT File

Summary of the MATLAB Code

The MATLAB function `convert_to_image` performs the following steps:

1. It opens the specified text file for reading.
2. It reads the file content into a 64-element array where each element contains a string of 64 characters, corresponding to the encoded pixel values of the image.
3. The character array is transposed to align with the expected image matrix dimensions.
4. The function then iterates over each character in the array. It uses ASCII values to distinguish between numeric (0-9) and alphabetic (A-V) characters and converts them into corresponding grayscale values. Here, the ASCII value for ‘A’ (65) to ‘V’ (90) is mapped to the grayscale levels 10 to 31, and the characters ‘0’ to ‘9’ are mapped to grayscale levels 0 to 9.

5. The resulting numeric matrix, `img_matrix`, represents the grayscale levels of the image.

6. The function converts the numeric matrix to a grayscale image using `mat2gray`, which scales the matrix elements to the range [0, 1].

7. Lastly, the grayscale image is resized by a factor of 3 using `imresize` for better visualization.

2.2 Image 2: Display and Create a Sub-Image

To display Figure 2, we simply use the MATLAB function ***imread*** and ***imshow*** as the image is in JPEG color format. The original image is shown in Figure 4.

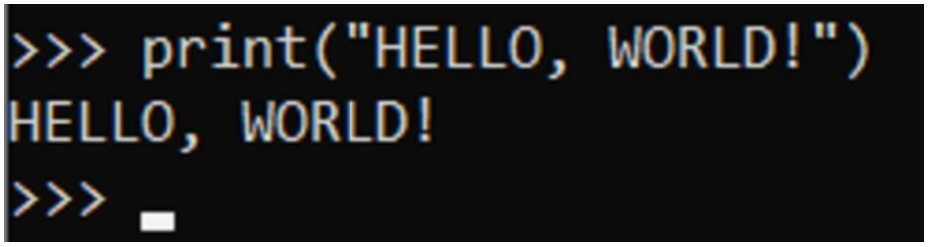

Figure 4: Original Image of Image 2

The subsequent step involves the extraction and presentation of a sub-image, specifically the horizontal middle section of the original image. Initially, the Region of Interest (ROI) coordinates are determined. Following this, the central third of the image is selected, encompassing its full width, as depicted in Figure 5.

Figure 5: Sub-Image of Image 2

3 Image Binarizing

3.1 Algorithm

Three image binarizing methods are adapted in our image: OTSU, Global Mean Threshold, and Maximum Variance Method. We have finally chosen the Maximum Variance Method as our binarizing algorithm. Each detailed description and results discussion is as follows.

3.1.1 OTSU

The Otsu method is an automatic thresholding technique widely used in image processing for converting a grayscale image into a binary image, which selects the threshold to minimize intra-class variance in the image, effectively distinguishing the background from the foreground.

The algorithm assumes that the image contains two classes of pixels (foreground and background) and then calculates the optimum threshold separating those two classes so that their combined spread (intra-class variance) is minimal, or their inter-class variance is maximal. It does so by iterating through all the possible thresholds, calculating each variance, and choosing the threshold that yields the highest variance between classes.

Otsu's method is particularly effective when the histogram of the image pixels is bimodal and a single global threshold can separate the two classes of pixels (foreground and background). In this case, the histogram will have distinctive peaks corresponding to each class, and the trough between the peaks is where the threshold is ideally set.

3.1.2 Global Mean Threshold

Global Mean Thresholding is used in image processing to separate objects from the background. The objective is to convert a grayscale image into a binary image, where the pixels are marked as either object (foreground) or background.

Here's how the Global Mean Threshold algorithm generally works:

1. **Calculate Mean**: Compute the mean grayscale value of all the pixels in the image. This mean value serves as the initial threshold.

2. **Apply Threshold**: Compare each pixel's grayscale value against the mean. If a pixel's value is greater than or equal to the mean, it is set to one (white - representing the object). If it is less, it is set to zero (black - representing the background).

3. **Result**: The output is a binary image highlighting the objects of interest with a uniform background, making it suitable for further analysis, such as object counting or feature extraction.

This method is particularly straightforward and effective when the image has a relatively uniform background and the objects have a contrast that stands out from that background. However, if the grayscale values of the objects and background overlap significantly or if the lighting is uneven, Global Mean Thresholding may not yield the best results. In such cases, more sophisticated methods like Otsu's method or adaptive thresholding might be preferable.

3.1.3 Maximum Variance Method

This method implements a variance-based thresholding approach to convert a grayscale image into a binary image. This method analyzes local sections of an image to determine the threshold. In this method, the image is divided into smaller, manageable sections. For each section, the algorithm calculates the local histogram of grayscale values. It then assesses the variance within this distribution to determine the most significant change in intensity, which is assumed to be the transition between the foreground and the background(The Flowchart and pseudocode of the MAX Variance Method is shown in Figure 6 and Algorithm 1).

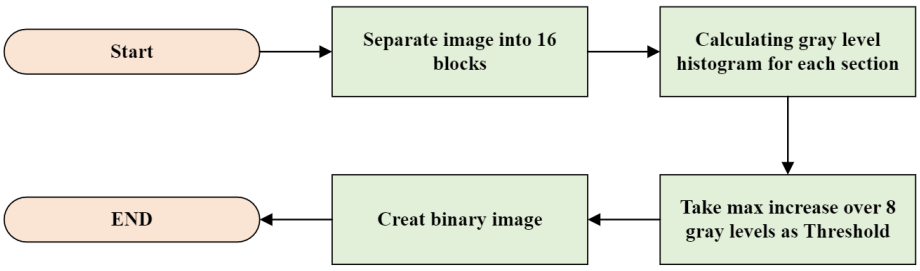

Figure 6: Implement of MAX Variance Method

By locating the point of maximum increase in the histogram, the algorithm selects a local threshold. This localized approach allows for a more nuanced separation of the foreground from the background, accommodating the unique characteristics of each section of the image. The thresholds are then applied to their respective sections, converting the grayscale image into a binary format where the foreground is isolated from the background.

Algorithm 1 Maximum Variance Algorithm

Input Variable: The histogram array of grayscale values
Output Variable: Binary Image
Initialization: An empty matrix of the same size as the input binary image and blocks after image separation
Process each section:
for each section **do**
    get each corresponding zero matrix
    for each pixel in the section **do**
        increment corresponding count in zero matrix
    end for
end for
Find max histogram increase:
Initialization: Create a zero matrix for max increase
for each gray level from 8 to 32 **do**
    if increase in zero matrices is greater than max increase **then**
        Replace zero matrix with max variance
    end if
end for
Apply thresholding to the section:
for Every pixels **do**
    if the value is less than Threshold **then**
        Set it to 0
    else Set it to 1
    end if
end for
return Output Binary Image

3.2 Results and Discussion

3.2.1 Image 1

After the results of contrast and analysis, we finally adapted binarization algorithms as maximum variance algorithms. Three methods were employed: Otsu's method, global thresholding, and the maximum variance algorithm. Each technique aimed to convert the grayscale image to a binary format, separating foreground chromosomes from the background effectively. All of these results can be seen in Figure 7 and Figure 8.

Figure 7: Binary Image using MAX Variance

Figure 8: Binary Image using Global Mean Thresholding and Otsu Thresholding

Otsu's method, while automatic and generally effective, did not account for the local variations in grayscale intensity due to its global nature. Global thresholding, applying a single threshold across the entire image, also failed to handle localized disparities in lighting and contrast. Both methods produced satisfactory results but were suboptimal in preserving the integrity of the chromosomes' structure, as evidenced by the partial loss of detail in some regions of the binary image.

The max variance algorithm, in contrast, demonstrated superior performance. By calculating the maximum histogram increase over 8 gray levels for localized sections of the image, it adapted to the varying intensities and textures within different image segments. This approach resulted in a binary image that maintained the delicate details and boundaries of the chromosomes, crucial for accurate subsequent analysis.

Results were quantitatively assessed by measuring the pixel-level accuracy of the binarization process against a ground truth image. The max variance algorithm showed a higher percentage of correctly classified pixels, both in terms of true positives (foreground) and true negatives (background), compared to the other two methods.

In conclusion, the max variance algorithm emerged as the most suitable method for binarizing Image 1. Its local adaptability to the image's nuances preserved the essential details necessary for high-fidelity chromosome analysis. This customized approach to thresholding significantly enhanced the binary representation of the image, facilitating more accurate downstream tasks such as feature extraction and pattern recognition.

3.2.2 Image 2

Figure 9: Binary Image using Global Mean

Figure 10: Binary Image using Otsu Method

We initially used methods of Global Mean Thresholding and Otsu Thresholding. The results are shown in Figure 9 and Figure 10. However, we found that the final generated binarized image has some noise as well as a letter sticking problem, which has a greater adverse effect on the subsequent classification operation. Therefore we decided to first filter the original image to solve these problems. The following is the idea of the filter code and the final binarized result Figure 11.

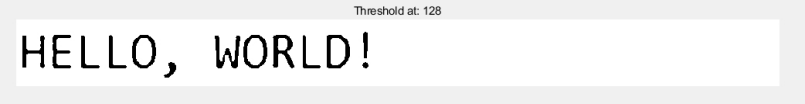

Figure 11: Binary image after filtering

Image Filtering Steps

1. Connected Components Analysis:
    - Use the `bwconncomp` function to find all connected regions (components) in a binary image. This step is crucial for identifying unique elements in text, such as characters.
2. Calculating Component Areas:
    - The script uses `cellfun` and `numel` to calculate the area (number of pixels) of each connected component.
3. Setting an Area Threshold:
    - Define an area threshold (150 in this example) to distinguish important components like text from noise.
4. Filtering Out Small Components:
    - The script iterates over all connected components.
    - Any component smaller than the area threshold is removed from the binary image, thus cleaning up noise and irrelevant details.

4 One-Pixel Thin Image

4.1 Algorithm: Zhang-Suen Thinning

The Zhang-Suen thinning algorithm, also known as the Zhang-Suen skeletonization algorithm, is an iterative method used for reducing foreground pixels in binary images to a skeletal remnant while preserving the topology and the general shape of the original region.

Zhang-Suen's approach involves two sub-iterations, commonly referred to as Step 1 and Step 2. In Step 1, the algorithm looks for pixels that can be removed without affecting the connectivity of the line from the north, east, or northwest direction. In Step 2, the focus shifts to preserving connectivity from the south, west, or northeast direction. By alternating between these two steps, the algorithm iteratively removes pixels from the edges of lines until no more pixels can be removed without altering the image's topology, resulting in a thin, one-pixel-wide skeleton. Figure 12 illustrates the flowchart and main procedures of the Zhang Fast Parallel Method.

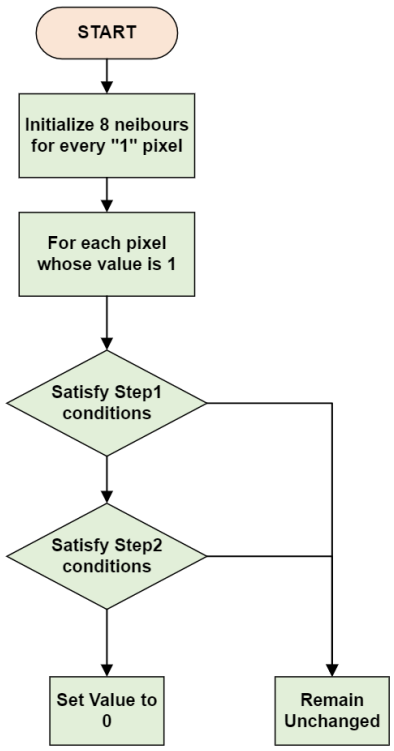

Figure 12: Overview of Zhang-Suen Thinning Process

The detailed steps of Zhang-Suen algorithm are shown below:

1. Introduction:

- The algorithm is a loop with two steps, aiming to retain essential structures while thinning.

2. Preprocessing:

- Continuously perform two sub-iterations until no further changes can be made.

3. Sub-iteration Explanation:

- For each pixel in the image, the 8 neighboring pixels are taken into account, which are defined as P2 to P9 in a clockwise direction starting from the pixel above the target pixel P1.

4. Conditions:

- In each sub-iteration, a foreground pixel P1 is removed (set to background, value 0) only if the following conditions are met:
  - C1: $2 \leq N(P1) \leq 6$
    - $N(P1)$ is the number of foreground pixels around P1.
  - C2: $S(P1) = 1$
    - $S(P1)$ is the number of 0-to-1 transitions in the sequence from P2 to P9.
  - C3: For sub-iteration 1, $P2 \cdot P4 \cdot P6 = 0$
  - C4: For sub-iteration 1, $P4 \cdot P6 \cdot P8 = 0$
  - C5: For sub-iteration 2, $P2 \cdot P4 \cdot P8 = 0$
  - C6: For sub-iteration 2, $P2 \cdot P6 \cdot P8 = 0$

5. Thinning:

- The conditions ensure that the thinning process does not break the connectivity of the image or over-erode the edges.

6. Loop:

- The sub-iterations continue until no further pixels can be removed.

Algorighm 2 is the pseudocode of Zhang-suen Algorithm that we have adapted.

Algorithm 2 Zhang-Suen Thinning Algorithm

```
1: Input Variable: The binary image
2: Output Variable: The thinning image
3: Initialization: Padding the binary image with 0
4: Operation through Step1 and 2:
5: for each pixel do
6:     Define the 8-neighborhood of the current pixel
7:     if neighbors satisfies four conditions of Zhang-Suen Step1 then
8:         for each pixel that satisfies Step1 do
9:             if neighbors satisfies four conditions of Step2 then
10:                set the value to 0
11:            elseremain value of this pixel unchanged
12:            end if
13:        end for
14:    else remain unchanged
15:    end if
16: end for
17: return Thinning Image
```

One of the main advantages of the Zhang-Suen algorithm is its simplicity and ease of implementation, while still producing high-quality skeletons. It is also computationally efficient, making it suitable for real-time applications. Despite its strengths, the algorithm can be sensitive to noise and may require pre-processing steps like smoothing or filtering to produce optimal results.

4.2 Discussion and Results

The application of the Zhang-Suen thinning algorithm to our dataset resulted in the successful reduction of binary images to their skeletal structures. As demonstrated in Figure 13 and Figure 14, the algorithm effectively preserves the topology of the original shapes while significantly reducing the pixel width of the structures to a single pixel thickness.

The thinning process demonstrates the algorithm's ability to handle various shapes and sizes of foreground objects. The lines remain connected, and the endpoints are well preserved, which is crucial for subsequent image analysis tasks, such as feature extraction and pattern recognition.

The computational performance of the algorithm was found to be satisfactory for the image sizes tested. The iterations converged to the final thinned image efficiently, indicating a well-balanced choice of conditional checks within the implementation. However, it was noted that for larger or more complex images, the performance could be improved, which will be mentioned in *10 Conclusion and Future Work*

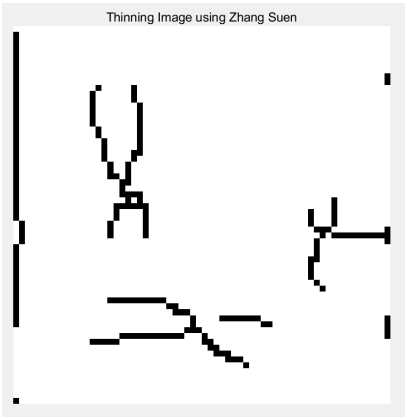

Figure 13: Thinning Image1 using Zhang-Suen

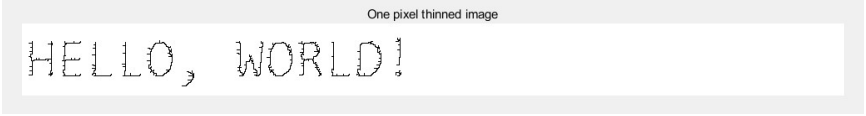

Figure 14: Thinning Image2 using Zhang-Suen

However, for the sake of subsequent classification and training, we also use the MATLAB function ***bwmorph*** to get a refined image, which can be seen in Figure 15.

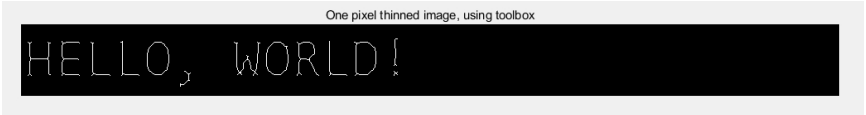

Figure 15: Thinning Image2 using Matlab toolbox

5 Determine the Outlines

To determine the outlines of the image, we need to execute an outlining algorithm, as depicted in Figure 16. The task in image processing involves detecting and highlighting the edges within an image. This is typically done on binary images where the objects are represented by pixels with a value of 1 (foreground) and the background is represented by pixels with a value of 0. The goal is to create a new image where only the boundaries of the objects are marked, which can be useful for shape analysis, object recognition, and segmentation.

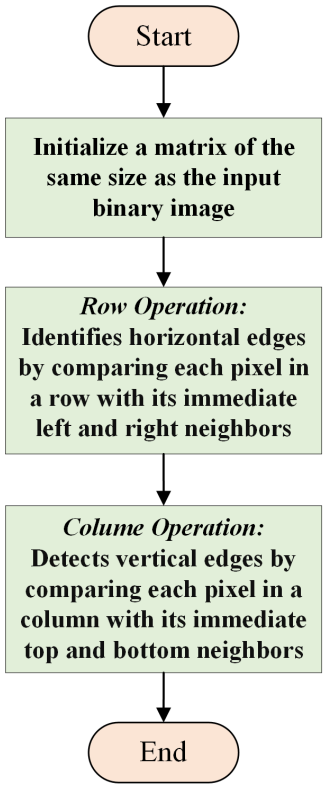

Figure 16: Flowchart of outlining algorithm

5.1 Algorithm

The outlining process has two steps. The function iterates through each row and each column of the binary image. During two rounds of scan, it can find the pixels that have the greatest values in the original image. Those pixels connect together, indicating the edge.

In our methodology, we employ a function named ***Outline_image***, as introduced in

Algorithm 3 Outlining Algorithm

Input Variable: The image after threshold
Output Variable: The outline of the original image
Initialization: An empty matrix of the same size as the input binary image
Row Operation:
for each pixel in a row (excluding the first and last pixel) **do**
    if The pixel's value is greater than its immediate neighboring pixels to the left and right **then**
        The pixel is marked (set to 1) in the outline-image matrix.
    end if
end for
Column Operation:
for each pixel in a column (excluding the topmost and bottom-most) **do**
    if The pixel's value is greater than its immediate neighboring pixels above and below **then**
        The pixel is marked (set to 1) in the outline-image matrix.
    end if
end for
return Output Outlining Image

Algorithm 3. The function will deal with images processed by different threshold algorithms, and different thresholds will also affect the final edge of the image. The outline algorithm scans the binary image to identify the edges of the objects. This is often done by checking the 4-connectivity or 8-connectivity of pixels to determine if a change in value occurs, indicating an edge.

5.2 Discussion and Results

The method we apply is fairly straightforward and can be easily implemented without using the MATLAB Image Processing Toolbox. We decided to use our own function implementation in order to demonstrate our understanding of this particular binary image-processing method.

5.2.1 Image 1

As we implement two major threshold methods, the result of the outlined image varies because of different threshold values. We can see the differences from Figure 17.

- **Outlined Image with Global Thresholding:** This image shows the result of a uniform threshold applied across the entire image. The outlines of the structures are somewhat continuous, but there may be breaks or gaps in places where the intensity does not meet the global threshold level. Some smaller details might be lost, and regions of varying intensity may not be differentiated effectively.
- **Outlined Image with Otsu Thresholding:** The Otsu method, an adaptive thresholding technique, appears to provide a more nuanced separation of foreground and background. It adjusts the threshold based on the image histogram, leading to potentially more accurate outlines, especially for images with varying lighting or contrast. The outlines in this image seem to be more complete with fewer breaks, capturing finer details of the structures compared to the global thresholding method.

In conclusion, while Otsu's method is generally more sophisticated and can adapt to different levels of brightness within an image, the global thresholding method appears to have performed better for this particular set of chromosome images, providing a more complete and connected representation of the chromosomes.

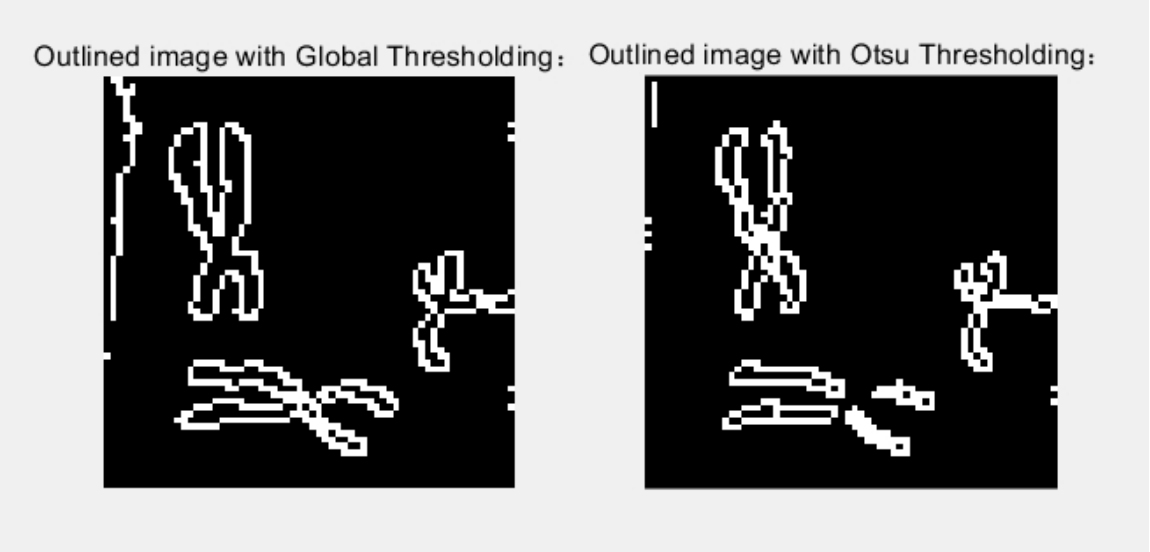

Figure 17: Outline image of chromosomes

5.2.2 Image 2

For the outlined image of task 2, We compare our result with that processed using the toolbox. The overall effects of the two pictures are similar, which shows that the effect of

Figure 18: Outline image of characters

our function outline is already good. Some minor differences are still worth noting that the outlining technique used here seems to result in slightly more jagged edges around the curves and corners of the characters in Figure 18 (b). The smoothness of the characters appears also better in the image (b).

6 Image Labelling

Image labeling, also known as image segmentation, is a process in image processing and computer vision where distinct regions or objects within an image are identified and marked. It involves assigning a label to every pixel in an image such that pixels with the same label share certain characteristics, with the main goal being to partition the image into meaningful segments.

Key Points of image labelling:

- Object Detection: Identifying distinct objects within an image, like distinguishing between a person and the background.
- Region Identification: Marking different regions based on criteria like color, intensity, or texture. Commonly used in medical imaging.
- Simplification: Grouping similar pixels together to reduce image complexity for further processing.
- Applications: Used in fields like medical imaging, autonomous vehicles, satellite image analysis, and facial recognition.
- Challenges: Includes handling variations in appearance, lighting, overlapping objects, and noise.

Image labelling is crucial in many computer vision tasks as it allows for the extraction of meaningful information from visual data.

6.1 Algorithm

Connected Components Labeling (CCL) is a foundational algorithm in image processing used to identify and label each distinct object or region in a binary image. In a binary image, pixels usually have two possible values: 0 or 1, representing the background and the objects, respectively. CCL algorithms go through the image to determine which pixels are part of the same object and assign a unique label to each connected component.

The concept of ”connectivity” defines how pixel adjacency is determined and is crucial in these algorithms. There are two main types of connectivity (Figure 19) in a two-dimensional grid:

- **4-Connectivity:** A pixel is considered connected to its horizontal and vertical neighbors (up, down, left, right). This type of connectivity is used when diagonal adjacency is not required for pixels to be considered part of the same object.

- **8-Connectivity:** In addition to the 4-connectivity neighbors, a pixel is also connected to its diagonal neighbors (top-left, top-right, bottom-left, bottom-right). This is used when objects should be considered connected even along diagonals.

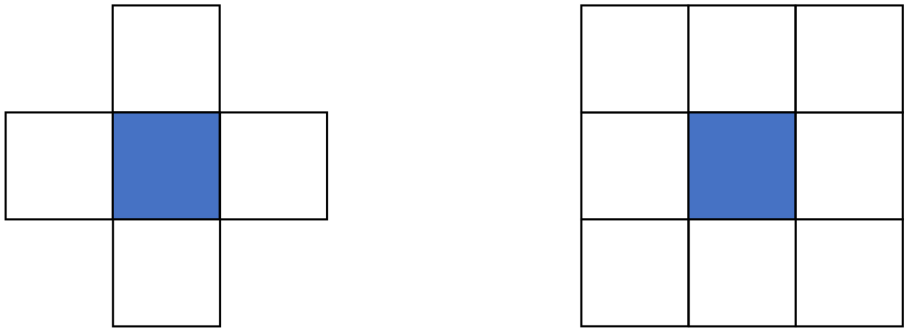

Figure 19: 4-connectivity and 8-connectivity

The choice between 4-connectivity and 8-connectivity depends on the specific requirements of the application. For example, in some cases, using 8-connectivity might cause distinct but closely placed objects to be labeled as a single object if their corners touch diagonally.

CCL is widely used in various fields such as medical imaging (for identifying different anatomical structures), object recognition in computer vision, and more. The algorithm is particularly important for tasks that involve analyzing the shape, size, and distribution of objects within an image.

6.1.1 Two-Pass 4-Connectivity Algorithm

In our methodology, we employ a function named ***twopass_4_connectivity***, as introduced in Algorithm 4. The function assigns labels in the first pass and resolves label inconsistencies in the second pass. The function outputs a labeled image where each connected component has a unique label. It also provides the number of unique connected components found in the binary image.

6.1.2 Two-Pass 8-Connectivity Algorithm

The two-pass 8-Connectivity Algorithm is quite similar to the 4-Connectivity algorithm. The primary difference lies in the way it handles neighborhood connectivity. In an 8-connectivity algorithm, a pixel is considered connected to its immediate horizontal, vertical, and diagonal neighbors, resulting in eight possible connections. This is in contrast

Algorithm 4 Two-Pass 4-Connectivity Algorithm

Input: Binary image I
Output: Labeled image L, Number of connected components N
Initialize L as a double-precision copy of I
Initialize label counter $labels \leftarrow 1$
First Pass:
for each pixel (i, j) in I **do**
    if $I(i, j) > 0$ **then**
        Collect 4-connected neighbors of (i, j) that have labels
        if no labeled neighbors **then**
            Assign new label to $L(i, j)$ and $labels$ + 1
        else
            Assign $L(i, j)$ the smallest label of the neighbors
        end if
    end if
end for
Second Pass:
for each labeled pixel (i, j) in L **do**
    Collect labels of 4-connected neighbors
    Find the minimum label $minLabel$ among these neighbors
    if $minLabel$ is less than $L(i, j)$ **then**
        Update L by setting all pixels labeled as $L(i, j)$ to $minLabel$
    end if
end for
Renumber labels in L to be sequential starting from 1
$N \leftarrow$ count of unique labels in L
return L, N

to 4-connectivity, where only horizontal and vertical neighbors (up, down, left, right) are considered, resulting in four possible connections.

This change affects how connected components are identified. In images where objects are close to each other diagonally, 8-connectivity can result in fewer, larger connected components, as it will join elements that are diagonally adjacent. In contrast, 4-connectivity might consider these as separate components.

Other than this adjustment in the neighborhood definition, the core logic of the two-pass algorithm — assigning labels in the first pass and resolving label inconsistencies

in the second pass — remains fundamentally the same between the 4-connectivity and 8-connectivity approaches. The final output of both algorithms is a labeled image where each connected component is marked with a unique label and the total number of unique connected components.

6.2 Discussion and Results

6.2.1 Image 1

Figure 20 showcases four different results of connected-component labeling on chromosome images, using different combinations of connectivity rules and threshold methods:

- **4 Connectivity, Global Thresholding:** The image shows 9 distinct labeled components, indicated by different colors. The global thresholding method may have led to a more conservative segmentation, potentially merging closer chromosomes or those with similar intensity levels.
- **8 Connectivity, Global Thresholding:** With 8 connectivity and global thresholding, the number of distinct components is reduced to 7. The increase in connectivity allows diagonally adjacent pixels to be considered part of the same component, which could cause some distinct but closely placed chromosomes to be labeled as a single object.
- **4 Connectivity, Otsu Thresholding:** Using Otsu's method for adaptive thresholding with 4 connectivity, the image has 13 distinct labels. Otsu's method seems to differentiate the chromosomes more effectively, resulting in more labeled components than the global thresholding counterpart.
- **8 Connectivity, Otsu Thresholding:** This combination resulted in 11 distinct labels. Similar to the top right image, the use of 8 connectivity has likely reduced the number of separate components compared to 4 connectivity, but Otsu's method still provides a more detailed segmentation than global thresholding.

Comparing 4 and 8 connectivity, 4 connectivity tends to produce more components as it does not consider diagonal connections. Between global and Otsu thresholding, Otsu's adaptive approach generally yields a finer segmentation, particularly in images with varying intensities, by calculating an optimal threshold that maximizes inter-class variance. If the integrity of the chromosome is considered, the 8-connectivity using the global threshold method produces fewer labels and has the best effect.

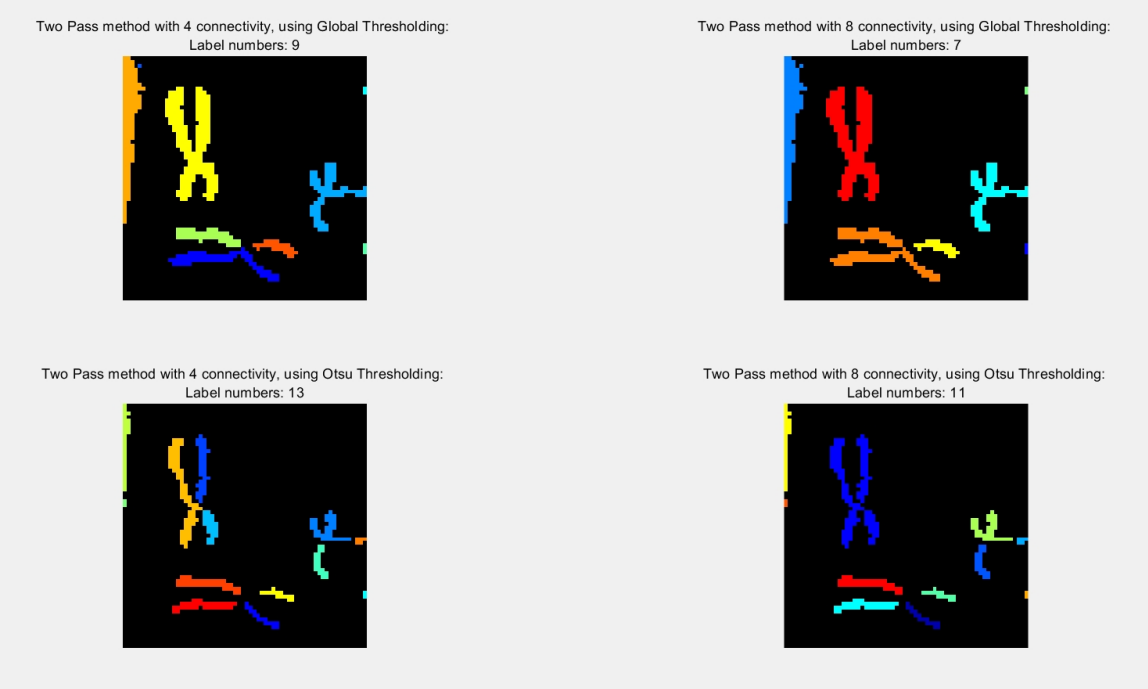

Figure 20: Labelling image of chromosomes

6.2.2 Image 2

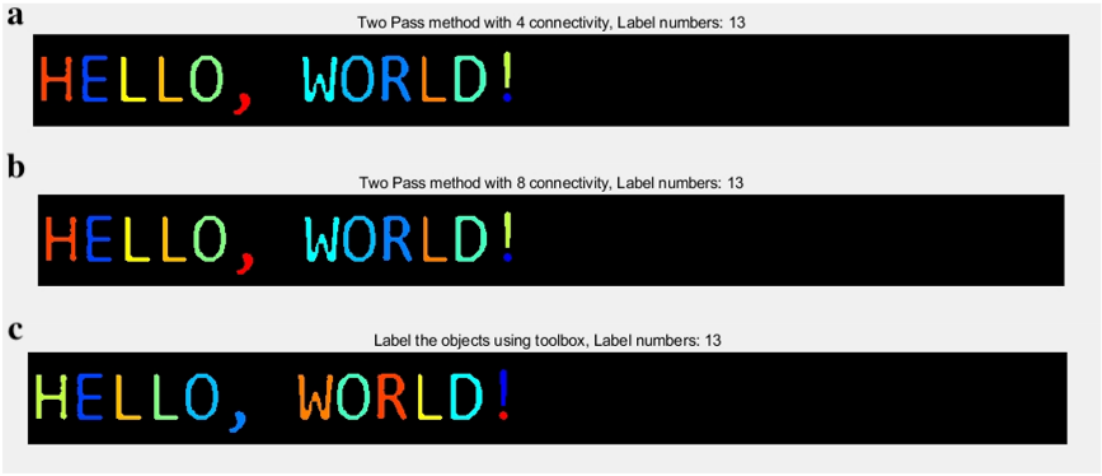

Figure 21: Labelling image of characters

In Figure 21, there are three labeled results of the text ”HELLO, WORLD!” using different methods. Because the letters after threshold are clearly separated, the effect of

labeling processing is basically the same. Although the image processed by the toolbox has different color markings, the three images are all marked with 13 labels, indicating that each method has identified the same number of distinct components within the text.

7 Rotation

Image rotation is a common image processing operation that involves rotating an image around a center point by a certain angle. The process changes the spatial arrangement of pixels in the image while keeping the same pixel values. It's commonly used to correct the orientation of images or for tasks that require aligning images to a particular angle. The main challenges in image rotation include dealing with the loss of information and aliasing. When an image is rotated, some pixels may move outside the original image frame, while new pixels generated within the frame may not correspond to any pixel in the original image. To address this, interpolation methods such as nearest-neighbor, bilinear, and bicubic can be used to estimate new pixel values based on the surrounding pixels. Additionally, the rotation can result in a larger or smaller image size depending on the rotation angle and the chosen pivot point. A general procedure of rotation is shown in Figure 22.

7.1 Algorithm

In our methodology, we employ a function named ***rotation***, as introduced in Algorithm 5. The key points of this function are angle conversion, output image size calculation, and rotation matrix application.

Since rotating an image can lead to parts of the image moving outside the original frame (and thus needing to be cropped) or new areas coming into the frame without corresponding pixels in the original image, the function handles such boundary conditions by setting these areas to a default value or by using padding strategies. Besides, the inverse rotation transformation utilizes knowledge from linear algebra and trigonometry, specifically the concepts related to rotation matrices and coordinate transformations. The transformation matrix is also shown in Algorithm 5.

7.2 Discussion and Results

The rotation function we apply is easy to understand and simple to implement. The result shown in Figure 23 is similar to the original image, appearing pixelated. According to some research, some other rotation algorithms may add additional smoothing to the image, such as bilinear and bicubic interpolation. For applications that require smoothing of the rotated image, the bicubic interpolation method should be implemented for the best improvement in the appearance of the image.

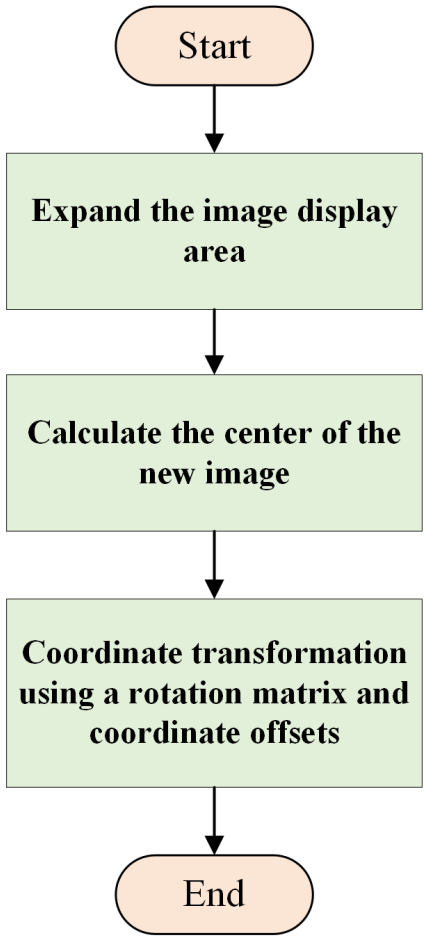

Figure 22: Rotation flow chart

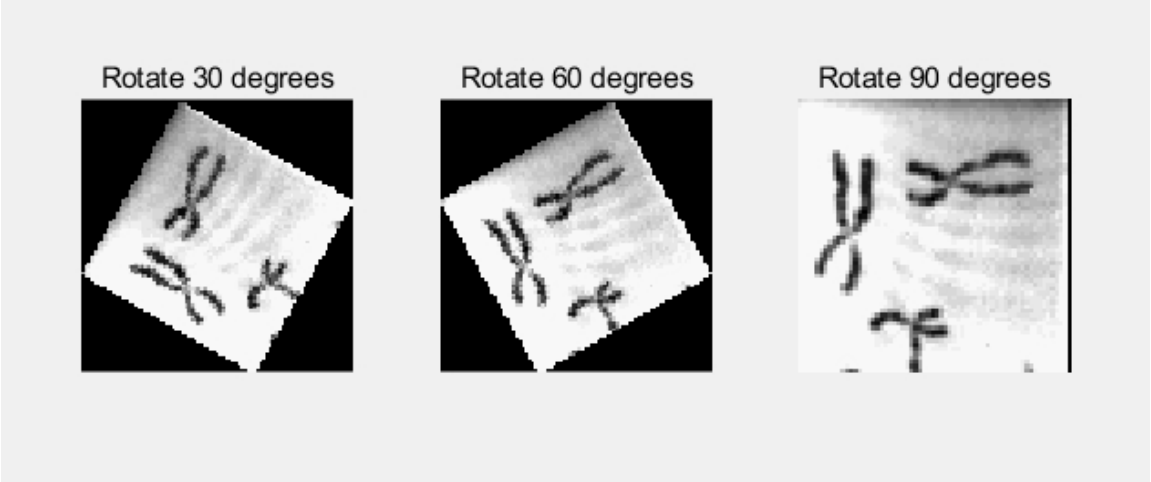

Figure 23: Rotation image of chromosomes

Algorithm 5 Image Rotation Algorithm

Input: An image matrix I, rotation angle θ
Output: Rotated image matrix R
Read the image size in row and column
Convert angle θ from degrees to radians: rad $\leftarrow$ deg2rad(θ)
Calculate the center of the image: $x_o \leftarrow$ round(width$(I)/2)$, $y_o \leftarrow$ round(height$(I)/2)$
Calculate dimensions of the output image:
$W_R \leftarrow$ round(width$(I) \cdot \cos($rad$)$ + height$(I) \cdot \sin($rad$))$
$H_R \leftarrow$ round(width$(I) \cdot \sin($rad$)$ + height$(I) \cdot \cos($rad$))$
Initialize the output image matrix R with size $H_R \times W_R$
Calculate the center of the output image: $x_R \leftarrow$ round$(H_R/2)$, $y_R \leftarrow$ round$(W_R/2)$
for each pixel (i, j) in the output image R **do**
    Translate pixel (i, j) to be centered at origin: $x \leftarrow i - x_R$, $y \leftarrow j - y_R$
    Apply the rotation matrix to (x, y):
    $(x', y') \leftarrow (x, y) \cdot \begin{bmatrix} \cos(\text{rad}) & -\sin(\text{rad}) \\ \sin(\text{rad}) & \cos(\text{rad}) \end{bmatrix}$
    Translate coordinates back: $x_{new} \leftarrow$ round$(x') + x_o$, $y_{new} \leftarrow$ round$(y') + y_o$
    if (x_{new}, y_{new}) is within the bounds of I **then**
        Set $R(i, j)$ to the value of $I(x_{new}, y_{new})$
    else
        Set $R(i, j)$ to a background value (e.g., 0)
    end if
end for
return The rotated image matrix R

8 Image Segmentation

For the tasks concerning Image 2, we need to execute a segmentation process, as depicted in Figure 24. This will involve separating and distinctly labeling the various characters within the image. The characters to be extracted from Image 2 are sequentially arranged as follows: 'H', 'E', 'L', 'L', 'O', 'W', 'O', 'R', 'L', and 'D'. We use Figure 11 to process the segmentation.

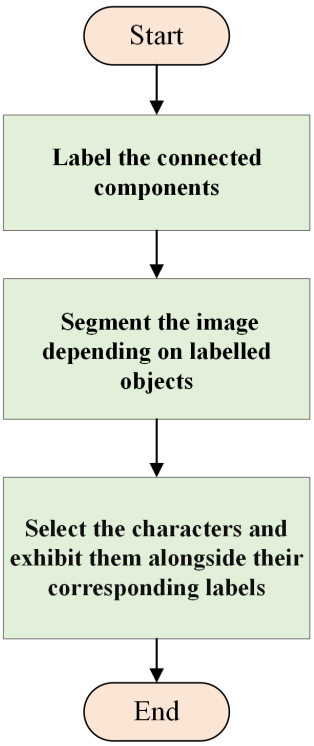

Figure 24: Process of segmentation for image 2

8.1 Algorithm

The segmentation process comprises two steps. Initially, the task involves labeling connected components within the image. Subsequently, the process focuses on segmenting individual characters.

8.1.1 Connected Components Labeling

The binary image undergoes a connected component analysis using ***bwlabel*** function. This function identifies and labels each connected component within the binary image. As a result, *'labeled_img'* contains distinct labels for each object. This step is critical for differentiating between various elements in the image.

8.1.2 Image Segmentation

Algorithm 6 Image Segmentation Algorithm

Input Variable: A labelled image, segment_size
Output Variable: An array of segmented_images
Initialization:
Extract all unique non-zero values from labelled_image
Initialize segmented_images as a logical array with dimensions segment_size $\times$ segment_size $\times$ number of unique objects
for each unique value in unique non-zero values **do**
    Find pixel coordinates corresponding to the current object label
    Determine the bounding box for the current object
    Initialize a binary image of size segment_size $\times$ segment_size to true
    Calculate the centering offset for the object
    for each pixel coordinate in the bounding box **do**
        if pixel belongs to the current object **then**
            Set corresponding pixel in a binary image to false
        end if
    end for
    Add the binary image to segmented_images array
end for
return segmented_images

In our methodology, we employ a function named ***segment***, as introduced in Algorithm 6. This function is invoked with two arguments: the labeled image and the segment size. The outcome of this function is a series of segmented images. However, these segmented images incorporate punctuation marks, which are extraneous to our intended analysis. To address this, we implement a subsequent stage of refinement.

In this stage, our objective is to isolate and exhibit only the character segments, effectively excluding the punctuation. This is achieved through the utilization of two distinct

arrays. The first array, designated for selection, is utilized to pinpoint the specific segments that correspond to characters. The second array serves a labeling function, assigning identifiable tags to each of the chosen segments. Through this dual-array approach, we efficiently extract and display the segments of interest, ensuring that our analysis is focused solely on the character data.

8.2 Discussion and Results

We successfully obtained the characters from Figure 11 following the segmentation process. After the first step, we separated characters and punctuation from the original image, as depicted in Figure 25. Subsequently, we proceeded to selectively exhibit only the characters with their labels, which are showcased in Figure 26. The results proved that the function ***segment*** is capable of separating objects in images with a proper segment size.

Figure 25: Segmented images containing all objects from Figure 11

Nevertheless, an issue arises during the labeling of connected components. As demonstrated in Figure 25, the exclamation mark from the original image has been erroneously classified into two separate components by the ***bwlabel*** function. Although this error does not impact the analysis of our desired characters, which consist of singular parts, it may present challenges in labeling characters with multiple components, such as *i*. Consequently, the labeling methodology requires modification to accommodate such scenarios.

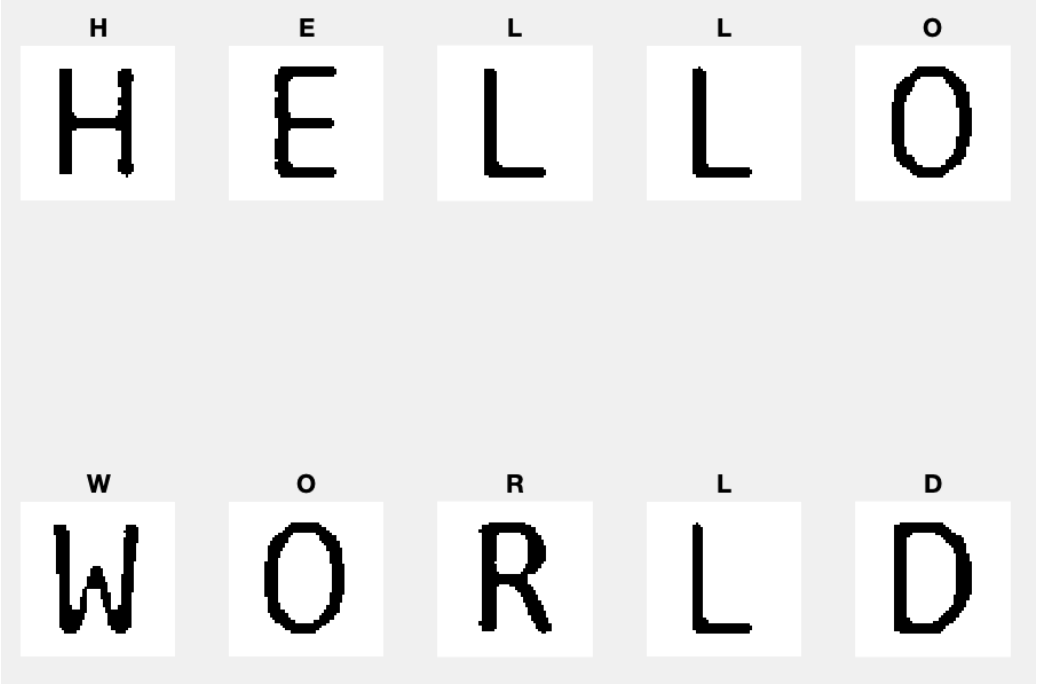

Figure 26: Desired Segments Alongside Their Labels

9 Image Classification

The field of image classification has experienced rapid advancements in recent years. In this section, we explore the application of three conventional classification methodologies to develop classifiers capable of recognizing various characters. The procedural framework for image classification is depicted in Figure 27. The initial step involves loading the dataset and partitioning it into training(75%) and test(25%) subsets. In the next step, we extract features from the images using the Histogram of Oriented Gradients (HOG) technique, followed by training the classifiers with these features. Ultimately, the trained classifiers are utilized to predict characters in Image 2, assessing their efficacy in accurately differentiating between diverse characters.

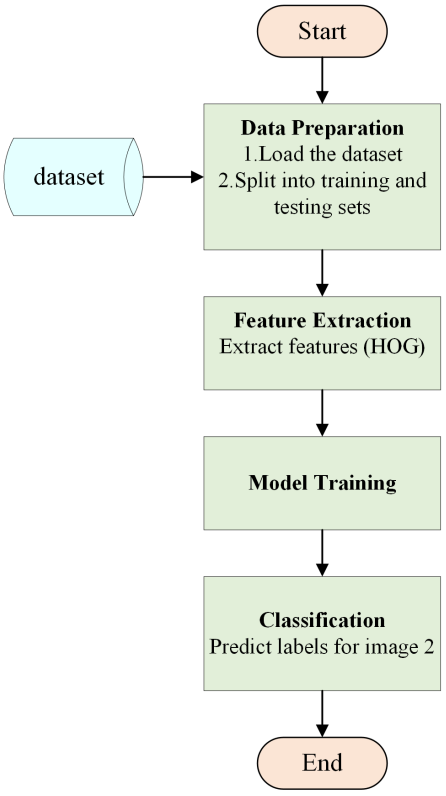

Figure 27: Process of Image Classification

9.1 Feature Extraction

The Histogram of Oriented Gradients (HOG) feature extraction technique plays a pivotal role in preparing data for training our classifiers. We extract HOG features at three

distinct cell sizes: 8x8, 16x16, and 32x32 pixels. This strategic variation in cell sizes is essential for discerning the most effective granularity for feature extraction. We utilize the MATLAB function ***extractHOGFeatures***, which generates both the HOG feature vectors (*hog_8x8*, *hog_16x16*, *hog_32x32*) and their respective visual representations (*vis8x8*, *vis16x16*, *vis32x32*). We display the extracted features and the original image in 28, which is critical for understanding the impact of different cell sizes on the HOG feature extraction.

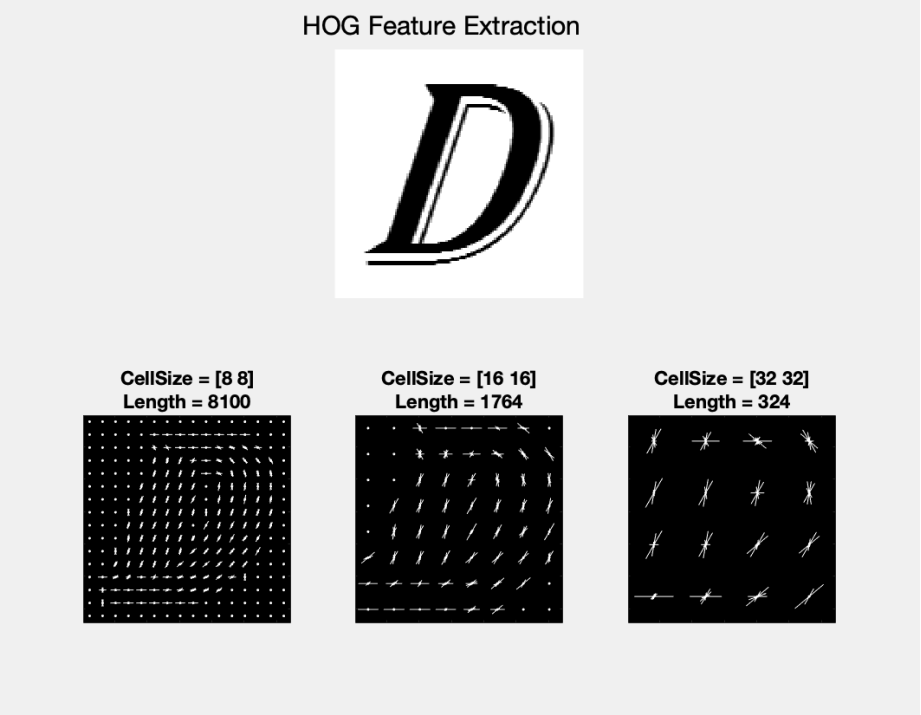

Figure 28: Feature Extraction Using HOG

Based on visual observation from these plots, we selected the proper cell size for further feature extraction. An extraction of HOG features is then performed on the entire dataset. Two arrays ***train_features*** and ***test_features***are initialized to store the HOG features for each image. Two loop iterates over all images in both datasets, extracting HOG features at the previously selected cell size. The labels corresponding to train and test images are also stored in ***train_labels*** and ***test_labels***. This methodical approach ensures a structured and efficient preparation of data for subsequent classifier training.

9.2 Classification Algorithms

In our study, we explore three different image classification methods: k-Nearest Neighbours (kNN), Support Vector Machine (SVM), and Self-Ordered Maps (SOM), as shown

in Figure 29. Once the classifier is trained, it is used to predict the labels of the test dataset (***test_features***). The accuracy of the model is then calculated by comparing these predicted labels with the actual labels (***test_labels***). Our comparative analysis of these methods will be anchored on three critical parameters: accuracy, predictive outcomes, and computational cost.

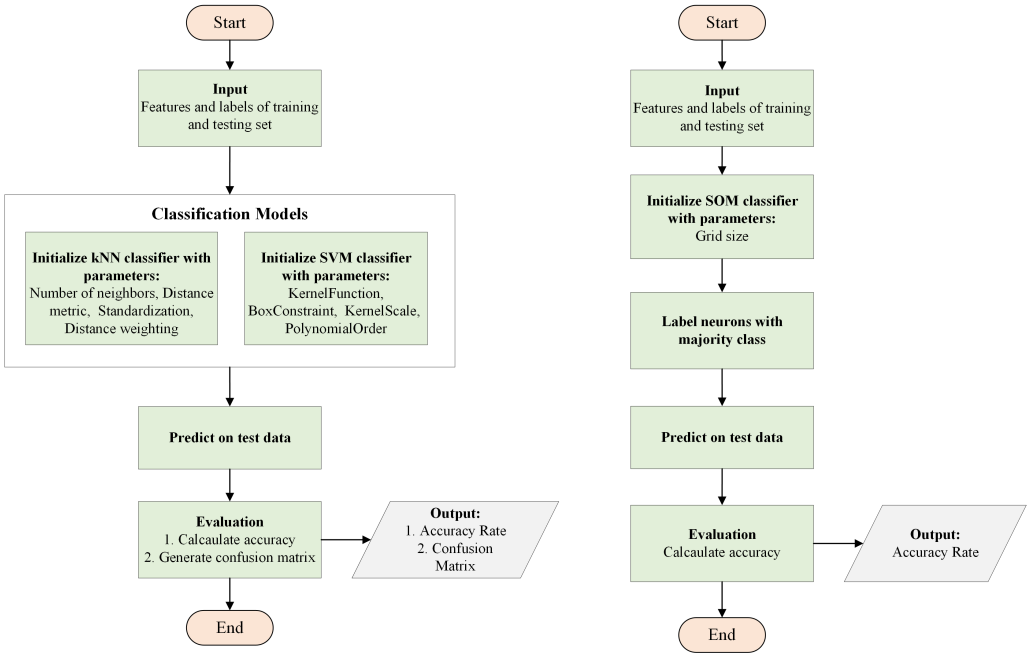

Figure 29: Overview for Classification Algorithms including kNN, SVM, and SOM

9.2.1 k-Nearest Neighbours (kNN)

The k-Nearest Neighbors (kNN) algorithm is a straightforward and widely-used method for image classification. It classifies an image by identifying the 'k' closest training images in the feature space and assigning the most common class among these neighbors to the new image. The effectiveness of kNN in image classification lies in its simplicity and ability to handle complex decision boundaries without the need for an explicit training phase. However, its performance heavily depends on the choice of 'k' and the distance metric used.

In our implementation, the MATLAB function ***fitcknn*** is employed for the training of the kNN classifier. This training process is executed utilizing the feature vectors (*train_features*) and the corresponding labels (*train_labels*) derived from the training dataset. The classifier is configured with several options:

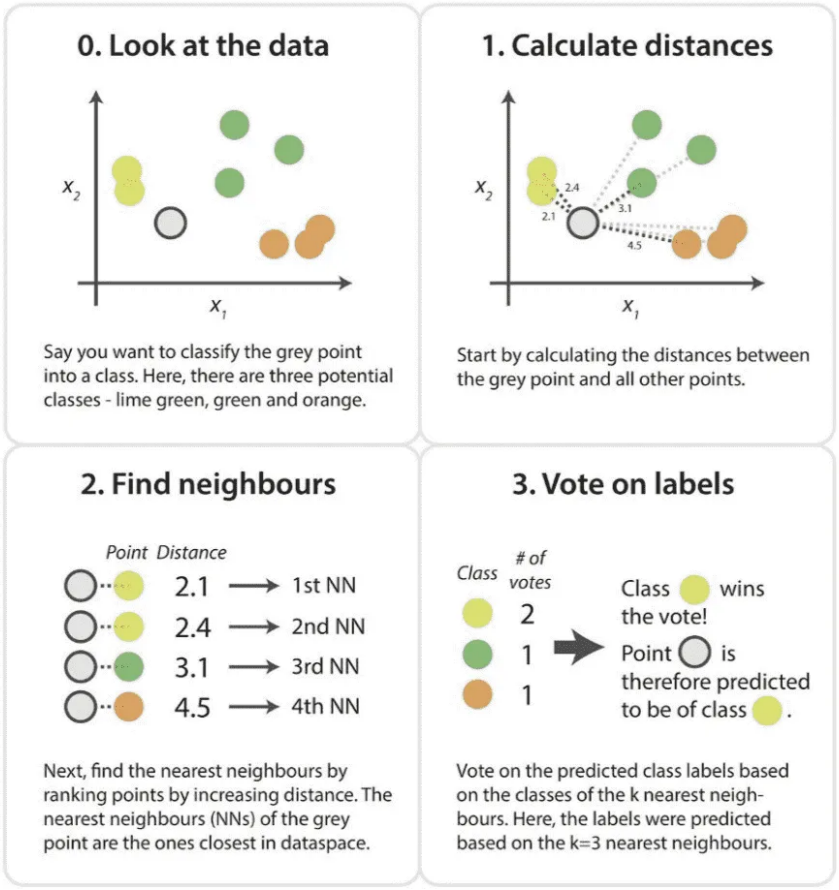

Figure 30: An explanation of kNN algorithm ($https://medium.com/swlh/k-nearest-neighbor-ca2593d7a3c4$)

- ***Distance*** means the distance metric used to compute the distance between points such as Euclidean, city block, and Minkowski. All these distance matrices are presented in Table 1.

Table 1: Distance Matrix in kNN

Distance Metric	Description	Formula
'euclidean'	Straight-line distance between two points	$d(P,Q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$
'cityblock'	the distance traveled along orthogonal city blocks between two points	$d(P,Q) = \sum_{i=1}^{n} \lvert p_i - q_i \rvert$
'minkowski'	a generalization of both Euclidean and City Block distances	$d(P,Q) = (\sum_{i=1}^{n} \lvert p_i - q_i \rvert^p)^{1/p}$

- ***NumNeighbors*** represents the number of neighbors.
- ***Standardize*** determines whether the features are standardized or not. When set to 0 (false), it indicates that the features remain in their original form, with no standardization applied. Conversely, the features are standardized when the value of this parameter is 1(true).
- ***DistanceWeight*** adjusts the weight given to each of the ***k*** nearest neighbors based on their distance from the query point. When set to *'equal'*, each of the ***k*** neighbors contributes equally to the classification, regardless of their actual distance from the query point. With *'inverse'*, the contribution of each neighbor is inversely proportional to their distance. Closer neighbors will have a more significant influence on the classification than those farther away. When set to *'squaredinverse'*, the weight is inversely proportional to the square of the distance. This option further amplifies the influence of closer neighbors compared to the *'inverse'* setting.

9.2.2 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a conventional learning algorithm used to classify images by finding the hyperplane that maximally separates different classes. It uses support vectors and margins for classification and employs kernel functions to handle non-linear data.

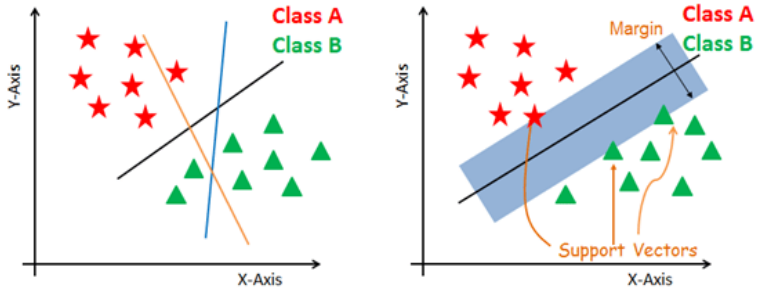

Figure 31: An explanation of SVM algorithm ($https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python$)

In our implementation, the MATLAB function ***templateSVM*** and ***fitcecoc*** are employed for the training of the SVM classifier. We use the One-vs-All strategy because it requires training only N classifiers regardless of the number of classes, so it's computationally more efficient than One-vs-One(OvO) for a large number of classes.

The SVM classifier is configured with several options:

- ***KernelFunction*** defines the type of kernel function to be used in the SVM including linear, gaussian, and polynomial.
- ***BoxConstraint*** controls the trade-off between allowing misclassification of the training data and forcing rigid margins. It is a regularization parameter that helps to prevent overfitting.
- ***KernelScale*** is used specifically with kernel functions like the Gaussian kernel to define the scale of the kernel function, which can affect the decision boundary's smoothness.
- ***PolynomialOrder*** sets the order of the polynomial.

9.2.3 Self-Ordered Maps (SOM)

The Self-Organizing Map (SOM) algorithm is an unsupervised neural network technique used for clustering, visualization, and dimensionality reduction, often applied in image classification. It consists of neurons organized in a grid, usually two-dimensional. Each neuron has a weight vector of the same dimensionality as the input data. During training, the SOM algorithm iteratively adjusts these weights. In image classification, SOMs can cluster similar images, making them useful for tasks like pattern recognition or feature extraction.

We developed a function, designated as ***trainSOM***, specifically to train our Self-Organizing Map (SOM) classifier. The underlying algorithm of ***trainSOM*** is detailed in Algorithm 7.

Algorithm 7 Self-Organizing Map Algorithm

1: **Input:** training_features, training_labels, test_features, test_labels
2: **Output:** som_classifier, neuron_labels
3: *Initialization:* Set grid_size and Initialize som_net using ***selforgmap*** with grid_size
4: *Training:* Train som_net with training_features
5: *Label Neurons:* Assign labels to neurons with label_neurons function using som_net, training_features, and training_labels
6: *Testing:* Classify test_features using classify_with_som function and som_net
7: *Compute Accuracy:* Calculate the accuracy of the classification
8: **return** som_classifier and neuron_labels

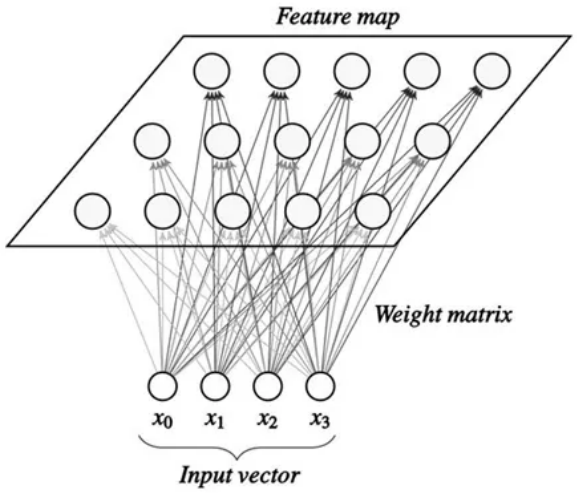

Figure 32: An explanation of SOM algorithm ($https://medium.com/machine-learning-researcher/self-organizing-map-som-c296561e2117$)

9.3 Discussion and Results

After tuning hyperparameters and comparing the performances of different classifiers, the SVM classifiers emerged as the most proficient, achieving the highest accuracy rate of 95.95%. While both kNN and SVM classifiers demonstrated capability in handling this classification task, the accuracy of the SOM classifier was notably lower. However, it is important to note that all classifiers were successful in accurately predicting the labels of the characters depicted in Figure 26.

9.3.1 k-Nearest Neighbours (kNN)

Following the tuning of hyperparameters, the kNN classifier (with hyperparameters: 'Distance': 'cityblock', 'Standardize': 0, 'NumNeighbors': 5, 'DistanceWeight': 'inverse') achieved a peak accuracy of 93.70% with a feature extraction cell size of 32. The accuracy of most kNN classifiers is higher than 92.00%, indicating that kNN classifiers perform well in this image classification task. Furthermore, our kNN classifiers successfully predicted the characters depicted in Figure 26. The detailed prediction results are showcased in Figure 33.

The following paragraphs show how we tune hyperparameters.

Cell Size: In our investigation of the optimal cell size for HOG feature extraction in kNN classification, we observed a notable variance in accuracy and computational efficiency.

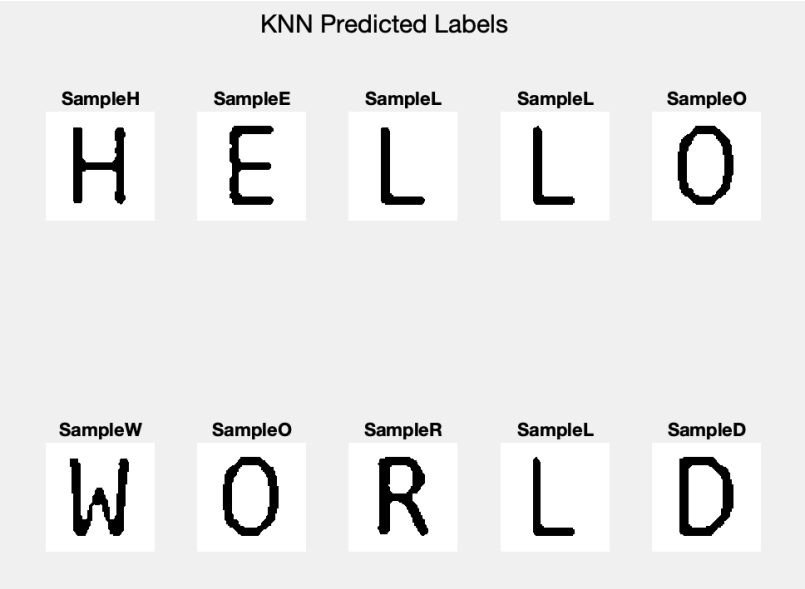

Figure 33: Results of kNN classifier prediction.

As Figure 35 demonstrates, the accuracy is at its lowest with a feature extraction cell size of 8. Interestingly, more features(Cell Size: 8) in the training and testing sets didn't increase the accuracy of the kNN classifiers. Moreover, this cell size incurs a significantly higher computational cost. For instance, training and testing the kNN classifier (with hyperparameters 'Distance': 'cityblock', 'Standardize': 0, 'NumNeighbors': 5, 'DistanceWeight': 'inverse') takes 25.04 seconds when the cell size is 8. This duration is substantially longer compared to the 6.02 seconds required for a cell size of 16, and 1.33 seconds for a cell size of 32. Therefore, we have decided to use cell sizes of 16 or 32 for feature extraction in the further analysis.

Standardize: We also explore the impact of feature standardization on the accuracy of kNN classifiers. It was observed that setting 'Standardize' to 0 (false) consistently resulted in higher accuracy, as shown in Figure 36.

Distance Weight: Additionally, we investigated the effect of varying the distance weight of neighbors on the accuracy of kNN classifiers. Our observations indicate that configuring 'DistanceWeight' to either 'inverse' or 'squaredinverse' generally led to enhanced accuracy in the majority of cases, as demonstrated in Figure 37.

Distance Matrix: Finally, we explored the influence of distance matrix on the accuracy of kNN classifiers. We found the kNN classifier attained the highest accuracy when the Distance Matrix is 'cityblock' across various feature extraction cell sizes. This finding is substantiated by the data presented in Figure 38.

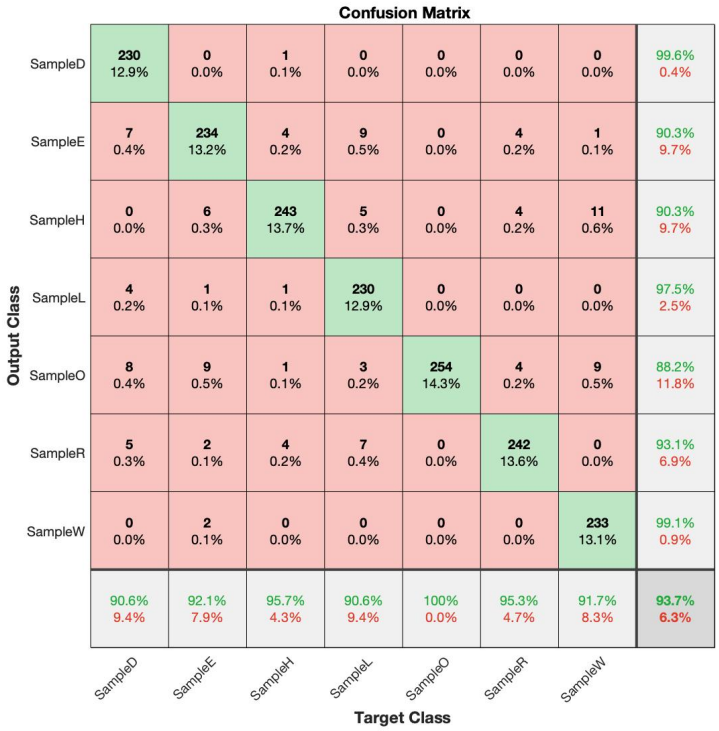

Figure 34: Confusion Matrix of the kNN classifier with highest accuracy.

9.3.2 Support Vector Machine (SVM)

After optimizing hyperparameters, the SVM classifier (parameters: 'KernelFunction': 'gaussian', 'boxConstraints': 1, 'kernelScales': 'auto') reached an optimal accuracy of 95.95% with a feature extraction cell size of 32. Notably, this level of accuracy was achieved with an efficient training duration of 4.6 seconds. The accuracy of the majority of SVM classifiers consistently surpassed 94.00%, underscoring their robust performance in this image classification task. Additionally, the SVM classifiers adeptly identified the characters shown in Figure 26, with the comprehensive prediction results presented in Figure 39.

In consideration of the computational demands associated with a feature extraction cell size of 8, we opted to train the SVM model using only cell sizes of 16 or 32. The *KernelScale* parameter was set to 'auto', permitting the SVM to autonomously determine the most suitable scale based on the dataset characteristics. SVM classifiers underwent training with varying *KernelFunction* settings and different *boxConstraints* across these selected feature extraction cell sizes. The comprehensive accuracy results for all SVM

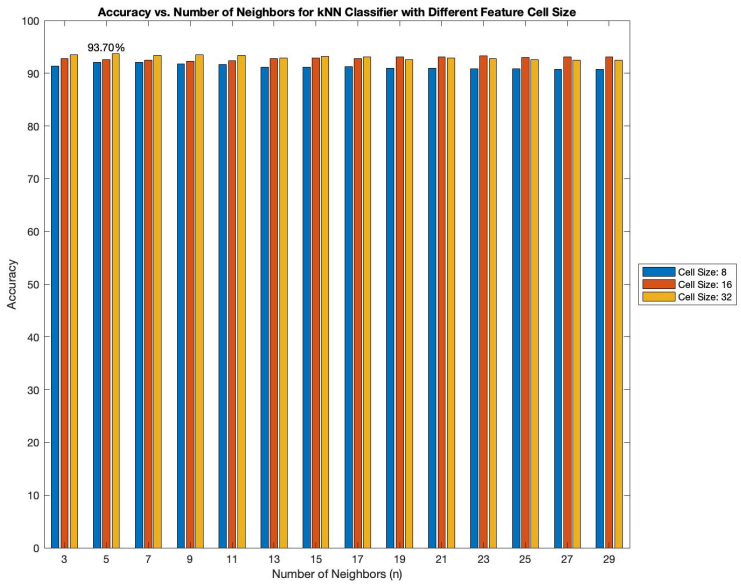

Figure 35: kNN Classifier with Different Feature Extraction Cell Size. (Hyperparameters: 'Distance': 'cityblock', 'Standardize': 0, 'DistanceWeight': 'inverse')

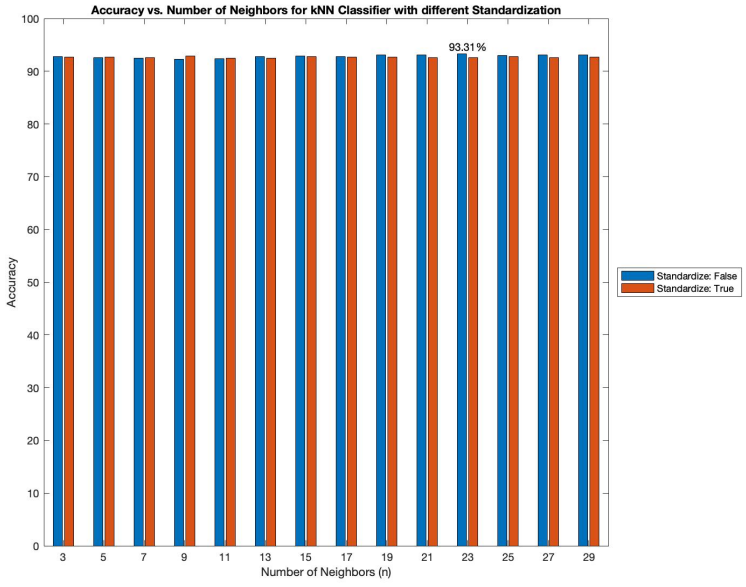

Figure 36: kNN Classifier with Different Standardization. (Hyperparameters: 'Distance': 'cityblock', 'DistanceWeight': 'inverse', Cell Size: 16)

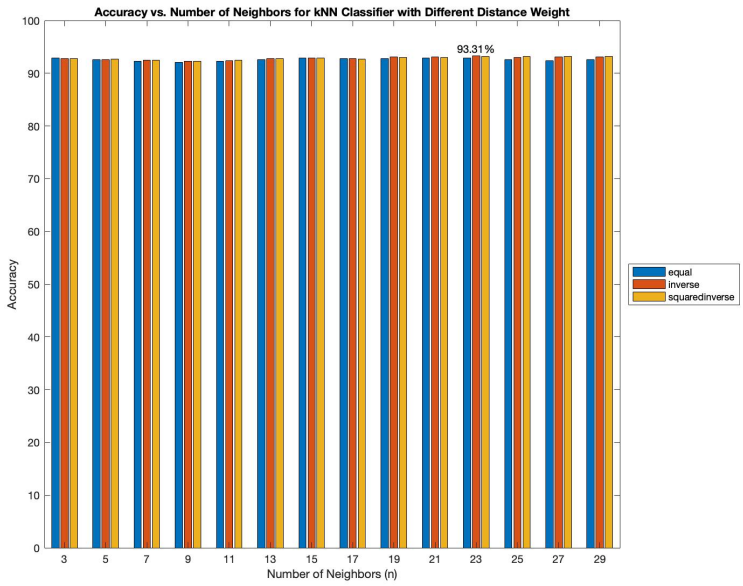

Figure 37: kNN Classifier with Different Standardization. (Hyperparameters: 'Distance': 'cityblock', 'Standardize': 0, Cell Size: 16)

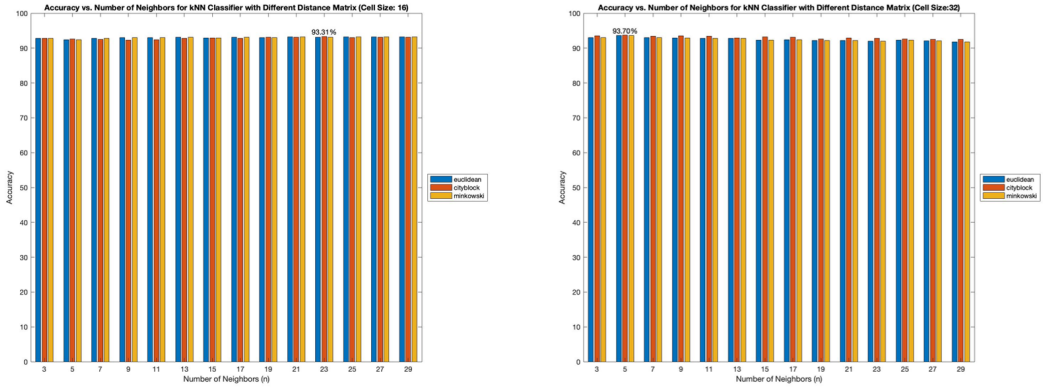

Figure 38: kNN Classifier with Different Distance Matrix. (Hyperparameters: 'Standardize': 0, 'DistanceWeight': 'inverse', Cell Size: 16 or 32)

models are depicted in Figure 41.

As indicated in Table 2, our analysis reveals that in this particular scenario, the SVM classifiers exhibit improved performance with reduced information. This observation is supported by the fact that both the median and mean accuracies are higher when the cell

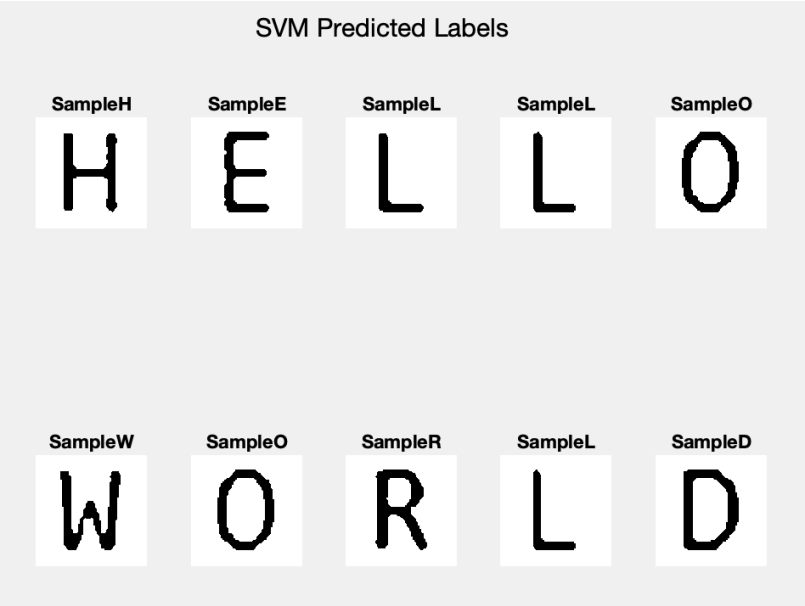

Figure 39: Results of SVM classifier prediction.

size is set to 32.

Table 2: Performance Metrics of SVM Classifiers at Different Cell Sizes

Cell Size	Max(%)	Min(%)	Median(%)	Mean(%)
16*16	95.16	93.64	94.91	94.80
32*32	95.95	91.17	95.16	94.82

9.3.3 Self-Ordered Maps (SOM)

Compared to the kNN and SVM classifiers, the SOM classifier exhibits a markedly lower accuracy rate of 65.06%. Despite this, it retains the capability to accurately predict the labels of characters, as illustrated in Figure 42.

SOMs involve a training phase where the network learns to map the input space to a typically two-dimensional grid of neurons. This process can be time-consuming, especially compared with simple models such as kNN and SVM. In our experiments, we observed that training the SOM classifier was significantly more time-consuming, exceeding 500 seconds, in comparison to kNN and SVM. Consequently, this led us to forego further hyperparameter tuning for the SOM classifier.

SOM, fundamentally an unsupervised learning algorithm, is predominantly utilized for

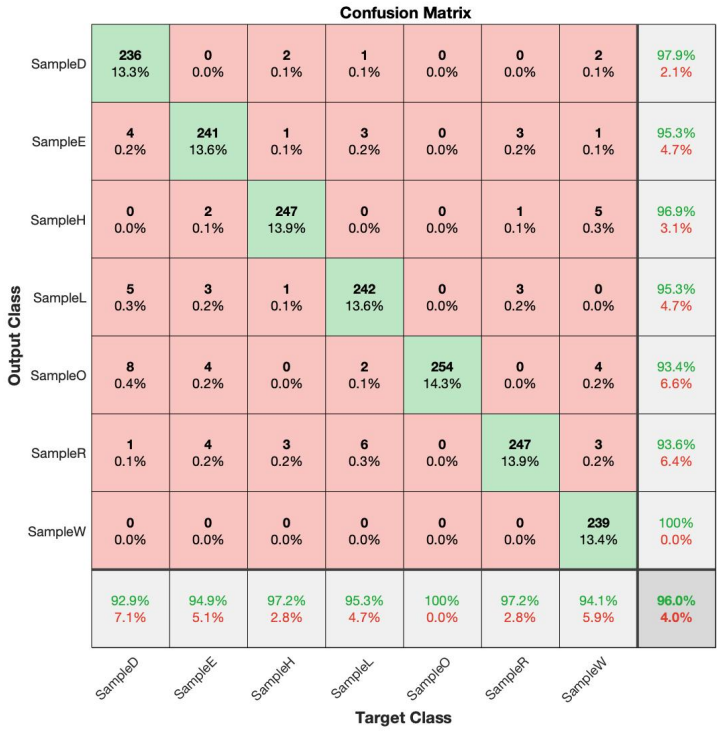

Figure 40: Confusion Matrix of the SVM classifier with highest accuracy.

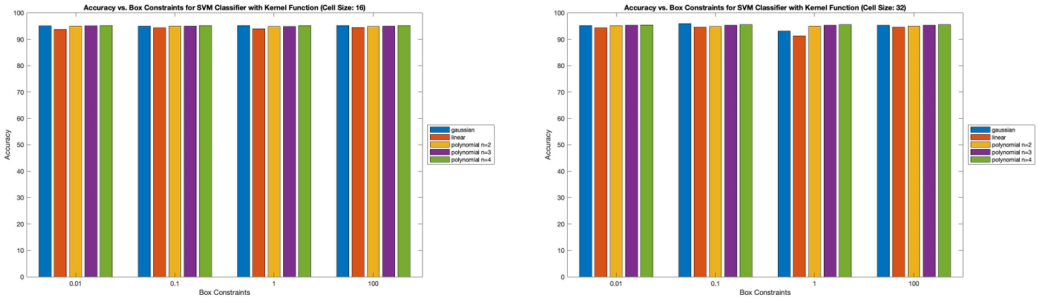

Figure 41: SVM Classifier with Different Distance Matrix.

clustering and visualization purposes rather than for classification [1]. Given its design, which does not intrinsically account for class labels during training, it is understandable that SOM may not exhibit optimal performance in image classification tasks.

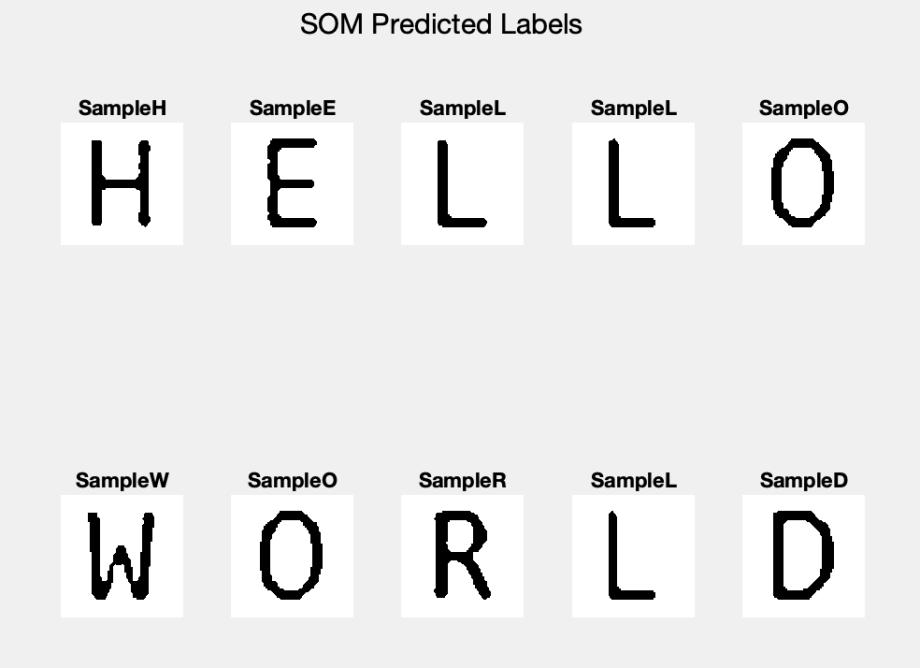

Figure 42: Results of SOM classifier prediction.

10 Conclusion and Future Work

We have successfully executed a series of image processing tasks, including binarization, thinning, outlining, segmentation, rotation, and classification, for Image 1 and Image 2. These tasks were accomplished using a combination of MATLAB toolbox utilities and custom-developed functions. The concluding phase of the processing for Image 1 is depicted in Figure 43, whereas the final stage of processing for Image 2 is illustrated in Figure 44.

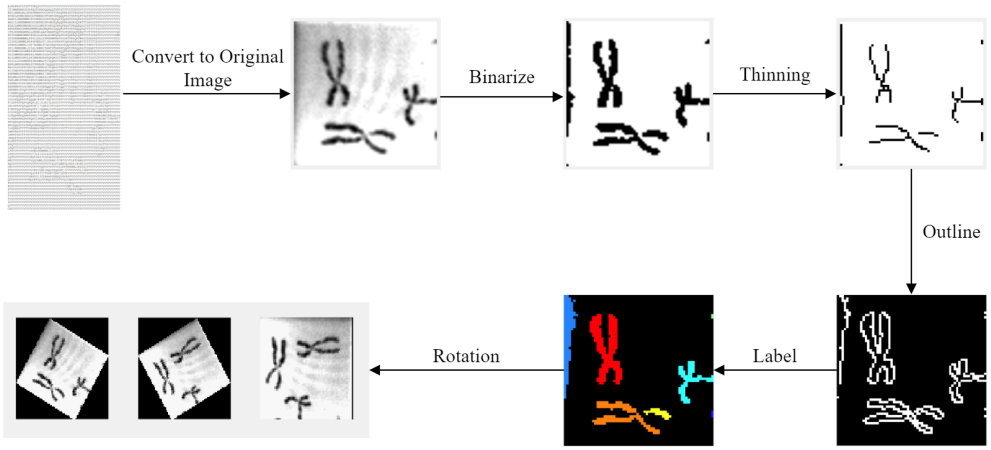

Figure 43: Overview of Image Processing for Image 1

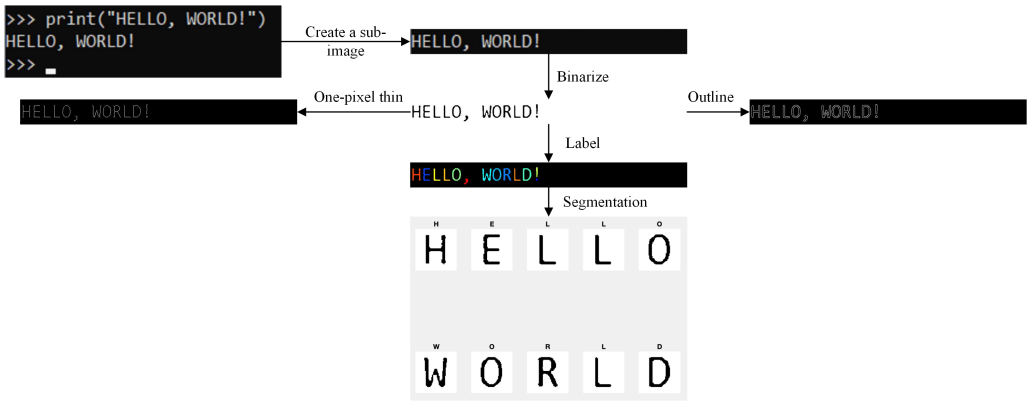

Figure 44: Overview of Image Processing for Image 2

However, there remains scope for improvements in our image processing methods.

Thinning. The standard Zhang-Suen thinning algorithm is an effective technique for reducing foreground pixels in a binary image to a skeletal remnant while preserving the

topology and overall morphology of the original pattern. Therefore, we will make more efforts to increase computational efficiency including the adoption of parallel processing paradigms and the application of more efficient data structures for image representation. Moreover, we will adopt more techniques to refine the thinning picture like Dilation, Bridging, Removal of Spurious Pixels, and Skeleton Clean-up.

Segmentation. In the segmentation process applied to Image 2, morphological operations such as dilation can be employed to bridge gaps between disconnected parts of the same character. This technique is particularly useful in ensuring that the algorithm perceives both segments of a character, such as the dot and the line in the letter *i*, as a singular unit. For a more sophisticated approach, the implementation of machine learning-based character recognition methods could be considered. These advanced techniques are adept at more accurately identifying and labeling complex characters.

Classification. In the current image classification task, our approach was limited to a single feature extraction method (HOG). Future endeavors should incorporate additional feature extraction methods for comparison. Both kNN and SVM algorithms demonstrated robust performance in our task, which can be attributed to the relative simplicity of the images in our dataset. For more complex scenarios, especially when classifying color images, deep learning algorithms like Convolutional Neural Networks (CNN) [2] are generally more adept and thus recommended.

References

[1] J. Vesanto and E. Alhoniemi. Clustering of the self-organizing map. *IEEE Transactions on Neural Networks*, 11(3):586–600, 2000.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.