1. Image#2

2.1 Introduce

This project implements several image processing tasks using MATLAB R2024a, including image extraction, binarization, skeletonization, character segmentation, and classification using k-Nearest Neighbors (kNN). The final step involves hyperparameter optimization and resize for the kNN model. The MATLAB codes and results has been open sourced in [github.com/0123YHYQ1129/YeQing\_Machine\_Version\_Project](https://github.com/0123YHYQ1129/YeQing_Machine_Version_Project).

2.2 Folder structure

data/: Contains the dataset used for classification.

data/hello\_world.jpg: The original image used in the image processing tasks.

data/p\_dataset\_26/: Contains subfolders for each character ('D', 'E', 'H', 'L', 'O', 'R', 'W'), each holding images for training and testing.

results/: Stores the output images generated during the image processing tasks.

src/: Contains the utility functions used in the project.

2.3 Matlab Scripts

2.3.1 Q1 Loading the original image

Read the image from the path data/hello\_world.jpg using imread() and store it as originalImage, then display the image using the imshow() function.



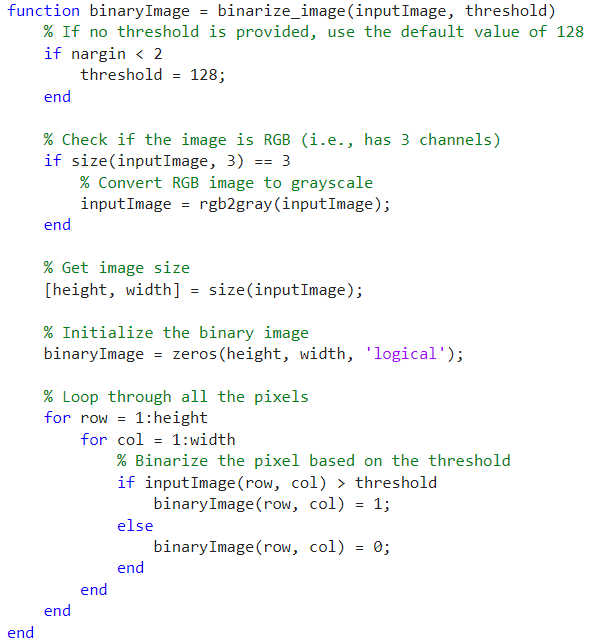
2.3.2 Q2 Extracting a sub-image

Get the image dimensions using the size() function. Take start\_row and end\_row as 1/3 and 2/3 of the height, respectively. Then, extract originalImage (startRow:endRow, :, :) as a sub-image. Finally, save and display the sub-image.



2.3.3 Q3 Binarizing the sub-image

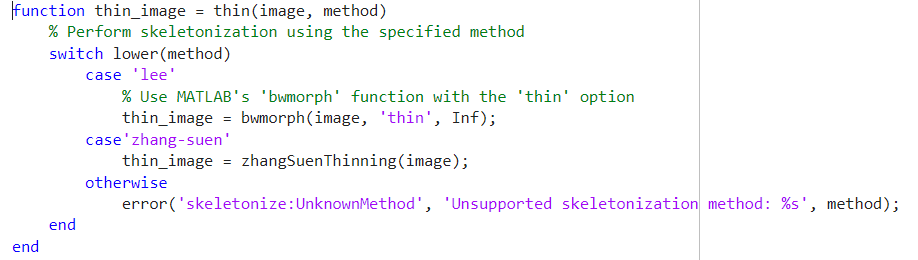
I designed the function binaryImage within src/binarize\_image.m. This function accepts an image and a threshold as input. If the input image is in RGB format, it's first converted to grayscale. Subsequently, the function iterates through each pixel. If a pixel's grayscale value is less than the provided threshold, it's set to 0; otherwise, it's set to 1. The binaryImage function is then called within main\_1\_6.m to binarize a sub-image, resulting in a binary image which is then saved.





2.3.4 Q4 Thin the binary image

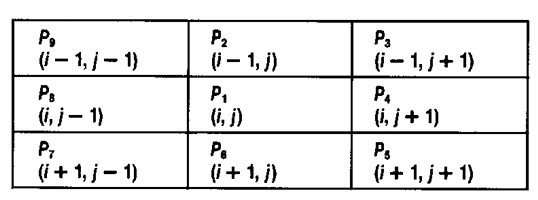
I designed the function thin within src/thin.m. This function accepts a binary image and a method as input, with the default method set to 'lee'. And you can also choose the ‘Zhang-Suen’.



The "**Lee algorithm**" utilizes MATLAB's built-in function bwmorph(image, 'thin', Inf) to perform thinning on binary images. Unlike skeletonization, thinning does not always guarantee that the resulting structure is the central symmetric skeleton, as it retains more information about the object's edges.The algorithm is introduced in the paper **["Thinning Methodologies - A Comprehensive Survey."](https://ieeexplore.ieee.org/document/161346)**

When n = Inf, the image is iteratively thinned by removing pixels from the object's boundary, reducing the object to lines while preserving connectivity. Objects without holes are reduced to lines with minimal connectivity, while objects with holes are reduced to loops that connect each hole with the outer boundary. This option preserves the object's Euler number.

The Zhang-Suen algorithm was implemented by me without using external libraries. It consists of two sub-iteration steps and operates by repeating these sub-iterations until no further changes are possible, resulting in a thinned pattern. For each non-zero center pixel, as illustrated below, its 8-neighborhood pixels are examined. The algorithm uses two sub-iterations to determine if the non-zero center pixel should be set to zero.



Step1：Check if the point meets the following conditions:

(a)

(b) A() = 1

(c)

(d)

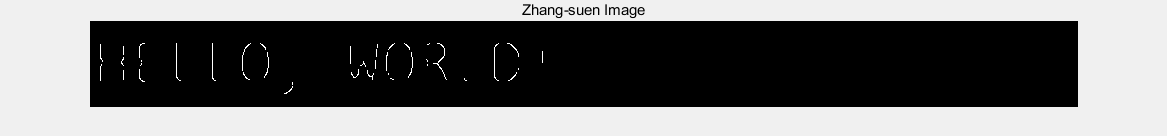
Here, A() represents the number of 0-to-1 transitions in the sequence from to and back to.

Step 2：After removing corner points, modify conditions (c) and (d) and execute again.

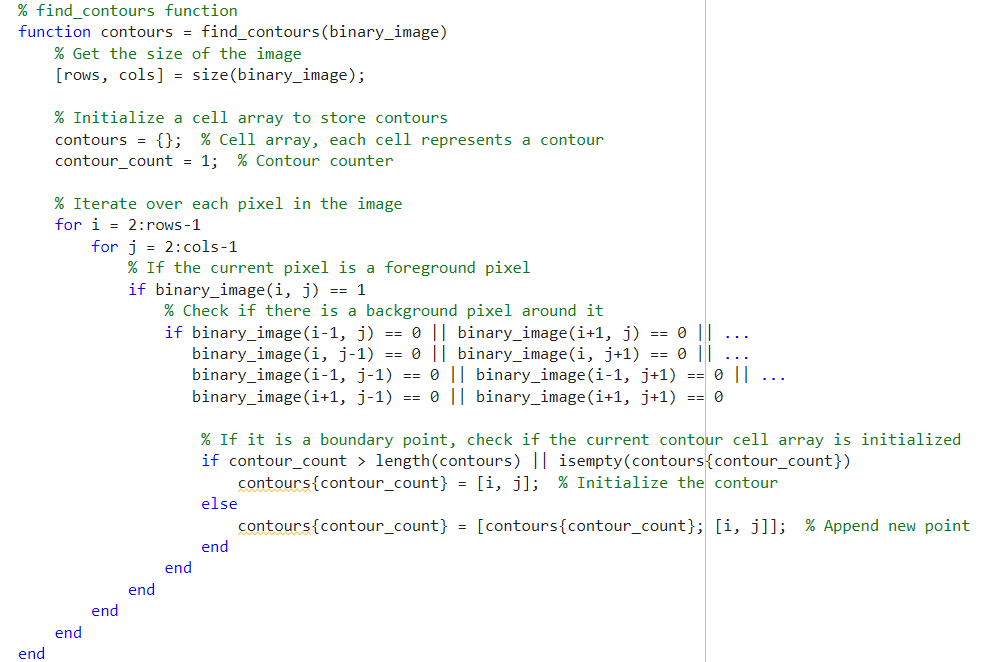
(c)

(d)

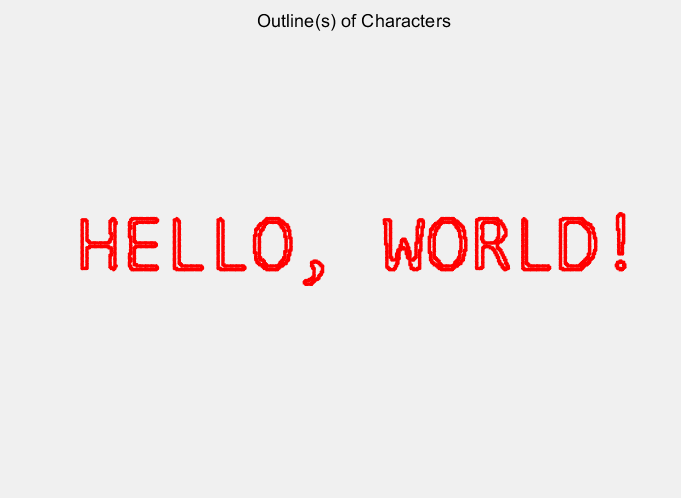
If the conditions for both Step 1 and Step 2 are satisfied, set the pixel's grayscale value to 0. This part of the code is located in src\zhangSuenThinning.m. Two methods can be used to obtain the results.



2.3.5 Q5 Get the outline(s) of characters of the image.

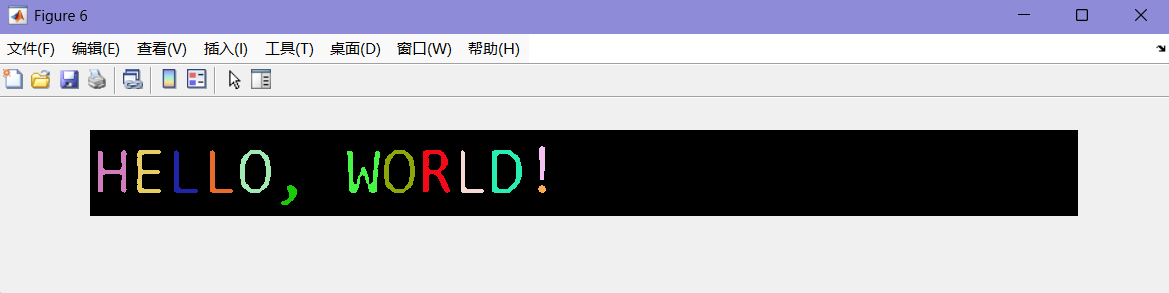
In the src/find\_contours function, the code aims to extract boundary contours from a binary image. The core approach involves iterating over each foreground pixel in the image and determining if it lies on the object's edge. The identification of boundary points is based on whether there are any background pixels within its 8-neighborhood.

In the main.m function, the binaryImage is passed as an input to this function to obtain the boundary contours, which are then plotted on the image.

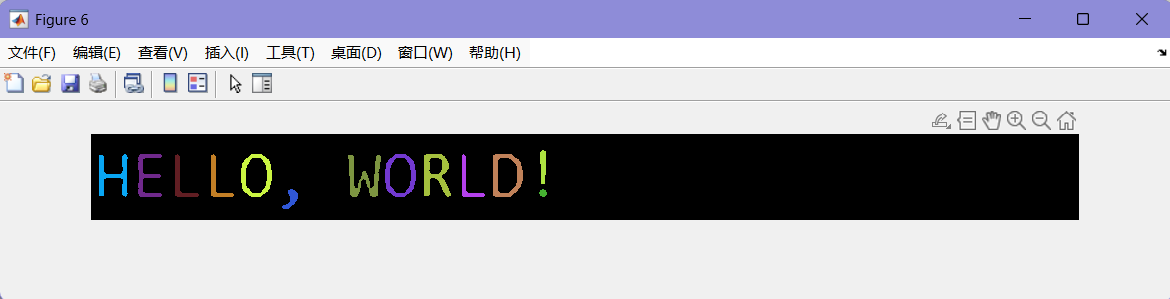


2.3.6 Q6 Segment the image to separate and label the different characters

In this section, I used the MATLAB built-in function bwlabel() to label the connected components in a binary image. The function allows selecting either 4-connectivity or 8-connectivity, with 8-connectivity as the default. This setting can be adjusted in src\segmentAndLabelCharacters.m. Each connected component is assigned a different color and displayed. The results for 4-connectivity and 8-connectivity are shown in the figure below.



8-connectivity



4-connectivity

Theoretically, when handling shapes with diagonal lines or complex structures (such as handwritten characters or natural shapes), 8-connectivity generally provides more accurate results. However, in this specific task, there was no significant practical difference observed between 4-connectivity and 8-connectivity.

2.3.7 Q7 train the (conventional) unsupervised KNN

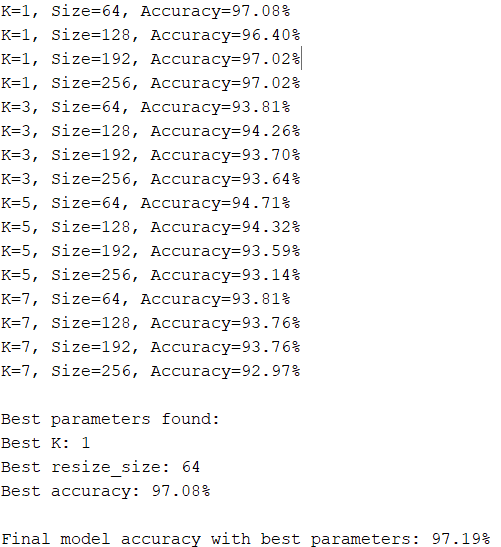
In main\_7.m, the code implements an image classification task using a k-Nearest Neighbors (k-NN) classifier. The k-NN algorithm is a widely-used machine learning method primarily applied in classification and regression tasks. Its core principle is that “similar samples have similar labels,” meaning that it predicts the category or output value of a test sample based on its distance from known samples in the feature space.

Images of each letter are read from a folder, with the grayscale data of each image flattened into a one-dimensional row vector, serving as the feature vector. The corresponding letter represented by each image is set as its label. Using cvpartition, the dataset is divided, with 75% used for training and 25% for testing. The fitcknn function is employed to create and train the k-NN classifier. Here, trainData and trainLabels are the training data and labels, and NumNeighbors = 3 specifies that the default number of neighbors used in k-NN is 3. The accuracy variable calculates the classifier's accuracy. When NumNeighbors = 3, the classifier achieved accurate classification results is **92.74%**.

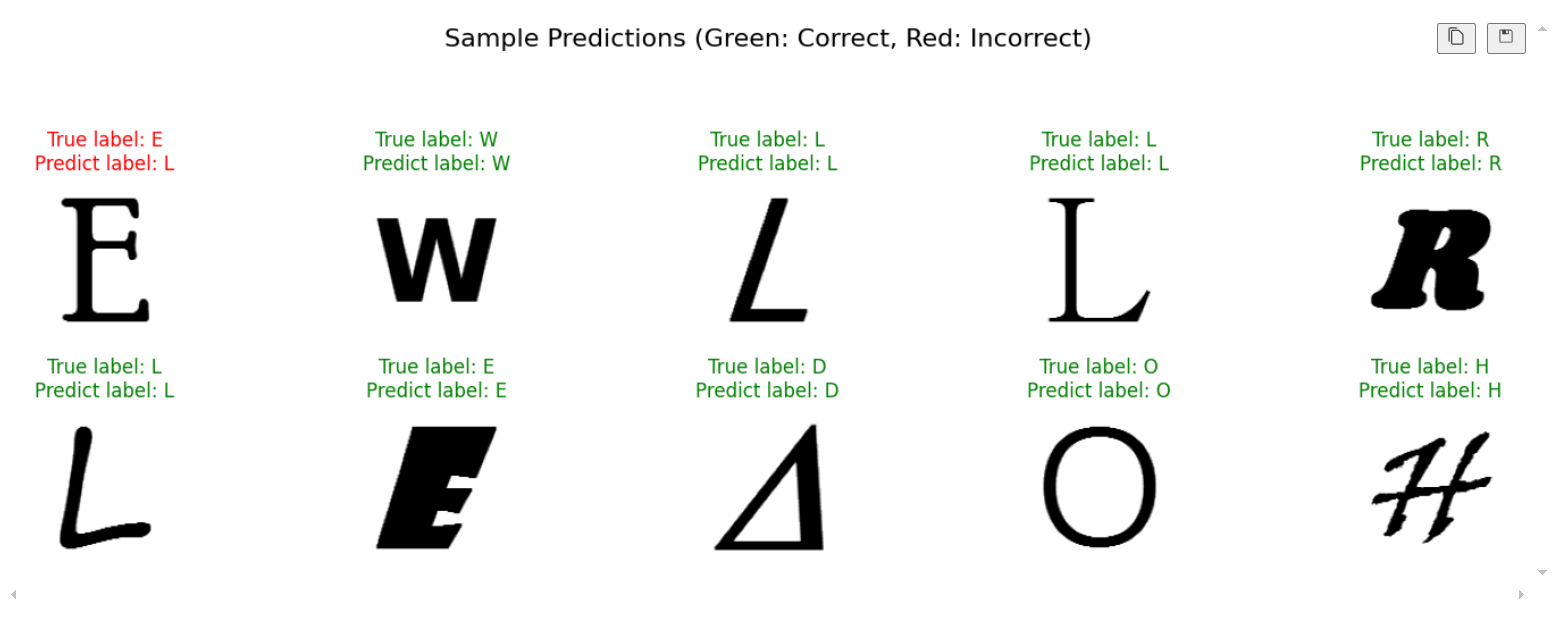


2.3.8 Q8 train the (conventional) unsupervised KNN

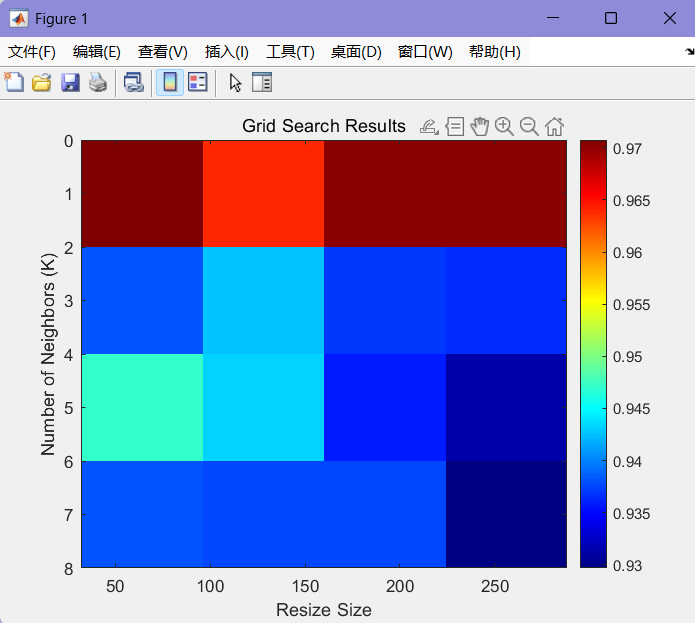
In Q8, we build upon the code from Q7 to perform grid search, experimenting with different values of k (the number of neighbors) and image resizing dimensions. The goal is to train a k-NN model to find the optimal combination of parameters. Specifically, we set the range of k to 1,3,5,7 and the resize dimensions to 64x64,128x128,196x196,256x256, with the original image dimensions being 128x128. Using these different parameter combinations, the k-NN models are trained on the training data and evaluated on the test set. The grid search helps determine the best parameter combination, and the final model's accuracy is reported.



The trained model can be utilized to recognize the image.



After visualizing the results, we obtain the following insights:



Conclusion: For the k-NN algorithm, smaller values of k (such as k = 1) often capture finer distinctions more effectively, as the prediction relies solely on the nearest neighbor. This works well for the current dataset, suggesting that the nearest neighbor is highly representative of each sample.

Additionally, an image size of 64x64 is sufficient to capture the key information in the images, while larger image sizes may introduce redundant information, leading to no significant improvement in classification performance.

Overall, the KNN model achieved very high accuracy, and the results of the grid search validate the effectiveness of the selected optimal parameters.