Computing Project Report

ME5411 Robot Vision and AI

Group 8

Zhang Binlin A0304090U

Jiang Chiheng A0304894W

Qiao Zhuoran A0304606M

Contents

Task 1

1.1. Introduction

Task 1 aims to experiment with and implement various image processing techniques on a microchip label image called **Charact2**. The goal is to preprocess the image for better visual clarity, which lays the foundation for subsequent classification tasks. Contrast enhancement techniques like contrast stretching, histogram equalization, and geometric transformations are explored to improve the readability of characters.

1.2. Description of Algorithm

1.2.1 Loading and Displaying the Image

- **Input**: The algorithm reads the original image from a BMP file using **imread** and displays it using **imshow**. This step obtains the initial quality and features of the image.
- **Output**: A visual representation of the original image is shown to compare with later transformations.

1.2.2 Monochrome Conversion

- **Description**: The image is converted from RGB to grayscale using **rgb2gray**. This simplifies the image by removing color information and retaining only intensity values which are easier to process for contrast enhancements.
- **Aim**: Simplifies subsequent processing steps and reduces computational complexity.

1.2.3 Negative Transformation

- **Formula**: New Intensity = L−1−Original Intensity, where L = 256 for an 8-bit image.
- **Description**: Each pixel's intensity is inverted, creating a negative image. Dark areas become bright and vice versa.
- **Aim**: Useful for enhancing the visibility of features that are otherwise faint in the original image.

1.2.4 Contrast Stretching

- **Description**: A piecewise linear function is applied:
  - Intensities in the range [0, 64): Scaled by 0.5.
  - Intensities in the range [64, 192): Enhanced by a factor of 1.5, shifted by 32.
  - Intensities in the range [192, 255]: Reduced by a factor of 0.5, shifted by 224.
- **Aim**: This method improves the contrast of the image by emphasizing mid-range intensities, making the characters more distinct.

1.2.5 Histogram Equalization

- **Description**: The grayscale image's histogram is equalized, redistributing pixel intensities to utilize the full intensity range. This enhances contrast uniformly, making features more visible.
- **Aim**: A clearer and more uniformly contrasted image, especially useful for images with uneven lighting.
- **Function Implementation**: The **hist_eq** function enhances the contrast of a grayscale image through histogram equalization. It first computes the histogram (**H**) by iterating over all pixel intensities, counting the occurrence of each intensity level, and adjusting for 1-based indexing. The cumulative histogram (**Hc**) is then calculated by summing the histogram values sequentially, representing the cumulative distribution of intensities. Using **Hc**, a transformation function (**T**) is created by normalizing the cumulative histogram to the range [0,255], which maps old intensity levels to new, evenly distributed levels. This transformation is applied to the input image, replacing each pixel intensity with its corresponding value from **T**, resulting in an equalized image. Finally, the output is converted back to an 8-bit unsigned integer format for standard image processing compatibility.

1.2.6 Geometric Transformation

- **Scaling**: The image is resized using bi-linear interpolation with a scale factor of 1.5.

- **Bi-linear Interpolation**: It computes the intensity value of a new pixel based on the intensities of four neighbouring pixels from the original image. Also, it ensures smooth scaling without introducing significant artifacts.
- **Aim**: Enhances image resolution while maintaining quality, making the text more readable.

1.3. Screen Captures

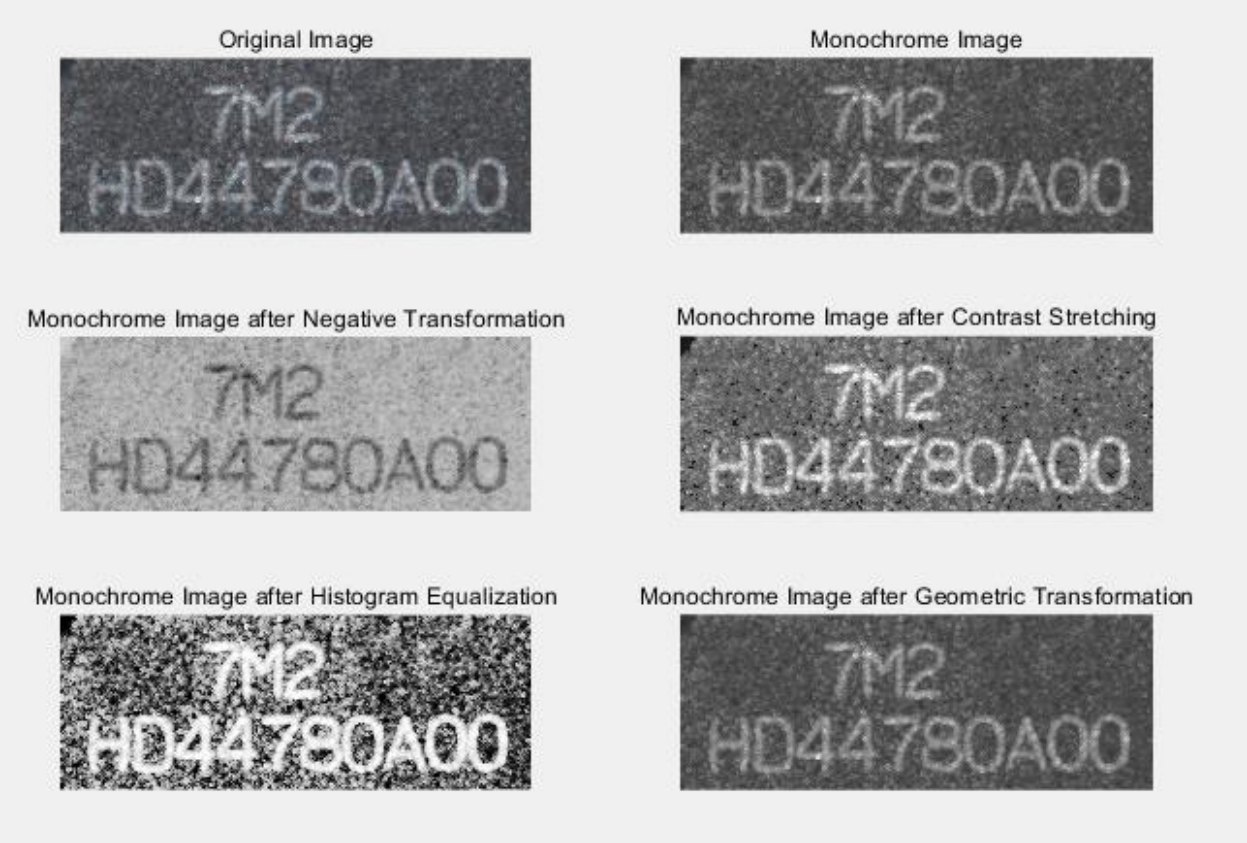

Figure 1: Comparison of different contrast enhancement methods

1.4. Discussion and Conclusion

As shown in Figure 1, negative transformation can reveal hidden details of the image and geometric transformation can improve the resolution. Besides, contrast stretching and histogram

equalization can enhance the contrast to make the characters more prominent. Compared to contract stretching, histogram equalization can realize global contrast enhancement while amplifying noise in the image.

Task 2

2.1. Introduction

Task 2 aims to reduce noise in the image and enhance its smoothness by applying spatial filters. This includes experimenting with a 5x5 averaging filter and filters of varying sizes to understand their effects on image quality. The results are compared to evaluate the effectiveness of smoothing techniques.

2.2. Description of Algorithm

2.2.1 Initialization and Input

- The code begins by clearing the workspace and loading the input image using **imread('charact2.bmp').**
- The image is then converted to a grayscale image using **rgb2gray(img)** to simplify processing.
- A grayscale colormap is set using **colormap(gray)** for better visualization.

2.2.3 Applying Averaging Filters

- **Function Implementation**: The **avg_filter** function creates a corresponding size mean filter and applies it to the input image via convolution.
- The custom **avg_filter** function is used to apply averaging filters with different sizes:
    - img3 = avg_filter(img_mono, 3): Applies a 3x3 averaging filter to the grayscale image.
    - img5 = avg_filter(img_mono, 5): Applies a 5x5 averaging filter.
    - img7 = avg_filter(img_mono, 7): Applies a 7x7 averaging filter.
- These filters calculate the mean intensity of the pixels within their respective neighbourhoods, effectively reducing noise.

2.2.4 Displaying the Results

- The original and smoothed images are displayed side-by-side using **imshow** in a single figure:
    - The original grayscale image is shown first.
    - The outputs of the 3x3, 5x5, and 7x7 filters are displayed sequentially for comparison.
- Then titles are added for clarity, indicating the filter size used.

2.3. Screen Captures

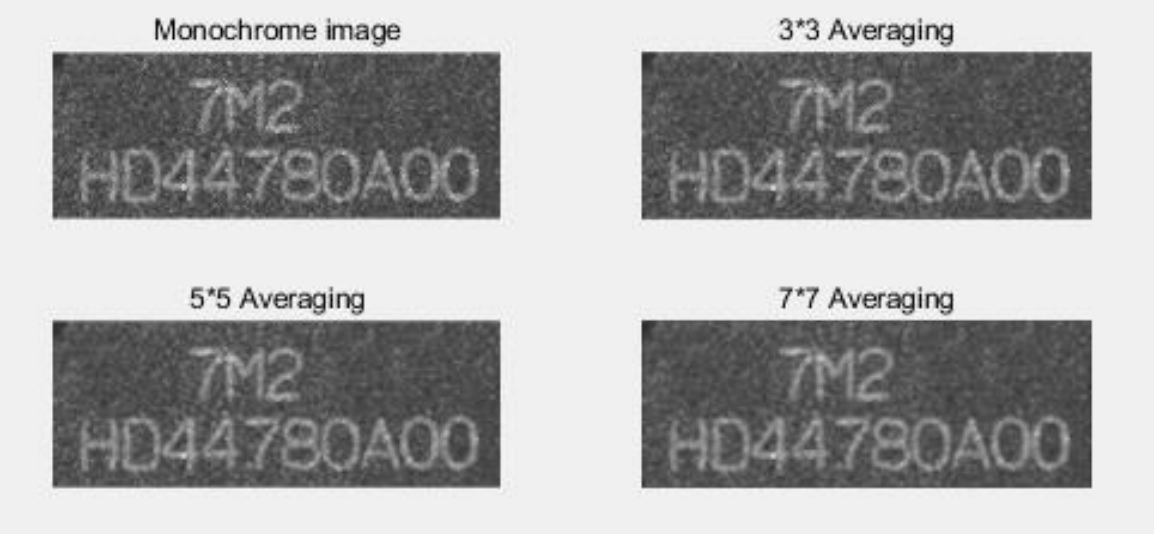

Figure 2: Result of applying different sizes of filters

2.4. Discussion and Conclusion

As Figure 2 shows, As the filter size increases, the image smoothness increases and noise is reduced, but at the cost of losing detail. A proper filter size has a good balance between denoising and retaining details. For images requiring only slight noise reduction, a smaller filter is preferable, while larger filters are useful for stronger smoothing but may be less effective in preserving important features.

Task 3

3.1. Introduction

Task 3 focuses on enhancing the edges and fine details in the image using a high-pass filter in the frequency domain. High-pass filters are designed to remove low-frequency components (smooth regions) while retaining high-frequency components (edges and details).

3.2. Description of Algorithm

3.2.1 Image Initialization

- The grayscale image is loaded and processed.
- The image size is recorded as **[row, col]** to manage further processing.

3.2.2 High-Pass Filter Design

- **Meshgrid Creation**:
    - A coordinate grid is generated using **meshgrid(u, v)** to calculate the distance of each frequency component from the center of the frequency domain.
    - U and V represent horizontal and vertical frequency indices.
- **Distance Calculation**:
    - A distance matrix D is calculated for each frequency component:

$$D(i,j) = \sqrt{u(i)^2 + v(j)^2}$$

- For better comparison, we use an ideal high-pass filter here and set the cut-off distance D_0 as 10. However, in reality, we don't often use an ideal filter since it will cause non-differentiable edges around the cut-off frequency and introduce unnatural high-frequency noise which produces distortion and artefacts.
- **Fourier Transform**: The grayscale image is transformed into the frequency domain using the 2D Fast Fourier Transform (**fft2**), with **fftshift** applied to center zero frequency.

- **Filter Application**: The high-pass filter is applied by multiplying the frequency-domain image with the filter mask.
- **Inverse Fourier Transform**: The filtered image is converted back to the spatial domain using the inverse 2D Fourier Transform (**ifft2**), and only the real part is retained.

3.2.3 Resizing and Visualization

The filtered image is cropped to its original size and displayed alongside:

- The original grayscale image.
- Frequency-domain representations before and after filtering.
- A comparison with spatial-domain methods (e.g. 5x5 averaging filter) to highlight the differences.

3.3. Screen Captures

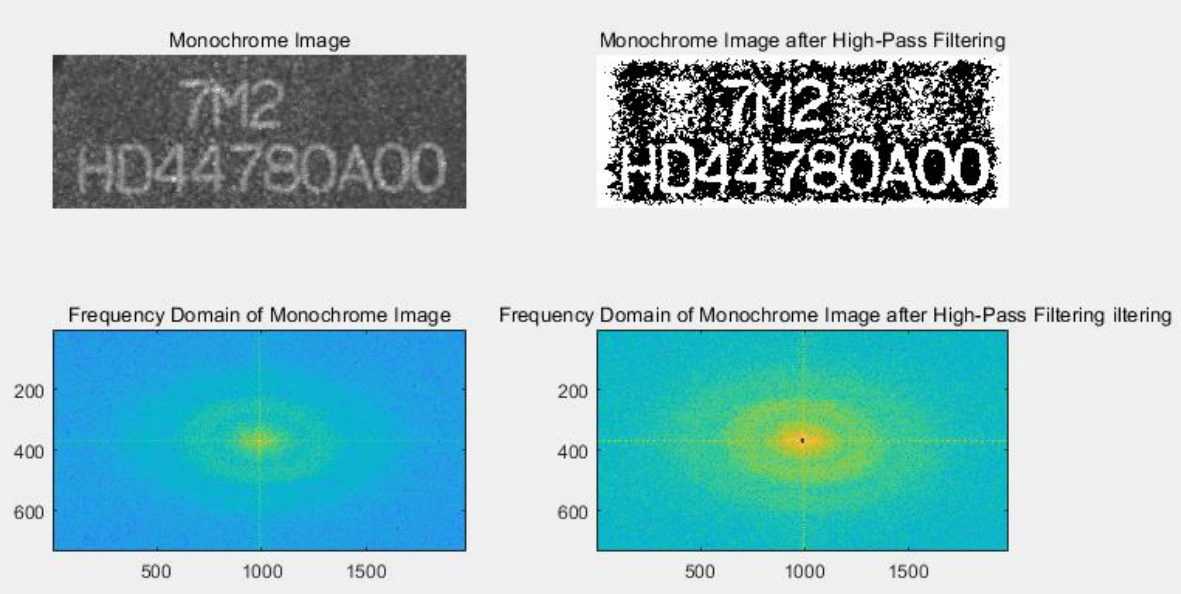

Figure 3: High-Pass Filter Result

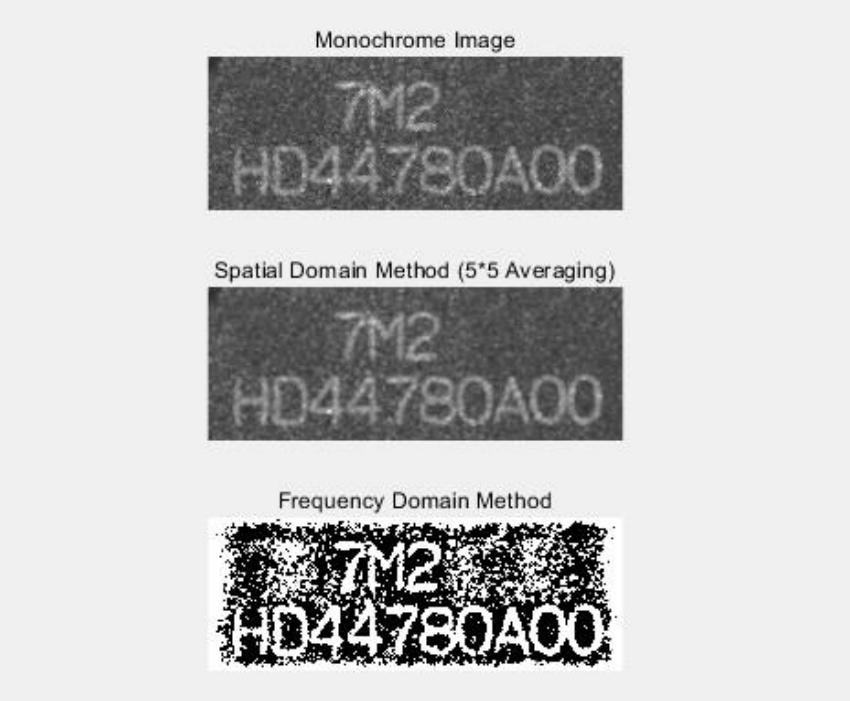

Figure 4: Comparison of frequency domain and spatial domain

3.4. Discussion and Conclusion

As Figure 3 shows, the high-pass filter enhances the sharp edges and high-frequency details and attenuates low-frequency components. It can enhance contrast around boundaries and edges for removing low-frequency noise. However, the smooth regions may appear darker or even lose information.

As Figure 4 shows, the average filters in the space domain can effectively reduce random noise but it will blur the image, especially with larger filters. While high-pass filter in the frequency domain can achieve targeted removal of noise in a certain frequency range and retain edge information, which is useful for edge detection.

Task 4

4.1. Introduction

The purpose of Task 4 is to create a sub-image containing the middle line "HD44780A00" from the grayscale image. This process isolates the region of interest for further analysis or processing, such as character recognition or segmentation.

4.2. Description of Algorithm

4.2.1 Determine Image Dimensions

- The size of the grayscale image (height and width) is obtained using the **size** function.

4.2.2 Extract the Sub-Image

- The middle section of the image is identified by dividing the height into two halves.
- The sub-image is created by selecting the rows from the middle to the bottom of the image and all the columns.

4.2.3 Display the Sub-Image

- The extracted sub-image is displayed using **imshow** with the title "Sub-image" for visualization.

4.3. Screen Captures

Figure 5: Sub-image including the middle line

4.4. Discussion

The easiest and most effective way to obtain a sub-image is to crop the original image and only save the useful part.

Task 5

5.1. Introduction

The goal of Task 5 is to convert the previously extracted sub-image into a binary image using a simple thresholding technique. This step simplifies the image by distinguishing the foreground (text) from the background, making it suitable for further tasks like contour detection or segmentation.

5.2. Description of Algorithm

5.2.1 Initialize a Binary Image

- A new image matrix (**img_bi**) with the same dimensions as the sub-image is initialized to zeros (black background).

5.2.2 Set a Threshold Value

- A threshold value (**bar = 120**) is chosen to differentiate between foreground and background pixel intensities.

5.2.3 Apply Thresholding

- Each pixel in the sub-image is iterated through:
    - If the pixel intensity is greater than the threshold, the corresponding pixel in the binary image is set to 255 (white).
    - Otherwise, it remains 0 (black).

5.2.4 Display the Binary Image

- The binary image is displayed using **imshow** with the title "Binary Image of Sub-Image."

5.3. Screen Captures

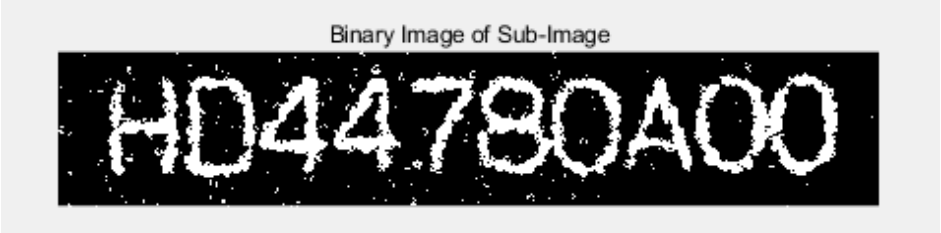

Figure 6: binary image of the sub-image

5.4. Discussion

Thresholding is the most popular way to convert a gray-scale image into a binary one. However, selecting an appropriate threshold is crucial for good results. We first observed the intensity distribution of the gray-scale image to estimate a possible threshold. Then we refined our choice via repeated experiments. When the text is immersed in the dark background, we increase the threshold. Conversely, if a significant portion of the background turned white, we lower the threshold.

Task 6

6.1. Introduction

The goal of Task 6 is to detect the outlines (edges) of characters in the image using edge detection techniques. Both Sobel and Laplacian operators are applied and compared on grayscale and binary images. This step helps in identifying sharp transitions in intensity, which correspond to edges, and prepares the image for subsequent segmentation or recognition tasks.

6.2. Description of Algorithm

6.2.1 Preprocessing

- The sub-image is preprocessed with histogram equalization (**hist_eq**) to enhance the contrast of the image for better edge detection.

6.2.2 Edge Detection with Sobel Operator

- We implemented edge detection using the Sobel operator in the function **sobel**, taking the image and threshold as inputs.
- The Sobel operator detects edges by calculating intensity gradients in both the x and y directions.
- The operator is implemented using convolution with predefined Sobel kernels for horizontal and vertical gradients. Here we use the classic Sobel kernel.
- The gradient magnitude is computed as the square root of the sum of the squared gradients in the x and y directions and is then normalized to the range [0, 1].
- A gradient threshold is applied to extract edges, with multiple thresholds tested (choose the best threshold of 0.15 for the gray-scale image).

6.2.3 Edge Detection with Laplacian Operator

- We implemented edge detection using the Laplacian operator in the function **Laplacian**, taking the image and threshold as inputs.

- The Laplacian Operator detects edges by calculating the second derivative of the image intensity, focusing on regions of rapid intensity change.
- The operator is implemented using convolution with predefined Laplacian kernels. Here we also use a classic Laplacian kernel.
- Similar to the Sobel operator, the gradient magnitude is normalized, and a proper threshold is applied after multiple tests (choose 0.07 as the best threshold for the gray-scale image).

6.2.4 Comparison of Grayscale and Binary Images

- Both Sobel and Laplacian operators are applied to grayscale and binary images to compare the differences and show the advantages of both methods.
- Since the binary image contains only two intensity values, the exact threshold for both operators is less critical, as long as it falls within a reasonable range. Therefore, we arbitrarily set it to 0.1.

6.3. Screen Captures

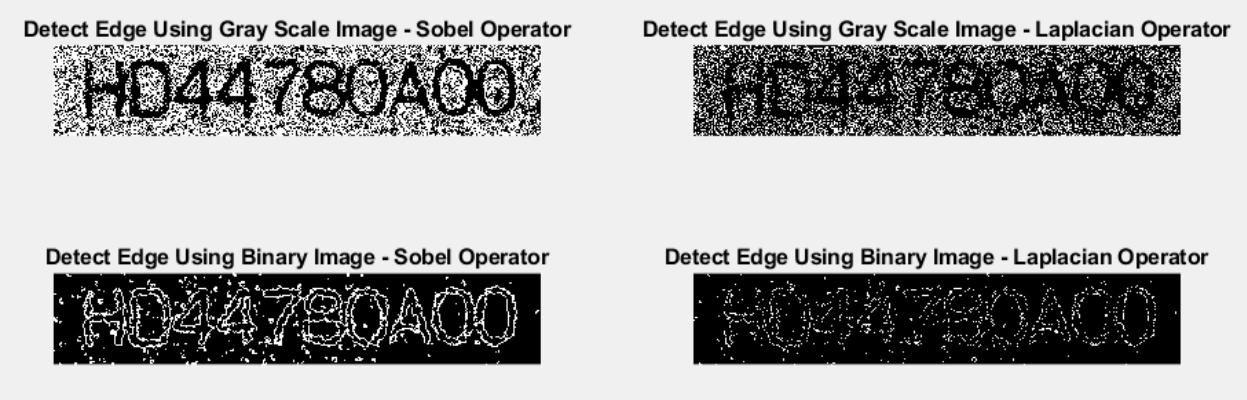

Figure 7: Result of edge detection

6.4. Discussion and Conclusion

Sobel Operators have the advantage of providing both differencing and smoothing effects, making it well-accepted in edge detection. On the other hand, Laplacian Operators are sensitive to noise but tend to work well when tackling images with sharp-intensity transitions and relatively low noise.

As shown in Figure 7, when applied to grayscale images, the Sobel Operator demonstrates better edge detection performance compared to the Laplacian operator, due to the latter's high sensitivity to noise caused by its second-order derivatives. However, when applied to binary images which have sharp intensity transitions and relatively low noise, the Laplacian operator seems to perform better than the Sobel operator.

Task 7

7.1. Introduction

The goal of Task 7 is to segment and label each character in the binary image with clear separation to proceed with further analysis. It refines the shapes of characters through morphological operations, removes irrelevant components, and performs connected component labelling to isolate and extract each character, thus facilitating the subsequent recognition task.

7.2. Description of Method

7.2.1 Preprocessing

- To facilitate the segmentation task, we preprocess the images using image eroding and dilating.
- The **imerode** function uses a rectangular structuring element [5, 1]. We use a rectangle shape to erode for it can eliminate useless horizontal details, helping to reduce the horizontal overlaps between characters.
- The **imdilate** function with a disk-shaped structuring element of radius 3 is used to restore the smooth shapes of the characters, filling breaks caused by the erosion process.
- Batch point processing is also used to cut off the unwanted connections between the characters.
- The **bwareaopen** function eliminates small, connected regions with fewer than 196 pixels, effectively removing irrelevant noise and making the image cleaner.

7.2.2 Labeling Connected Components

- The **bwlabel** function is applied to the binary image to distinguish different regions. This function detects connected components and returns both the total number of regions and a label matrix with the same dimensions as the original image, where each connected region is assigned a unique integer label.

- The **regionprops** function is used to extract the top-left coordinates (x, y) and the width and height of the bounding box (BoundingBox) for each connected region. This provides the necessary positional information for subsequent cropping and segmentation.

7.2.3 Cropping Characters

Using a loop, the bounding box of each connected region is extracted, and the **imcrop** function is applied to crop the characters based on these bounding boxes. This method isolates and segments each character region from the binary image.

7.3. Screen Captures

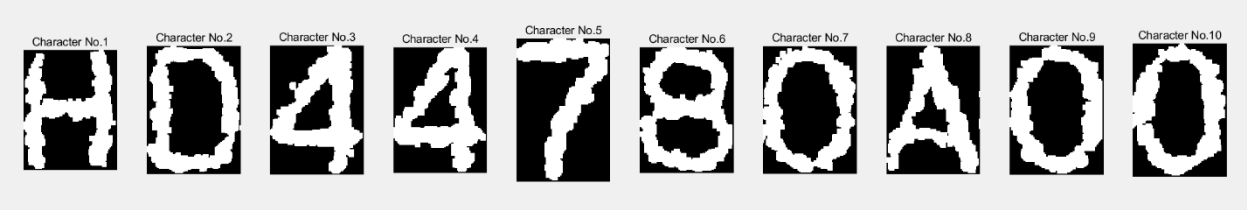

Figure 8: Successful image segmentation

7.4 Discussion

Initially, we applied the edge detection directly, but we encountered excessive noise. So, then we added **bwareaopen** function to remove small, irrelevant connected regions. However, we still found some unwanted connections between different characters. To address this, we employed **imerode** and **imdilate** functions to reduce these unwanted connections and meanwhile refine the shapes of the characters. We experimented with different structuring elements like disk, square, rectangle and line to figure out their effects of eroding and dilating. After many attempts, we came out with the result as Figure 9 shows, where the shapes of characters are refined but unwanted connections still exist. Finally, we used batch point processing to eliminate the unwanted connections.

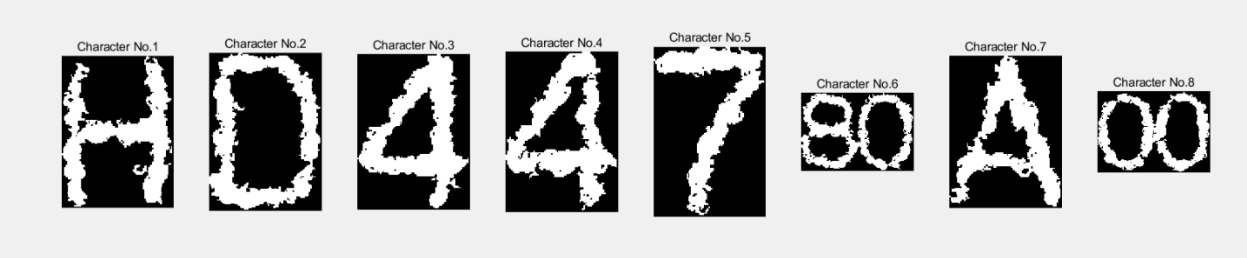

Figure 9: Unsuccessful image segmentation ("80" and "00")

Task 8

8.1. CNN Methods

8.1.1. Introduction of Task 8.1

The goal of Task 8.1 is to design a CNN network to classify each character in the image. The network will consist of the following kinds of layers: the input layer for loading the images, convolution layers for feature extraction, pooling layers for dimensionality reduction, and fully connected layers followed by a Softmax layer for classification.

8.1.2. Description of Method

The structure of our CNN network is illustrated in Figure 10. Apart from the input layer, the fully connected layer, and the output layer, our CNN network is structured into eight blocks. The first seven blocks are convolution-pooling layers, each consisting of a convolutional layer, a batch normalization layer, a ReLU activation function, and a max pooling layer. These blocks gradually increase the number of feature maps while reducing spatial resolution. The final block only includes a convolution layer to further extract high-level features.

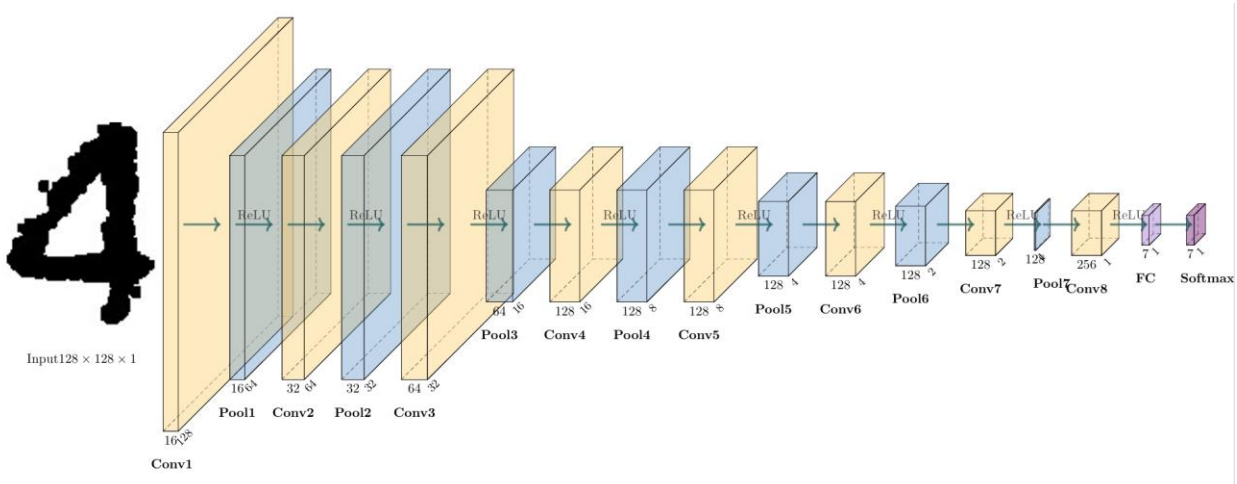

Figure 10: Our CNN Network Structure

8.1.2.1. Dataset Preparation and Input Layer

First of all, the dataset is divided into two parts: 75% is used as the training set, and 25% is allocated as the validation set. Next, the input image size is fixed as 128 x 128 and 1 color channel for gray-scale images.

8.1.2.2. Lower-Level Feature Extraction (Block 1 to 3)

The first three blocks extract the lower-level features of the image. The first block with 16 channels preliminarily captures lower-level features such as edges and textures. The second block with 32 channels and the third block with 64 channels begins to extract local shape and contour features. Simultaneously, the max-pooling layers progressively reduce the spatial dimensions of the feature maps from 128×128 down to 16×16, focusing more on local features.

8.1.2.3. Higher-Level Feature Extraction (Block 3 to 8)

The fourth to seventh blocks with 128 channels further extract deeper features of the image. The feature maps have 128 dimensions while the spatial dimension continues to decrease from 16×16 to 1×1. As shown in Figure 10, the last layer has a minimal spatial dimension and focuses more on feature abstraction for global information.

8.1.2.4. Character classification (FC and SoftMax layers)

The CNN network ends with a fully connected layer (with 7 classes) and a SoftMax layer. This is the decision-making part of the network, converting the previously extracted features into final classification predictions.

The fully connected layer performs a linear transformation by applying a weight matrix and a bias term to the output. The SoftMax layer converts the output of the neural network into a normalized probability distribution, and the label with maximum probability is regarded as the prediction.

8.1.3. Discussion

When designing the CNN network, we first implemented a relatively simple structure with only three convolution-pooling layers. It came out with a high accuracy in the validation dataset but a poor accuracy in the test sets (the characters in the sub-image). After excluding the possibility of overfitting, we thought the reason is that the network is too simple so it has poor robustness to non-standard or distorted characters. Therefore, we added the number of convolution-pooling layers gradually from 3 to 8 and followed the principle of adding channel number gradually. It turned out that the accuracy of character recognition increased from 30% with 3 layers to 70% with 8 layers. We also tried to change the kernel size and rearrange the channels of different layers but resulted in using a 3*3 kernel and layers with 16, 32, and 64 channels subsequently and four layers with 128 channels followed by the last layer with 256 channels for its best performance.

To further improve the accuracy of the CNN network classification, we adjusted the hyperparameters like optimizer, batch size and learning rates. After experiments, we found that the Adam optimizer works better than the SGDM optimizer. The most suitable batch size is 256 and the learning rate is 0.005. With a too-small batch size, the learning speed is too slow and may cause large gradient noise. While a too-large batch size may cause an unstable learning process and overfitting. With a too-small learning rate, the learning process is too slow and easy to stuck in the local optimum. While with a too large learning rate, the learning process is harder to converge well. Worthy to mention, we encountered a situation where two learning rates resulted in the same accuracy. To choose the optimal one, we use different datasets (same characters with different extents of preprocessing) to test different models.

8.2. Non-CNN Method (SVM Methods)

8.2.1. Introduction of task8.2

The goal of Task 8.2 is to design a classification system using a non-CNN-based method to classify each character in the images. We use SVM here which is more explainable than neural networks and has a simpler structure than CNN.

8.2.2. Description of Method

Our SVM model extracts HOG features first and then uses a linear kernel to train.

8.2.2.1. Preparation of dataset and setting parameters

Similar to Task 8.1, the dataset is split into 75% for training and 25% for testing. The size of the HOG feature cell is set as [8, 8]. A linear kernel function is selected for the SVM classifier to perform the classification task.

8.2.2.2. Extract HOG features from the dataset

The process begins by iterating through every image in both the training and testing datasets. Each image is read using the **readimage** function, and the **extractHOGFeatures** function is then applied to extract HOG features by analyzing gradient directions to capture texture information. These features are used to construct a feature matrix, with each row representing the HOG feature vector of an individual image.

8.2.2.3. SVM model training

The training process begins with **trainFeature**, a two-dimensional feature matrix where each row represents the HOG features of a training sample, and **trainLabel**, which provides the corresponding labels for these samples. Given that character classification is often relatively simple and may be linearly separable, a linear kernel is well-suited for handling such data efficiently. Therefore, the SVM is configured with a linear kernel. The **fitcecoc** function is employed to train a multi-class SVM model, utilizing a One-vs-All strategy. Under this approach, a binary SVM classifier is trained for each class to determine whether a given sample belongs to a certain class or not.

8.2.2.4. Calculate the accuracy of the test set

After the training step in 8.2.2.3, The trained SVM model is used to predict the labels of the test feature matrix. The **predict** function is called to return the predicted labels for each test sample. These predicted labels are then compared with the actual labels. The number of correctly predicted samples is counted and divided by the total number of test labels to calculate the classification accuracy.

8.2.3. Discussion

We also tried the RBF kernel for SVM which is well-known for classifying characters with complex shapes and structure. However, the validation accuracy is only 33%. After analysis, we thought it was because our task is much simpler and using RBF kernel will make the model too complicated and thus cause overfitting.

8.3. Results

8.3.1. Test Results without Image Preprocessing

Test the best CNN models and SVM models without preprocessing the image. The accuracy of the CNN model is 70%, and the accuracy of SVM is 50%.

8.3.1.1. Result of CNN

In this task, we use the best CNN models (3*3 kernel with 16,32,64,128,128,128,128,128,256 channels, with Adam optimizer, a batch size of 256 and a learning rate of 0.005) to recognize the characters. It reaches an accuracy of 70%. And only mistake 8 as H and 0 as D which are quite similar. It shows the high robustness to distortion of the CNN model.

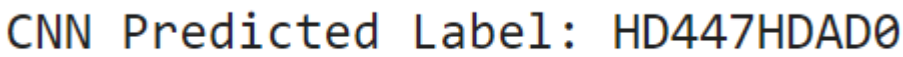

Figure 11: Test results of CNN without image preprocessing (70% accuracy)

8.3.1.2. Result of SVM

The accuracy of SVM model is 50% which is lower than the fine-tuned CNN models. And it also results in some weird mistakes like mistake 0 as 7. There are two possible reasons for this. First, the performance of SVM heavily depends on the feature extraction method. In this project, we utilized HOG features, which effectively capture local gradient information. However, in more complex global contexts, HOG's descriptive capabilities are somewhat limited. Additionally, we employed a linear kernel function with the expectation of achieving linear separation of the data.

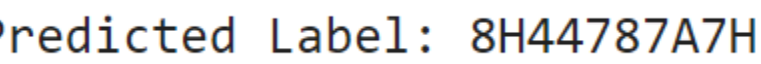

Figure 12: Test results of SVM without image preprocessing (50% accuracy)

8.3.2. Test Results with Image Preprocessing

After preprocessing, the characteristics of each character are more distinct.

Figure 13: Filling the edges of the test image

8.3.2.1. Result of CNN

After preprocessing, the accuracy of CNN reached an astonishing 100%. After passing through multiple convolution layers, the CNN network is well capable of capturing subtle non-linear features. Additionally, with the adjustments shown in Figure 13, the differences between "0" and "8" can be easily distinguished by the CNN network.

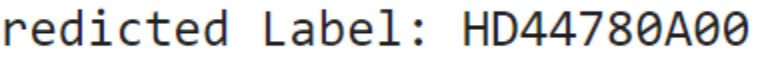

Figure 14: Test results of best CNN with image preprocessing available (100% accuracy)

To further compare the results of CNN, we also used the CNN model without fine-tuning (3*3 kernel with 16,32,64,128,128,128,128,128,256 channels, with adam optimizer, a batch size of 128 and a learning rate of 0.001) to test. It only has 80% accuracy, which is a bit lower.

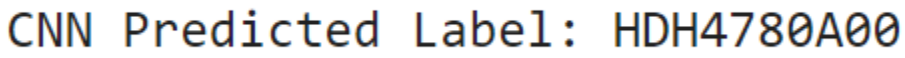

Figure 15: CNN model without fine-tuning recognition result

8.3.2.2. Result of SVM

The accuracy of the SVM method also reached 80%. Its performance is also improved but not as much as CNN models.

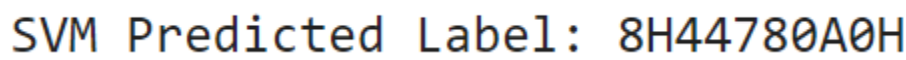

Figure 16: Test results of SVM with image preprocessing available (80% accuracy)

8.4. Discussion

8.4.1. Discussion about the Experiment Result

As the comparison mentioned above, we can see preprocessing has a positive effect on character recognition.

For model comparison, we can find that after fine-tuning CNN can perform better than the SVM model. But without fine-tuning, CNN is not always better than SVM methods. Since SVM is much easier to implement, when the task doesn't require a very high accuracy, using SVM is adequate and efficient. For tasks requiring high accuracy, a fine-tuned CNN model is better.

8.4.2. Data Scarcity

For the dataset provided in Task 8, the primary issue lies in the insufficient quantity of data (only 762 samples about 0), which hinders the neural network's robustness. Although accuracy reached 100% after augmenting and padding the images, both SVM and CNN performed poorly under the original conditions.

8.4.3. Inaccurate Datasets

In the training dataset provided for Task 8, the limited data quantity is further compounded by the presence of low-quality samples. These cases directly impact the accuracy, such as misclassifying "D" as "1" or "0" or misjudging "H" as "1" due to overly narrow gaps, as shown in Figure 17.

Figure 18: Inaccurate characters (Training set)