National University of Singapore

ME5411 Robot Vision and AI

CA – AY23/24 Semester 1

Computing Project

CA Group 17

Group members:

Mo Yin Chung (A0285214H)

Min Anse (A0285307B)

Ye Xiaohui (A0285243A)

1. Introduction

In robotic design and development, image processing and artificial intelligence are one of the significant research areas. It allows the robot to perform image pre-processing, analysis, recognition and so on. This project will be mainly separated into two sections, the image processing and learning-based machine vision. The first part included implementation of image display, smoothing, sub-image cropping, binary image generation, outlining and segmentation. While the second part concentrated on characters classification using CNN-based and SVM-based methods. The effectiveness and efficiency of two approaches were compared and analyzed. Additionally, with varying pre-processing of data image, the sensitivity of our approaches were tested.

2. Part 1: Image Processing

2.1 Task 1: Display original image

Figure 1 showed a JPG image of a label on a microchip displayed, with size 367×990 colored image, through MATLAB, which was provided by the ME5411 teaching team for analysis, processing, and classification.

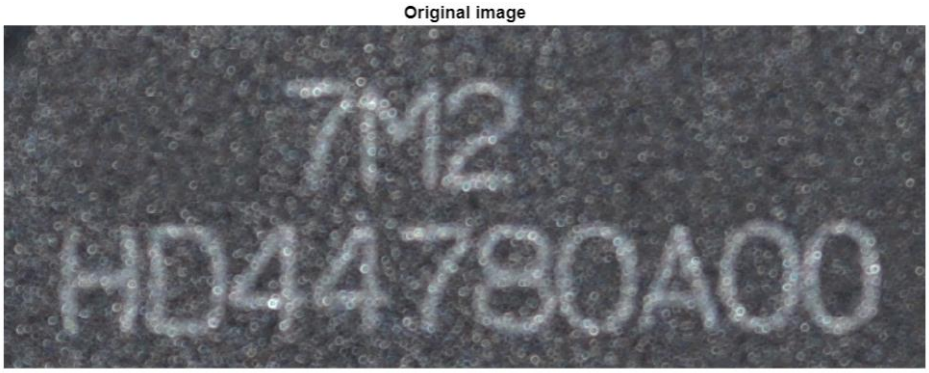


Figure 1, image of label on a microchip displayed using MATLAB

Displaying the original image with MATLAB can be achieved by *imread()* and *imshow()* function.

2.2 Task 2: Averaging mask and rotating mask

In image processing, averaging mask is a type of spatial filter commonly used for smoothing or blurring an image. It removes small details or noise while preserving and extracting the overall structure and important features.

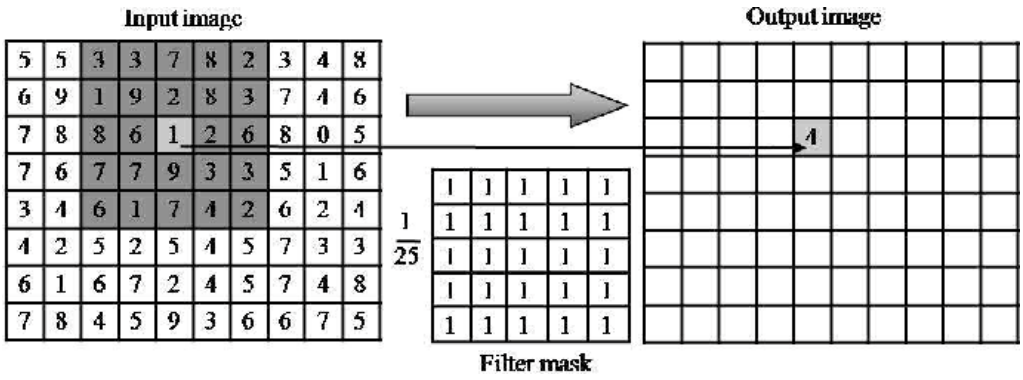
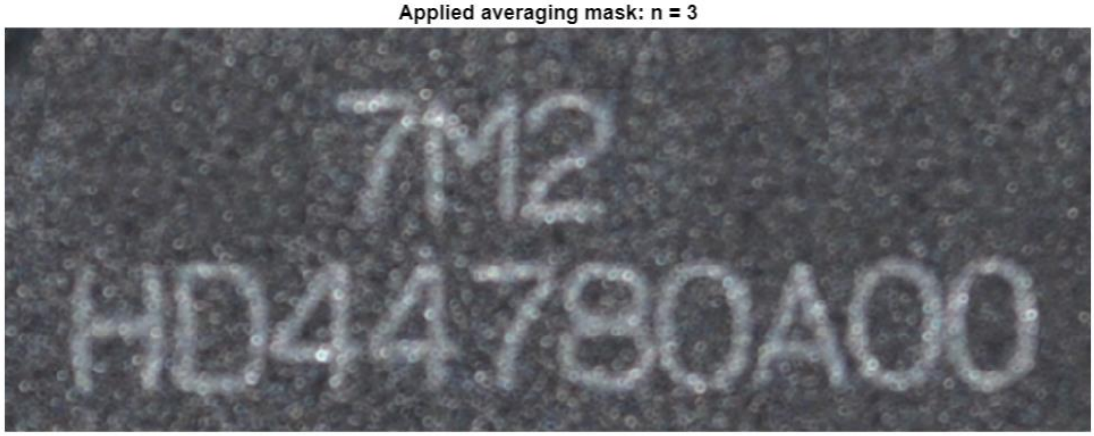


Figure 2, convolute averaging mask with a kernel size $n = 5$ *[1]*

Figure 2 depicts the application of a convolute averaging mask with kernel size n = 5, which the target pixel is located at the center of the kernel. So that by taking the average of the 5×5 mask's brightness, we assign the target pixel with this average value and then move on and shift the mask to the next target pixel. In our algorithm, we first setup an identity matrix (mask) with a kernel size n (i.e. the mask is $n \times n$), which could be varies to test the filtering effect, using the function *ones()* and multiply it by $\frac{1}{n^2}$. After that, the *imfilter()* function is used to apply the averaging mask and slide it over the original image. By setting the padding option as "symmetric", the input array values outside the bounds of the array are computed by mirror-reflecting it across the array border.



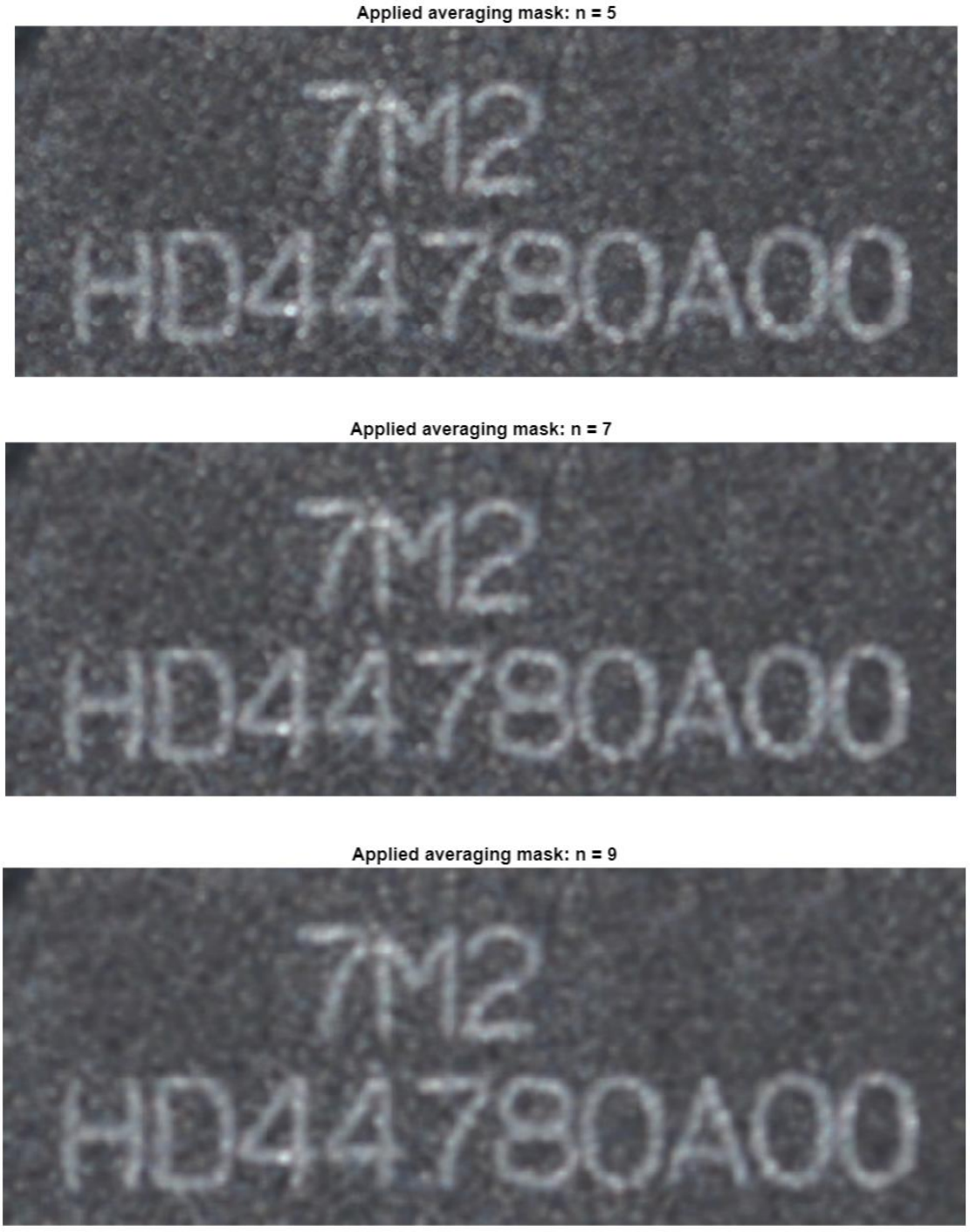


Figure 3, effect of different averaging mask size

Figure 3 gives the smoothening effect of averaging masks $n = 3, 5, 7, 9$. It shows a more significant effect of diminishing image noise when the averaging mask size increases. Meanwhile,

it also blurs the details of the image, which can be explained by the easier alteration caused by neighborhood around the target pixel.

Apart from utilizing the convolute averaging mask, the averaging technique can also be applied using the rotating mask.

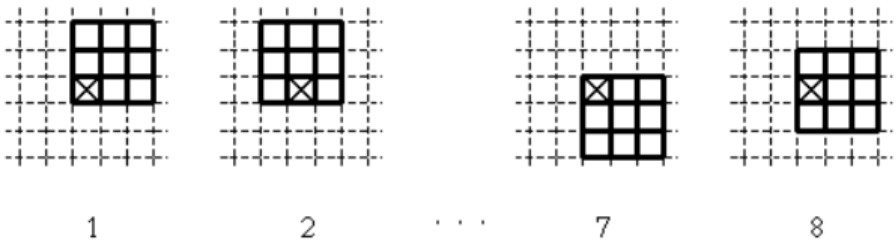


Figure 4, 8 possible rotations of a 3 × 3 mask in a 5 × 5 neighborhood

Figure 4 demonstrates the working principle of averaging using a rotating mask. It gives an instance with kernel size equal to 3 and neighborhood size equal to 5. The target pixel is marked by a cross at the center of the 5 × 5 neighborhood. So that we will assign the result brightness into this pixel. The relationship between the mask size and the neighborhood size is given by equation 1, which $S_{neighbor}$ is the size of neighborhood and n is the mask size. For example, if $n = 5$, the neighborhood will be 9 × 9.

$$S_{neighbor} = 2n - 1 \tag{1}$$

$$\sigma^2 = \frac{1}{n}\sum_{(i,j)\in R}\left(g(i,j) - \frac{1}{n}\sum_{(i,j)\in R} g(i,j)\right)^2 = \frac{1}{n}\left[\sum_{(i,j)\in R}\big(g(i,j)\big)^2 - \frac{1}{n}\left(\sum_{(i,j)\in R} g(i,j)\right)^2\right] \tag{2}$$

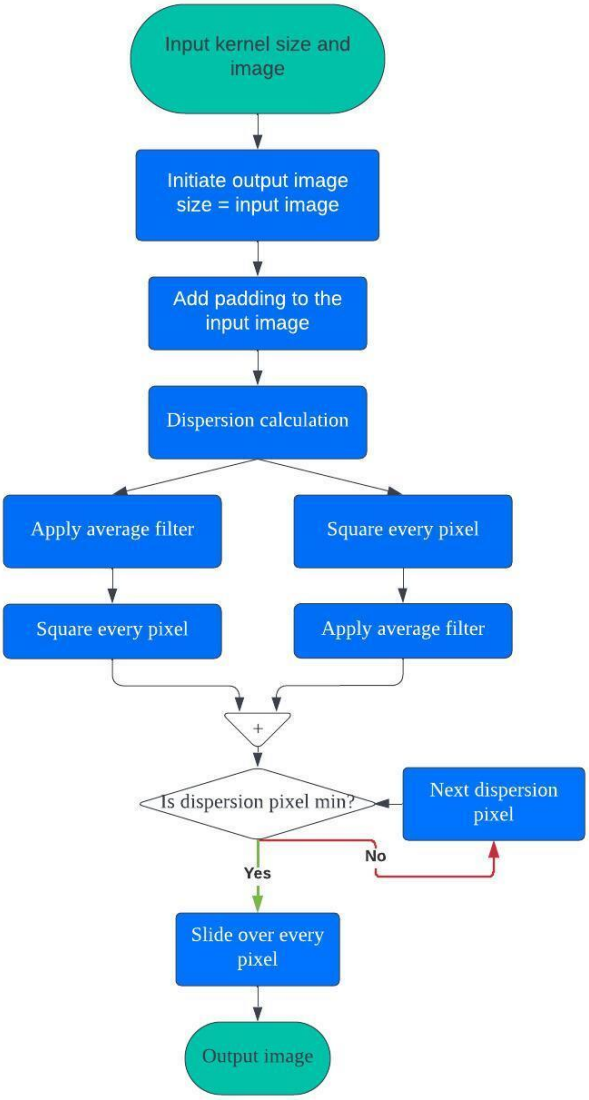


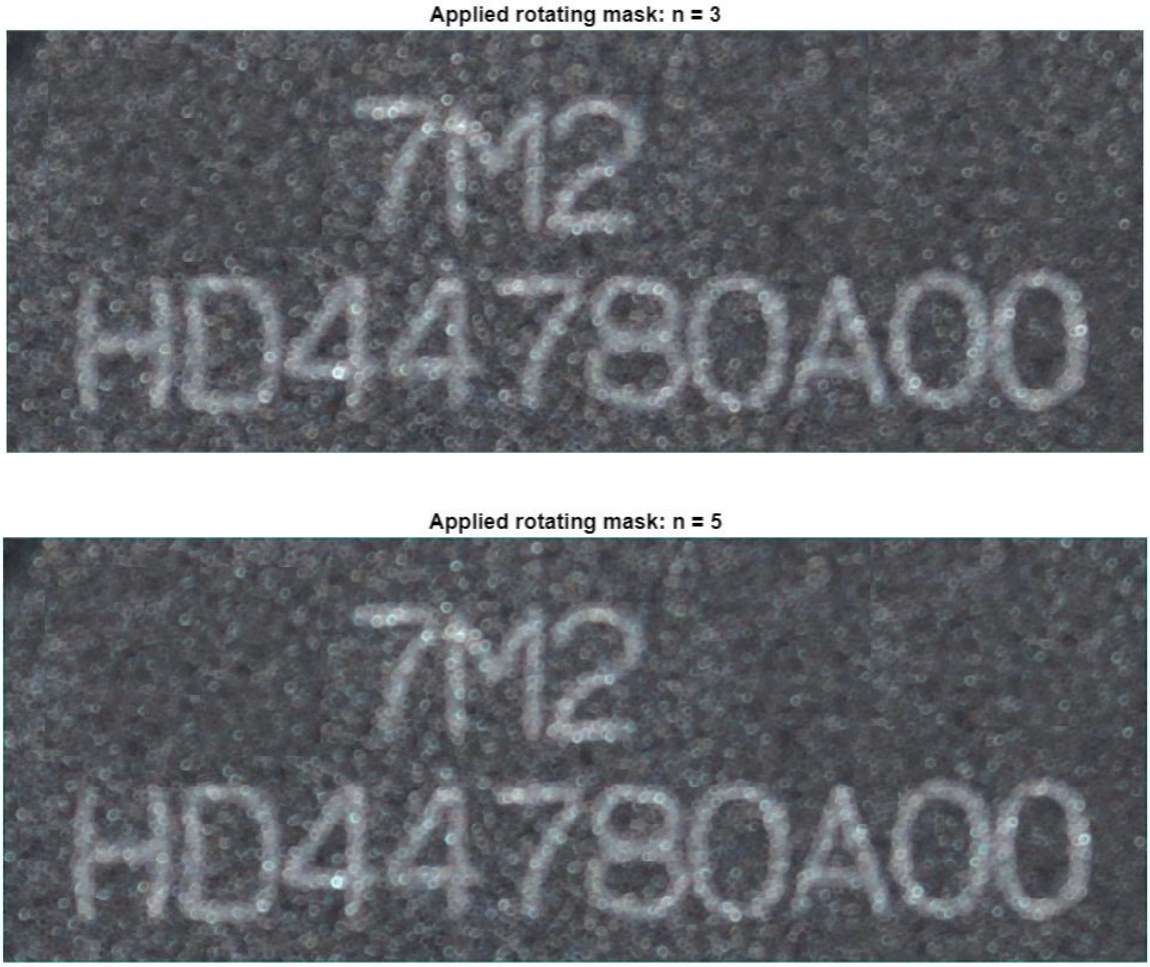
Figure 5, flowchart of rotating mask algorithm

With the mask rotating about the it, the dispersion of each mask position can be calculated with equation 2, by comparing the dispersion values among all the possible mask rotations, the mask with minimum dispersion was chosen. Its average brightness was then computed and assigned to the target pixel. Repeatedly implemented this operation resulted in a smoother output image.

As there is no built-in function for averaging with rotating mask. We had to implement the algorithm manually by first initiate the output image with zeros brightness and add padding to the original image, i.e. the padding size is by rounding down half of the kernel size (e.g. kernel size

$= 5, padding = round\ down\left(\frac{5}{2}\right) = 2$). The overall flowchart of the rotating mask algorithm is given by Figure 5. We assume the padding at the boundaries of the image to be zero brightness. After that, we can perform the computation of the dispersion map by implementing equation 2, which could be understood as the "average of the image brightness's square" minus the "square of the average image brightness". By selecting the center pixel of the mask with the smallest dispersion, we replace the target pixel with it. Finally, it was iteratively processed pixel by pixel.

The results of the rotating mask application with different kernel size are shown in Figure 6.



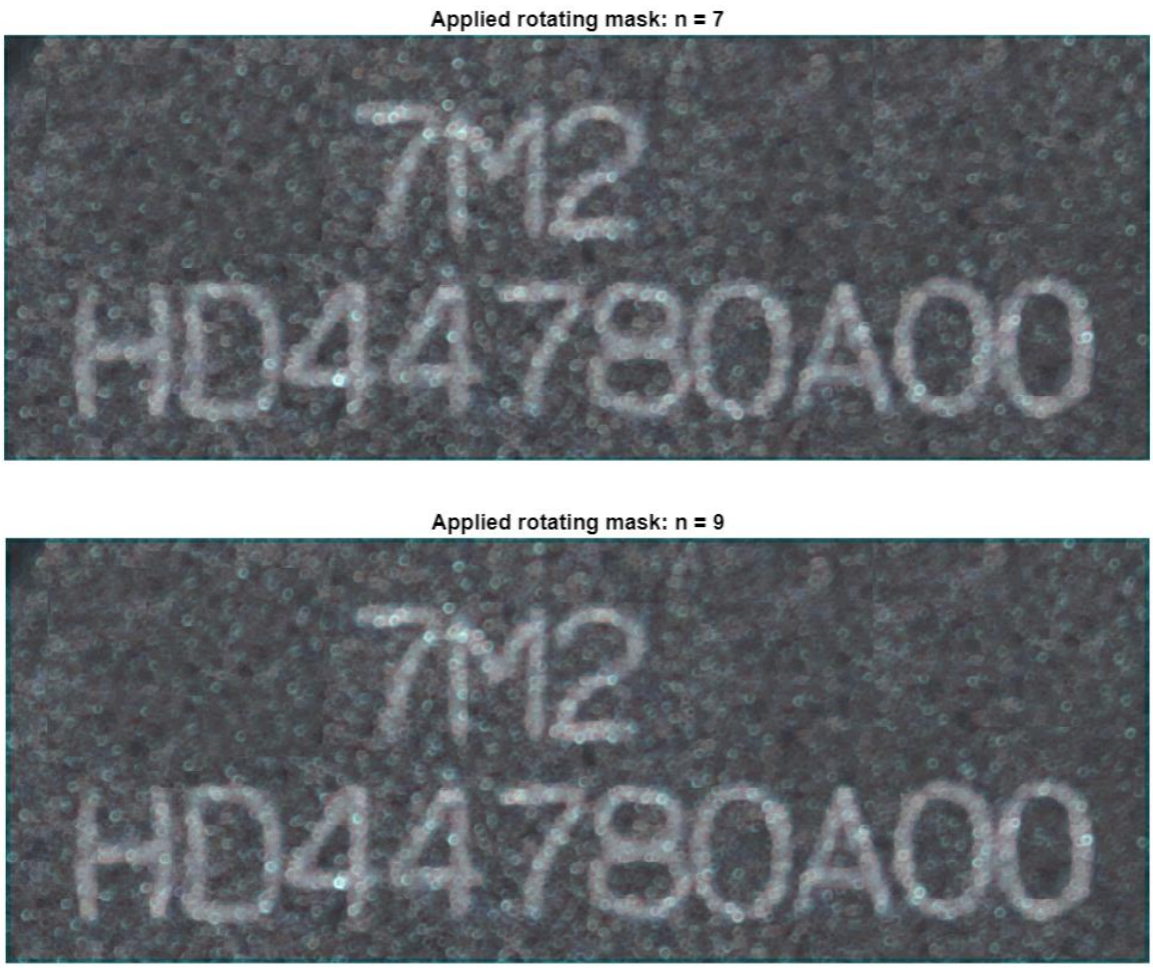


Figure 6, effect of applying rotating mask with different mask size

Figure 6 gives the results of rotating mask size $n = 3, 5, 7, 9$. The outcomes show that the rotating mask with larger mask size has a more significant effect on smoothing the image without losing image details. Additionally, when compared to the averaging filter, rotating mask has a less evident blurring effect with increasing mask size. It could be explained by different mechanisms they adopted respectively, which averaging filter takes consider of all the elements within the mask size to alter the target pixel, while the rotating mask directly replaces the target pixel with the lowest dispersion mask's center pixel.

On the other hand, in terms of computation time. Rotating mask approach is more time-consuming compared to averaging filter. It is because the operation required computing an extra dispersion map, comparing to find the lowest dispersion mask, tracing back to the corresponding mask element and replacement. As a trade-off, it has better quality in removing the image noise.

2.3 Task 3: Create sub-image

In this section, a sub-image was created by extracting the line “HD44780A00” from the original image. We can simply crop the image by half horizontally to obtain the sub-image shown in Figure 7. To generate the binary image of the sub-image in task 4, we had also smoothen the sub-image with the averaging filter and rotating mask we achieved in previous section, which is given in Figure 8.

Figure 7, sub-image of the line “HD44780A00” before smoothing

Figure 8, sub-image of the line “HD44780A00” after smoothing

2.4 Task 4: Binary image generation

We can first convert the original image to a gray-scale image and then the image histogram, shown in Figure 9, can then be generated for threshold selection.

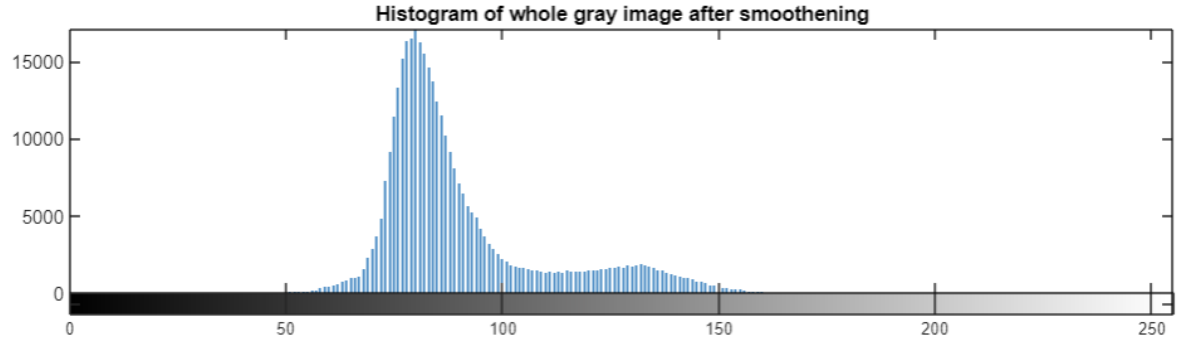


Figure 9, image histogram of the gray image after smoothening

With the smoothening operation, the histogram becomes smoother and more continuous. It was observed that there are two peaks, one at around brightness = 80 and 130. Therefore, we need to test and set the threshold between these two peaks.

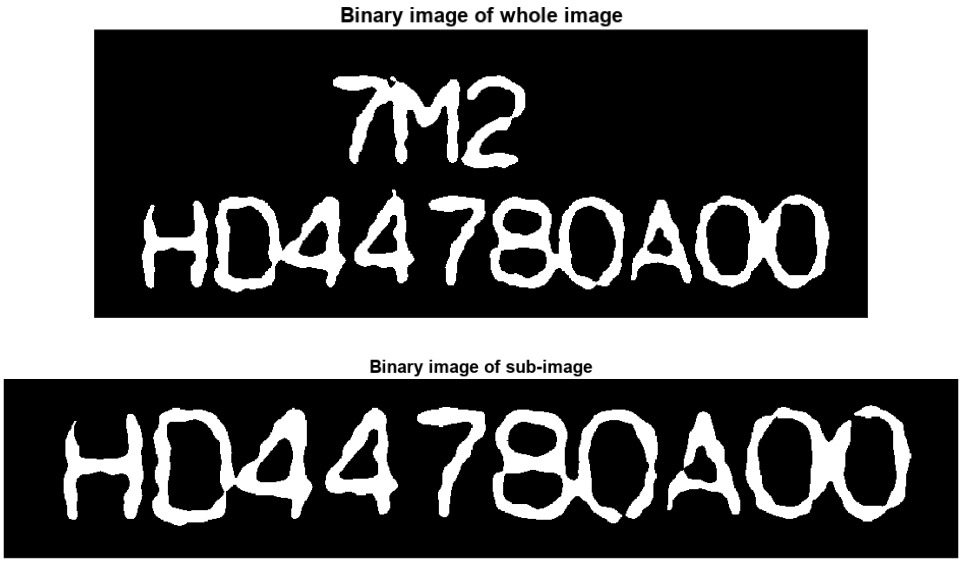


Figure 10, binary image of the whole image and sub-image

Figure 10 displays the binary image of both the whole image and sub-image. It is generated by using the built-in function *imbinarize()*, which convert a gray image into a binary image with a input threshold. The threshold used in Figure *10* are 0.455 and 0.47 (ranging from 0 to 1) for whole image and sub-image.

2.5 Task 5: Outline determination

Flowchart for task 5：

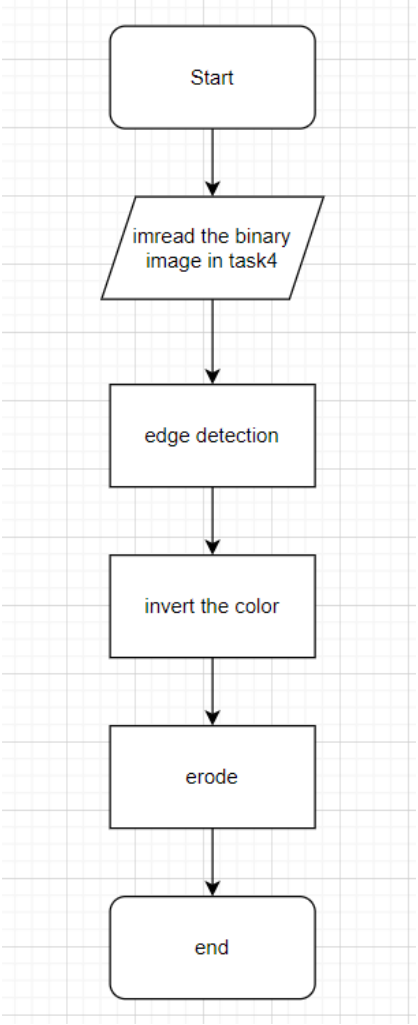


Figure 11, flowchart for task 5

Task 5 requires the determination of the outlines of the characters within the image, which means we need to accurately determine the edges of the characters and extract the edges of the characters. In task 4, we get the binary image as shown below.

Figure 12, binary image of whole image

After obtaining this binary image, we proceed with edge detection using the Sobel operator to highlight the edges. The Sobel operator is a filter used for edge detection, which emphasizes the edges in an image by calculating the spatial gradient of the image's brightness. It is highly effective in marking out the edges during this image processing phase. The extracted edges are shown below.

Figure 13, outlines of characters

We can invert the colors of the background and edges to make the edges more distinct and visible, as shown in the following figure.

Figure 14, outlines of characters (Inverted)

Subsequently, to emphasize these edges, we employed the process of erosion. Erosion helps in making the edges in the image more pronounced and isolated, assisting in the removal of small

impurities and enhancing the clarity of the key features. The resulting clear edges are depicted in the figure below.

Figure 15, outlines of characters (Bolded)

2.6 Task 6: Image segmentation and labeling

Flowchart for task 6:

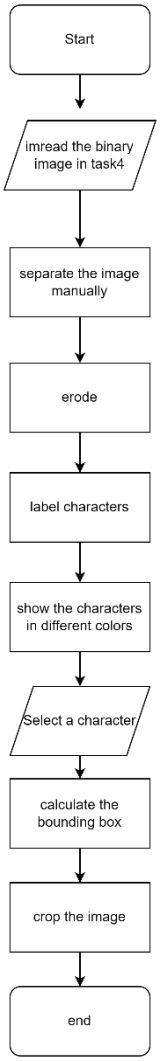


Figure 16, flowchart for task 6

Task 6 requires us to segment the image and label each character individually. We observed that these characters are tightly connected; therefore, our first step was to manually separate them.

Figure 17, Binary image after manual segmentation

After completing the manual segmentation, we applied an erosion process to the image. This erosion was designed to separate connected characters by thinning them, thereby facilitating subsequent tasks such as character segmentation and recognition. It also helped in eliminating some noise and small artifacts. Figure 18 displays the outcome following the erosion.

Figure 18, Binary image after erosion

As shown in Figure 18, each character has been effectively segmented. Moving forward, we will proceed to label these characters and fill them with various colors for distinct visualization, as shown in Figure 19.

Figure 19, Segmented image, each object is shown using a random color

We have now labeled each character in the image. If we wish to focus on a specific character, we first calculate the bounding box of the labeled characters. Then, we select the character of interest. For instance, as shown in the figure below, we have chosen the character "0".

Figure 20, Character “0”

Next, we can display this character within a 128×128 pixels image. To achieve this, we first crop out the character. Then, we center it within a 128×128 pixels black image. This process results in the figure shown below.

Figure 21, Character in 128*128 pixels

3. Part 2: Optical Character Recognition Model

3.1. Problem Introduction

The objective of the 2nd part of the report is to explore and develop two distinct models for Optical Character Recognition (OCR): one based on Convolutional Neural Networks (CNN) and the other utilizing Support Vector Machines (SVM). Both models are implemented in MATLAB and focus on identifying individual characters from BMP images of labels on microchips, i.e. (HD44780A00).

Subtask 1 focuses on using CNN to detect characters of BMP images. CNNs stand out for their ability to autonomously learn and identify critical features within images, bypassing the need for manual feature extraction. The networks of CNN consist of multiple layers of neurons that convolve and pool to extract hierarchical features from input data. This end-to-end training enables the model to take raw images as input and perform classification directly, significantly enhancing the speed and efficiency of character recognition. Subtask 2 utilizes the SVM-based model to carry out the OCR task. This approach is tailored for datasets comprising binary images, making it well-suited for the dataset given. The Histogram of Oriented Gradients (HOG) technique is employed to extract informative features from the images, which are then classified using SVM. This method is especially effective in high-dimensional spaces and for handling complex feature sets, making it a viable option for scenarios where computational resources are limited or datasets are smaller and more defined. SVM represents a more traditional, yet robust method of OCR, relying on manual feature extraction and offering a simpler, less resource-intensive alternative to CNN [2]. Sub-task 3 involves reporting and comparing the results obtained from the CNN and SVM models. This will entail a critical assessment of their effectiveness and efficiency, along with an analysis of the reasons behind any observed differences in their performance outcomes, particularly in terms of accuracy and computational demands.

Furthermore, in both subtasks 1 and 2, a thorough examination of data preprocessing steps, such as image resizing and padding size, along with the exploration of various hyperparameters, will be conducted. This investigation aims to evaluate the sensitivity of the model to these modifications. The impact of these alterations will be assessed by comparing the accuracy metrics and observing the resultant changes in model performance. This approach is intended to provide a comprehensive

understanding of how specific preprocessing and hyperparameter tuning strategies influence the overall efficacy of the model.

3.2 Literature Review

The introduction of Convolutional Neural Networks (CNNs) and Support Vector Machines (SVMs) in image classification and OCR rendered a significant evolution in the field of machine learning. This literature review synthesizes the findings and implications of three different studies, each contributing uniquely to our understanding of these technologies.

"An Architecture Combining Convolutional Neural Network (CNN) and Support Vector Machine (SVM) for Image Classification" by Abien Fred M. Agarap, is a novel approach to image classification proposed through the combination of CNNs and linear SVMs. This hybrid model challenges the conventional use of softmax functions in CNNs by integrating SVM as the final layer classifier. CNN is often known for its strength in feature extraction through convolutional layers and non-linearities such as ReLU. This study demonstrates that CNNs can be enhanced by SVM's classification capabilities. Agarap's research shows a significant accuracy of 99.23 % and 90.72% from MNIST and Fashion-MNIST datasets respectively, accentuating the potential of combining CNNs with SVM in tasks requiring refined classification, such as OCR [3].

Yang Gong and Pan Zhang's literature, "Research on MNIST Handwritten Numbers Recognition based on CNN," delves into the application of CNNs in handwritten digit recognition. Their research utilizes the MNIST dataset to train a CNN model comprising various convolutional and pooling layers. The model achieved impressive accuracy rates, 100% on the training set and 99.25% on the test set, demonstrating the effectiveness of CNNs in OCR tasks. This study highlights the importance of feature extraction and data dimensionality reduction in recognizing individual characters from complex backgrounds [4].

The study "Comparison Between Neural Network and Support Vector Machine in Optical Character Recognition" by M. R. Phangtriastu et al. examines various feature extraction techniques in Optical Character Recognition (OCR). It focuses on zoning algorithms, projection profile, Histogram of Oriented Gradients (HOG), and their combinations. The literature compares these methods using Support Vector Machine (SVM) and Artificial Neural Network (ANN) classifiers. Notably, the highest accuracy achieved in the experiment is 94.43%, using the SVM classifier with

a combination of the projection and HOG algorithm. This result highlights the effectiveness of SVM in OCR tasks, particularly when paired with specific feature extraction techniques [5].

Collectively, these studies illustrate the evolving landscape of machine learning, where the integration of different methodologies like CNNs and SVMs leads to more robust and accurate image classification and OCR systems. The continuous improvements in these technologies, driven by seminal research, hold great promise for their application in the future. The challenge remains in optimizing these models for generalization and efficiency for real-world application.

3.3 CNN-based OCR model

3.3.1 The dataset

The dataset contained 1778 binary images representing seven different alphanumeric characters (0, 4, 7, 8, A, D, H), each measuring 128x128 pixels. Each corresponding alphanumeric symbol is represented in multiple typefaces, illustrating the diversity of font styles, shown in Fig 22. The binary format—consisting solely of black and white pixels—significantly simplifies the feature extraction process for the CNN model while each image's resolution of 128x128 pixels represents a stable balance between the need for detail against computational efficiency. The dataset also exhibits a uniform distribution, with each of the seven characters being represented by 254 images. Having the same number of images for each class is pivotal in machine learning, as it mitigates the risk of model bias towards classes that are more prevalently represented in the dataset. This uniformity is instrumental in fostering a balanced learning environment for CNN, ensuring equitable attention to all character classes.

The dataset was divided randomly into training and validation subsets, with the training subset comprising 75% of the total images and the remaining 25% allocated for validation. This method of random splitting is employed to reduce potential biases that could emerge from non-random selection and to diminish the likelihood of overfitting. It also enhances the model's accuracy by ensuring more solid and dependable testing metrics and providing a more accurate representative of the overall distribution of data. No preprocessing or data augmentation was performed on the dataset before its use.

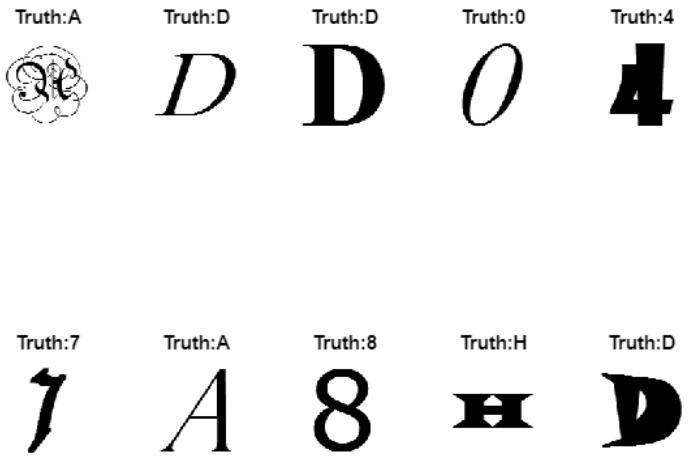


Figure 22, Example images of the Dataset

3.3.2 CNN Model Architecture

The CNN architecture is specifically suited for processing data such as images with a grid-based structure. This is characterized by a series of layers, each performing distinct functions that contribute to the network's ability to interpret and classify visual data effectively. The initial convolutional layers are typically responsible for feature extraction, using filters to detect edges, textures, and other basic visual elements. In this project, the architecture detects the features of the alphanumerical characters shown in Figure 22. This report evaluates various model architectures by testing configurations with differing numbers of layers and exploring the impact of using max pooling, average pooling, or a combination of both. The report also adjusts preprocessing steps and hyperparameters—including learning rate, optimizer selection, data shuffling, maximum epochs, padding, and image resizing—to determine their effects on model performance. This comprehensive approach allows us to identify the most effective strategies for improving the accuracy and efficiency of our models.

3.3.2.1 The Optimal CNN Model Architecture

Figure 23 illustrates the optimal CNN architecture, which encompasses one input layer, seven convolutional layers, a single dropout layer, and three distinct output layers. The detailed composition of this architecture is as follows:

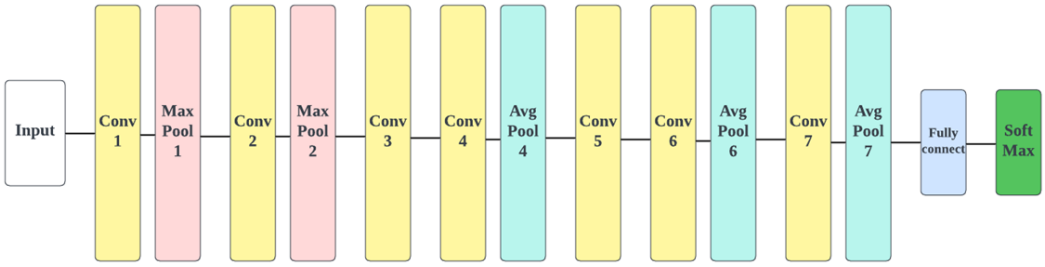


Figure 23, The Optimal CNN architecture

- **Input Layer**: Accepts binary images with dimensions of 128x128 pixels.
- **Convolutional Layers**: The model includes seven convolutional layers, each followed by batch normalization to address the issue of internal covariate shift. The internal covariate shift occurs when the distribution of each layer's input changes as the parameters of the previous layers change. This can slow down the training process as the layers need to continuously adapt to new distribution. Batch normalization prevents this problem by normalizing the output of each convolutional layer, thereby enhancing convergence and performance. Convolutional operations utilize ReLU activation functions. ReLU function can introduce non-linearity into a model to solve complex functions and be computationally efficient because any input less than zero is set to zero and any input greater is left unchanged. Pooling layers were included for some layers and excluded for others.

    1. **First Convolutional Layer**: Utilizes 8 kernels of 3x3 size, stride 1, and padding is kept at [1, 1, 1, 1] for top, bottom, left and right size. This was found through an experiment in the next section. Padding is kept constant until the seventh layer. Overall, it produced an output of 128x128x8. Smaller kernel sizes (3x3) require fewer parameters, reducing computational load. They also allow for deeper architectures by stacking more layers without too quickly reducing the spatial dimensions of the feature maps. A smaller number of kernels (8) is chosen initially because it acts as a form of regularization, reducing the risk of overfitting by not capturing high-frequency variations, irrelevant patterns, or noise. Subsequent max pooling with a 2x2 filter and stride 2 reduces the dimensions to 64x64x8.

2. **Second Convolutional Layer**: Employs 16 kernels of 3x3 size with identical stride and padding as previously, yielding an output of 64x64x16. A max-pooling layer of size 2x2 with a stride of 2 further reduces the feature map to 32x32x16. The stride halves the size of the feature map, thereby reducing the feature extraction needed to be processed by the later layers.

3. **Third Convolutional Layer**: 32 kernels of size 3x3 were used with a stride of 1. This resulted in an output volume of 32x32x32. As the network depth increases, the number of kernels also increases. This is because deeper layers in the network are capable of learning more complex and specialized features, which justifies the use of a larger number of filters. The pooling layer was omitted to preserve spatial information. At this stage, retaining the full resolution of the feature map is critical for capturing details and complex features. By not reducing the dimensions, it enables the construction of deeper networks while maintaining the spatial dimensions of the layers.

4. **Fourth Convolutional Layer**: 64 kernels measuring 3x3 with a stride of 1 were utilized, resulting in an output volume of 32x32x64. Notice that kernel size is maintained constant as it keeps the focus on local features while still capturing enough spatial context to allow for the combination of lower-level features into higher-level structures within the image. An average pooling layer with a 2x2 size and a stride of 2 was employed in place of a max pooling layer. This approach was chosen because deeper layers contain crucial high-level feature information. By using average pooling, the reduction of vital spatial information is minimized while textural information is preserved. The resulting feature map for this layer had dimensions of 16x16x64.

5. **Fifth Convolutional Layer**: Uses 128 kernel size 3x3 with identical stride and padding as previously, resulting in a feature map of 16x16x128. The pooling layer was omitted for the reasons stated in the third convolutional layer.

6. **Sixth Convolutional Layer**: Also uses 128 kernels size of 3x3 with a stride of 1, followed by an average pooling layer of size 2x2 with a stride of 2, leading to an

8x8x128 output. The average pooling layer was employed for the same reason stated earlier.

7. **Seventh Convolutional Layer**: Features 256 kernels of 3x3 size with stride 1 and 0 padding was employed for this convolution operation resulting in the output volume size of 6x6x256. After applying average pooling, the output volume was reduced to 3x3x256. Zero padding means that the convolutional filter only passes over the valid region of the input image or feature map where the filter fully fits. This leads to a concentrated feature map that represents higher-level features. This is required for the last layer because the filters typically capture complex and abstract representations of the data rather than simple, low-level features like edges and textures.

- **Dropout Layer**: Incorporated to regularize the network and diminish the risk of overfitting. Overfitting is a scenario where the network learns the overlearns the training data including noises and outliers, thus performing poorly when presented with unseen data. The dropout layer ignores a subset of neurons along with their connections which renders the network less sensitive to the specific weight of the neurons. In effect, it forces the network to learn more robust features that are more useful with many different random subsets of the other neurons.

- **Output Layers**: Made up of 3 layers; a fully connected layer with seven neurons, a SoftMax activation layer for probability distribution across classes, and a final classification layer. The fully connected layer combines all the high-level features from previous layers are combined. Each of the seven neurons corresponds to a different class that the network is attempting to classify. The output of this layer is raw predictions that are fed into the softmax functions which are then converted into normalized probabilities, ensuring that the output of the values is between 0 and 1, sum up to 1. The output can be interpreted as the network's confidence in the input in each of the possible classes (0, 4, 7, 8, A, D, H). The classification layer computes the difference between the predicted probabilities and the true labels, producing a signal (loss) that measures the performance of the network. During the training, this loss is used to perform backpropagation, where

the network adjusts its weights to minimize the loss, thereby improving its accuracy over time.

3.3.2.2 Training Hyperparameters/parameters of the Optimal CNN Model

Table 1 displays the optimal training hyperparameters/parameters of the optimal CNN model which consistently maintained over 99 percent accuracy on the validation set during the training process.

The optimal configuration of hyperparameters was determined through a series of experiments, the specifics of which are comprehensively detailed in Section 3.3.3.

Table 1, Hyperparameter/parameter of the model

Parameters/Hyperparameters	**Value**
Optimizer	'adam'
InitialLearnRate	0.001
LearnRateSchedule	'piecewise'
LearnRateDropPeriod	3
LearnRateDropFactor	0.5
MaxEpoch	10
shuffle	'every-epoch'
ValidationData	Testset
ValidationFrequency	10
MiniBatchsize	64
Verbose	false
ExecutionEnvironment	'gpu'

The hyperparameter/parameters shown have the following roles in the training process of the neural network. Details of how we came to the optimal value for each parameter are outlined in 3.3.3:

1. **Optimizer**: This is the algorithm used to update the weights of the network based on the gradient of the loss function. Following the experiments utilizing both Stochastic Gradient

Descent with Momentum (SGDM) and Adam optimizers, it was determined that the Adam optimizer enhanced the accuracy of our CNN model more effectively. Adam also had a faster learning rate than SGDM.

2. **InitialLearnRate**: This sets the starting learning rate, which controls the size of the steps taken during the optimization process We tested the learning rate at various magnitudes: 0.1, 0.01, 0.001, and 0.0001, to find the optimal setting for our model. The rate of 0.001 emerged as the most effective, striking a balance between accuracy and training time. While a learning rate of 0.0001 ensured a gradual and precise approach towards convergence, it significantly extended the training duration, making it less practical for efficient model development.

3. **LearnRateSchedule**: defines how the learning rate changes over time during training. It can be kept constant or can be adjusted based on certain criteria or after a certain number of epochs, which can be employed to fine-tune the convergence towards the end of training. Setting it at "piecewise" gave the most optimal results.

4. **LearnRateDropPeriod**: specifies the number of epochs between each decrease if the learning rate schedule involves decreasing the learning rate over time. We found that the result was most optimal when it was set to 3.

5. **LearnRateDropFactor**: details the factor by which the learning rate should be decreased at each drop period. A drop factor of 0.5, which halves the learning rate at each drop was set to give the best result.

6. **MaxEpoch**: specifies the maximum number of epochs for which training will continue before stopping. Max Epoch was tested in the next section with 5,10,15,20,40. The best result was obtained at 10. Therefore, this was set for the optimal model.

7. **Shuffle**: determines whether to shuffle the training data before each epoch. Shuffling helps prevent the network from learning the order and forming biases in the training data. Since our experiments focused on a limited set of characters, omitting the shuffle did not significantly affect performance. The limited scope and size of the dataset likely reduced the risk of the model picking up on sequence-specific patterns, which shuffling typically

mitigates. Should the dataset be expanded, or its complexity increased, it's conceivable that shuffling could play a more critical role in enhancing the model's effectiveness.

8. **ValidationData**: Dataset separate from the training set used to evaluate the model's performance during training.
9. **ValidationFrequency**: sets how often the model is evaluated on the validation data. Set to 10 iterations as it gave the best accuracy.
10. **MiniBatchSize**: determines the number of samples in a batch from the training dataset that are propagated through the network before the optimizer updates the weights. Overall similar results for batch sizes 16,32,64. However, 64 is chosen because it renders a faster convergence and reduces the computational cost.
11. **Verbose**: controls the amount of information to be printed out during training. A higher verbosity level can include progress bars, loss metrics, and other detailed information. This was set to a default value.
12. **ExecutionEnvironment**: This specifies where the training should take place, this was set to GPU.

3.3.3. Optimization of the CNN model

This section explores the optimization of a Convolutional Neural Network (CNN) for enhanced accuracy and efficiency. It aims to investigate the effects of varying the architecture's structure, including adjustments to pooling layers, the number and type of convolutional layers, and the selection of activation functions. Additionally, it evaluates the impact of different preprocessing techniques, such as image resizing, padding, stride adjustments, and kernel sizes, on model performance. Fine-tuning hyperparameters, including the choice of optimization algorithm, the initial learning rate, learning, epoch number, and many more are also explored. This detailed investigation aims to pinpoint the most efficacious combinations of hyperparameters and architectural choices that enhance model accuracy and generalization capabilities across various datasets. The outcomes of this exploration are systematically presented in Table 1.

3.3.3.1 Effect of Resizing Images

This section examines the effects of resizing images to the CNN model. For this analysis, all other training hyperparameters are held constant, as detailed in Table 1. Some parameters of model architectures in section 3.3.2 were modified to accommodate the change in image size. The stride of the sixth layer was reduced to 1 to ensure that the feature map size remains larger. The padding was changed to 1 in the seventh layer to maintain the spatial dimension. Figure 24 presents the validation accuracies of confusion matrices where the images were resized to different dimensions. Max Epoch was kept to 10. Specifically, the images were resized to [32x32], [64x64], [256x256], and [512x512] pixels, respectively.

Table 2 represents the average validation accuracy of each resize from ten different training sessions. We can observe that, with the smallest image size of 32x32 pixels, the model is compelled to focus on the most essential features of the alphanumeric characters, which can lead to robust but slightly less accurate recognition due to the loss of finer details. As the image size increases to 64x64, it benefits from additional spatial resolution, which helps in capturing more nuanced characteristics of the various fonts and characters, leading to improved accuracy. However, beyond the default size, when the images are upscaled to 256x256 and 512x512 pixels, the CNN might start to overfit, learning intricate patterns and noise that do not generalize well, leading to a slight decline in accuracy. This effect is exacerbated as the total dataset is relatively small such as ours of 1778 images. Overfitting can be prevented if the increase in resolution is matched by an increase in the size of the dataset.

The increase in elapsed time with larger image sizes is a direct consequence of the computational complexity involved in processing higher-resolution images. As the resolution of the input images increases, the number of operations required for each convolution scales, accordingly, significantly increasing the computational load. This results in longer training times for each epoch and, cumulatively, a substantial increase in total training duration. This is markedly noticeable when comparing the training times for 256x256 images against 512x512 images; the fourfold increase in pixel count leads to a considerably longer training time, as evidenced by the jump in elapsed time from 78 seconds to 270 seconds.

Table 2, Accuracy of various resizing images

Size of the image	Accuracy (%)	Elapsed Time (s)
32x32	97.53	37
64x64	99.2	43
256x256	98.1	72
512x512	97.2	270

3.3.3.2 Effect of Padding

In this section of the report, we delve into the impact of varying padding sizes on the performance of a CNN model. Padding is an essential factor in CNN architecture as it influences the spatial dimensionality of the output feature maps and, by extension, the network's ability to preserve spatial information critical for accurate predictions. We systematically evaluate four distinct padding strategies: 'same' padding, which maintains the original spatial dimensions of the image as per MATLAB's default settings, and three additional custom padding configurations—minimal, moderate, and large—specified respectively as [1,1,1,1], [2,2,2,2], and [4,4,4,4] for the top, bottom, left, and right dimensions of the input images. We kept the maximum number of epochs at 10. All other parameters were consistent with the settings outlined in section 3.3.2. In Table 3, we present the average accuracy derived from ten separate training iterations for each padding scenario.

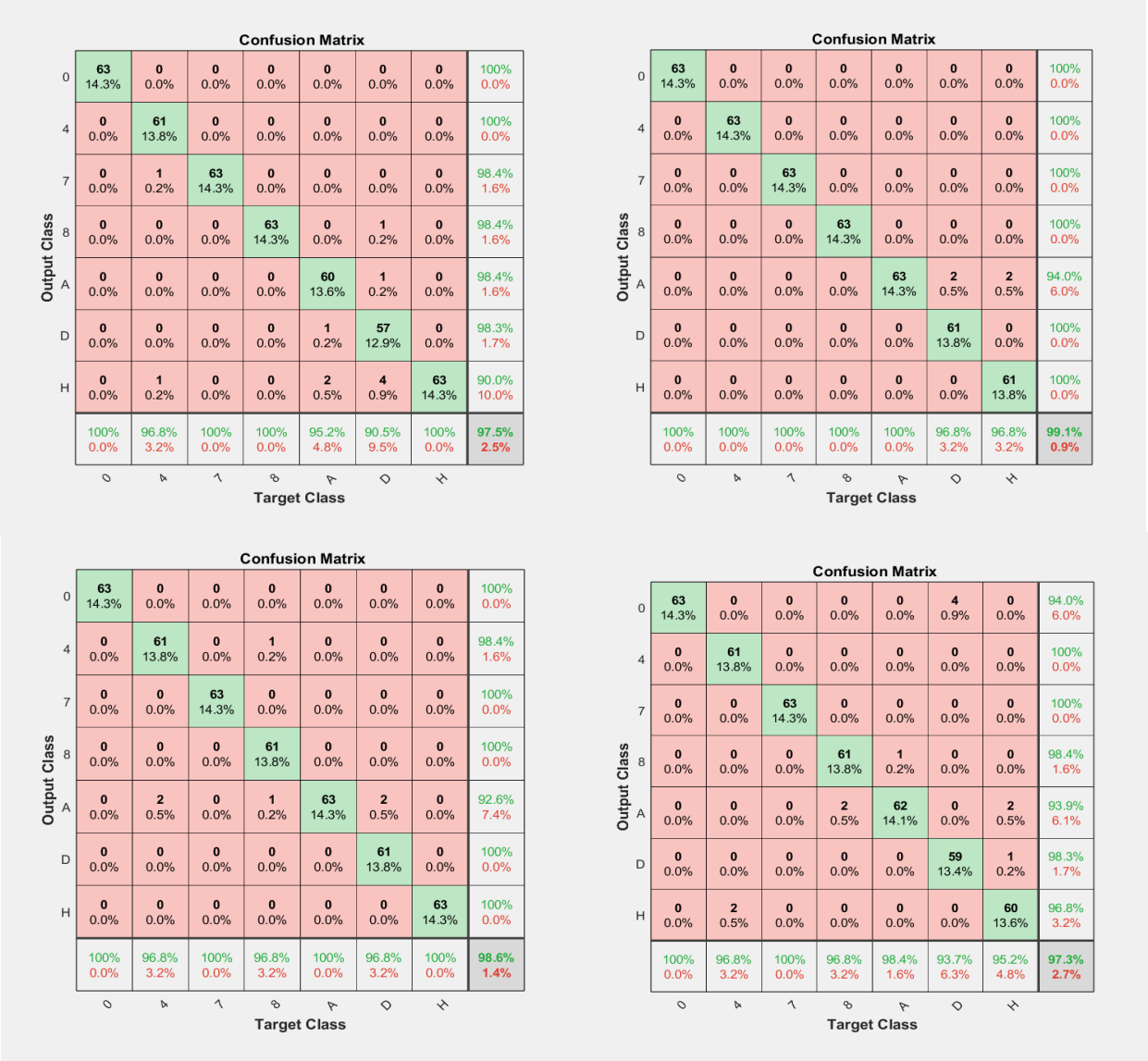


Figure 24, Confusion Matrix of 32x32 (top left), 64x64 (top right), 256x256 (bottom left), 512x512 (bottom right)

Table 3, Effect of Padding Size on Accuracy

Padding	Accuracy (%)
"same"	99.1
Minimal	99.81
Moderate	99.0
Large	98.1

The observed trend in accuracy relative to padding size, as detailed in Table 3, can be attributed to the balance between maintaining spatial information and introducing unnecessary context to the convolutional neural network (CNN). With the 'same' padding, the spatial dimensionality of the input is preserved, allowing the network to leverage the entire feature space of the input data. However, this can sometimes include the retention of irrelevant spatial information, which may not be beneficial for the model.

When minimal padding is applied, there is a slight increase in accuracy, suggesting that a small amount of padding can help CNN learn the important features near the edges of the input images without overwhelming the model with excessive information. The minimal padding likely provides just enough additional context to help CNN recognize characters that are closer to the border of the image, which might otherwise be distorted or partially lost with 'valid' padding.

As the padding size increases to moderate and then large, the accuracy starts to decrease. This can be explained by the introduction of too much irrelevant information, which could potentially dilute the important features that the network needs to learn. Large padding may cause the network to focus on the padded pixels rather than on the actual content of the image, leading to less effective learning and lower accuracy. Additionally, excessive padding might also increase the likelihood of overfitting, as the network becomes too tuned to the training data's specific padded patterns, which do not generalize well to unseen data.

3.3.3.3 Effect of Epoch

In this section, we investigate the effect of varying the 'MaxEpochs' setting within the architecture of a CNN tasked with character recognition. The experiment involves adjusting the maximum epochs to 5, 10, 15, 20, and 40, to identify the point at which the model reaches its optimal performance. The accuracy results from this experiment, averaging over ten training sessions, are presented in Table 4. The minimal padding from section 3.3.3.2 was applied to the padding setting throughout as it gave the highest accuracy.

Table 4, Average accuracy for different MaxEpochs

MaxEpochs	Accuracy (%)
5	98.6
10	99.9
15	98.9
20	98.5
40	99.5

Analyzing the accuracy data, we observe a noteworthy trend: the accuracy increases significantly as the number of epochs grows from 5 to 10, suggesting that the model benefits from additional iterations over the training data, possibly moving towards convergence. However, post-10 epochs, there is a dip in accuracy when the training is extended to 15 and 20 epochs. This decline could be attributed to the model beginning to overfit the training data, where it learns patterns and noise specific to the training set that do not generalize well to unseen data. Interestingly, the accuracy climbs once more as we increase the epochs to 40. This could indicate a complex dynamic where the model, after initially overfitting, begins to refine its weights and biases to better generalize, possibly due to additional regularization effects that emerge over more extended training or because the model finds a better local minimum within the loss landscape after initially settling for a suboptimal one. This secondary rise in accuracy underscores the nonlinear and sometimes unpredictable relationship between epoch count and model performance in machine learning.

Consequently, while a resurgence in accuracy is noted at 40 epochs, the optimal balance between computational efficiency and model performance is achieved at 10 epochs, as higher epoch counts demand disproportionately greater computational resources without a commensurate improvement in accuracy.

3.3.3.4 Effect of Initial Learn Rate

The learning rate in CNN is a hyperparameter of paramount importance as it dictates the size of the steps the model takes during the optimization process. A learning rate that is too high can cause the model to converge too quickly to a suboptimal solution, while a rate that is too low may result in a prolonged training process that may never converge. In this section, we analyze the impact of

varying the initial learning rate on the performance of a CNN designed for character classification. We adjust the 'InitialLearnRate' in the training options to 0.1, 0.01, 0.001, and 0.0001, and observe the resulting accuracy, as shown in Table 5. This experiment was conducted with the optimal settings from previous sections, using minimal padding and setting the maximum number of epochs to 10, while all other parameters remained as defined in section 3.2.2.

Table 5, Various Initial Learn Rate and their accuracies

InitialLearnRate	**Accuracy (%)**
0.1	65.4
0.01	98.7
0.001	99.9
0.0001	97.3

The accuracy trends depicted in Table 5 reveal a clear peak at an initial learning rate of 0.001. With a learning rate of 0.1, the model's accuracy is significantly lower, indicating that large step sizes may be causing the optimization process to overshoot the minimum in the loss landscape. This can be shown in Figure 25 where accuracy was as low as 14.5% for an initial learn rate of 0.1. As the learning rate is reduced to 0.01, we observe a substantial increase in accuracy, suggesting that smaller steps in the weight updates allow the model to fine-tune its parameters more effectively without bypassing optimal points. The peak accuracy at a learning rate of 0.001 implies that this rate is most conducive for the model to carefully navigate the loss landscape and converge to the best solution. However, further reducing the learning rate to 0.0001 results in a slight decline in accuracy. This could be due to the model's learning process becoming too conservative, potentially getting stuck in local minima or not allowing the model to fully converge within the given number of epochs. This demonstrates the delicate balance required in setting the learning rate to ensure adequate convergence without compromising the training efficiency or model accuracy.

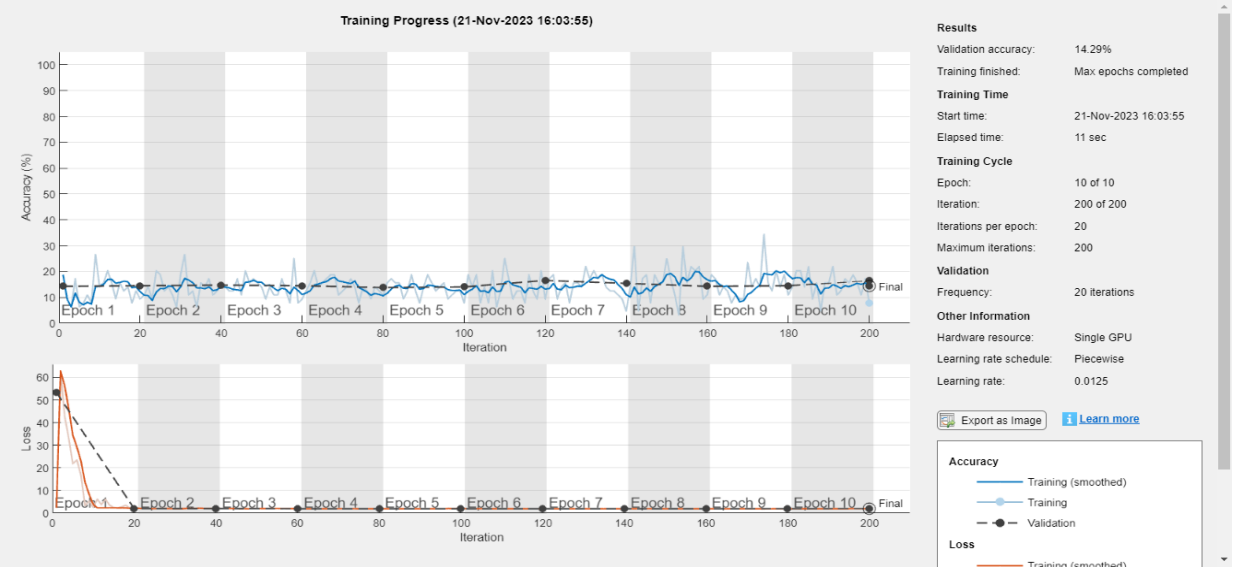


Figure 25, Training result of CNN model with initial learn rate of 0.1.

3.3.3.5 Effect of optimizers

In this section, we evaluate the performance of two widely used optimizers: Adaptive Moment Estimation (Adam) and Stochastic Gradient Descent with Momentum (SGDM). The choice of an optimizer in training convolutional neural networks (CNNs) is a critical decision that influences how the model weights are updated during training. It affects the speed and quality of the convergence to a minimum in the loss function. These optimizers have distinct mechanisms for navigating the loss landscape, and understanding their effect on the accuracy of a CNN model is essential. Our experiments are based on the optimal parameters identified in previous sections, which include minimal padding, a max epoch count of 10, and an initial learning rate of 0.001. The comparative analysis of the optimizers averaged over 10 training sessions, is presented in Table 6, with the rest of the model parameters held constant as specified in section 3.2.2.

Table 6, Accuracy of adam and sgdm

Optimizer	**Adam**	**SGDM**
Accuracy (%)	99.9	97.4

Table 6 reveals a marked difference in accuracy between the two optimizers, with Adam achieving 99.9% accuracy compared to SGDM's 97.4%. The superior performance of Adam can be attributed to its adaptive learning rates for different parameters and its use of moment estimates, which enable it to navigate the loss landscape more effectively. Adam combines the benefits of two other extensions of stochastic gradient descent: Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp), allowing it to handle sparse gradients on noisy problems.

On the other hand, SGDM, while often effective, can struggle with the same challenges due to its reliance on a constant learning rate and momentum to accelerate the gradient vectors in the right direction. This can lead to suboptimal convergence behavior, especially in complex loss landscapes or when dealing with highly non-linear functions as is typical in deep learning for character recognition. The fall-off in accuracy for SGDM in this context suggests that Adam's sophisticated mechanisms for adjusting learning rates and considering the momentum of past gradients are particularly beneficial for the nuances of our CNN model's training process.

3.3.4 The Result of the Optimal Model

After designing the most optimal CNN model, a validation accuracy of 99.9% was obtained. This aligns closely with the top-tier results of many existing models in character and digit recognition, where accuracies are generally around 99.8%. Although there is potential for further enhancement, the current efficacy of our model is notable, reflecting its robust capability in character and digit identification through its high accuracy. Figure 26 displays the training process of the optimal CNN model. The detailed performance of the model can be visualized through the confusion matrix and chart depicted in Figures 27 and 28, respectively. Moreover, Figure 29 displays a selection of accurate character recognition instances from the validation set. Finally, when applying our CNN to segmented characters extracted from the original image, it impressively achieved a perfect accuracy rate of 100%, as exhibited in Figure 30.

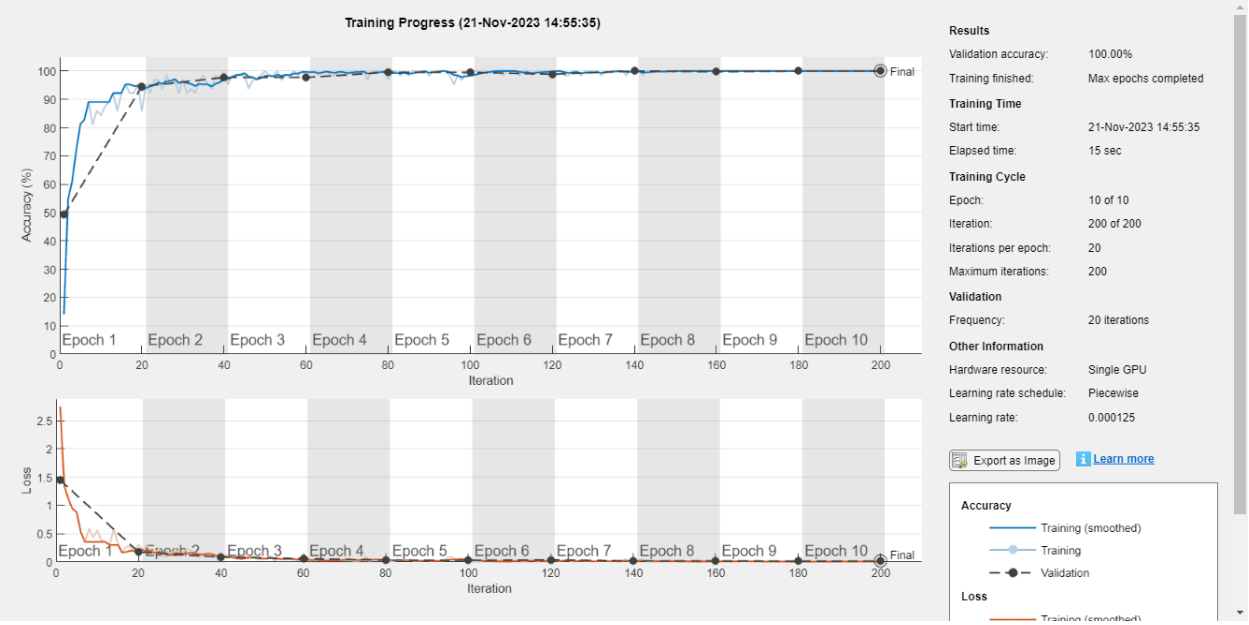


Figure 26, Training process of the optimal CNN

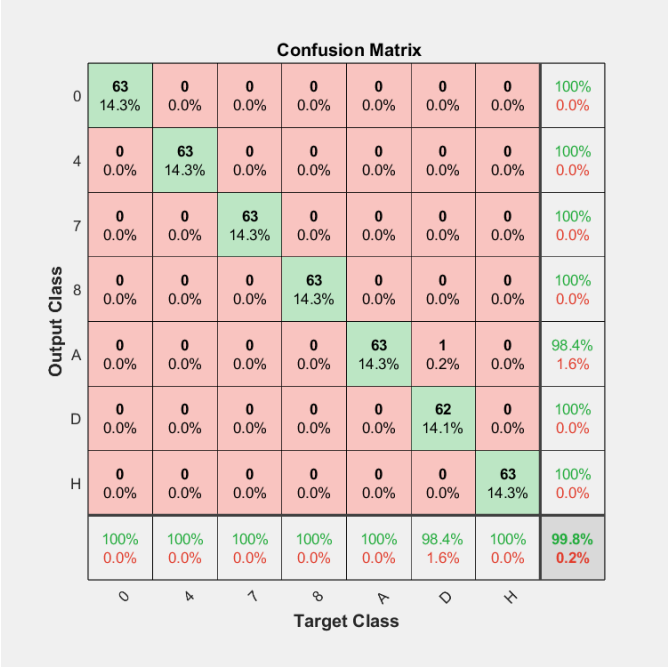


Figure 27, Confusion Matrix of the optimal CNN

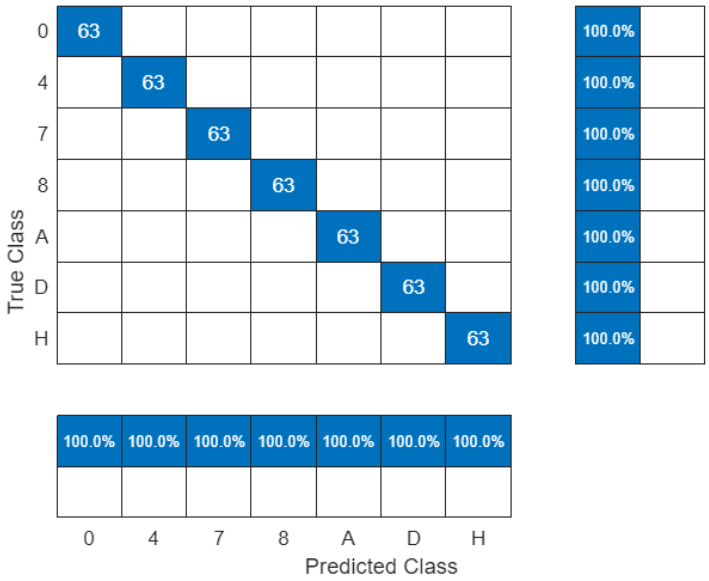


Figure 28, Confusion Chart of the optimal CNN

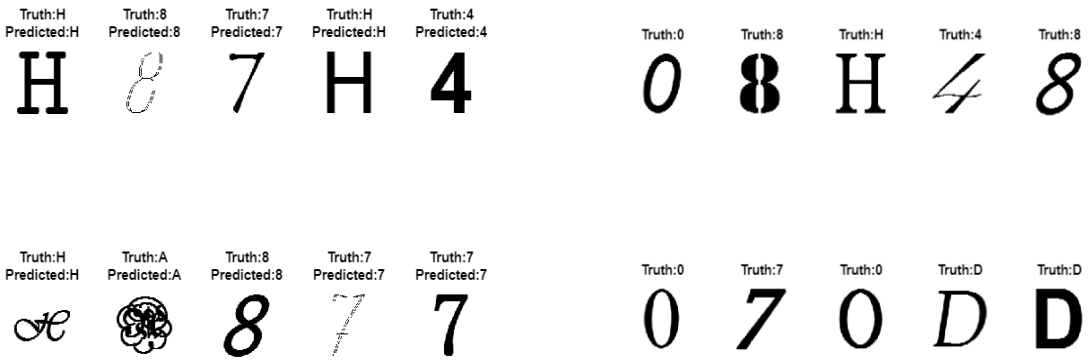


Figure 29, Validation set; predicted vs truth label

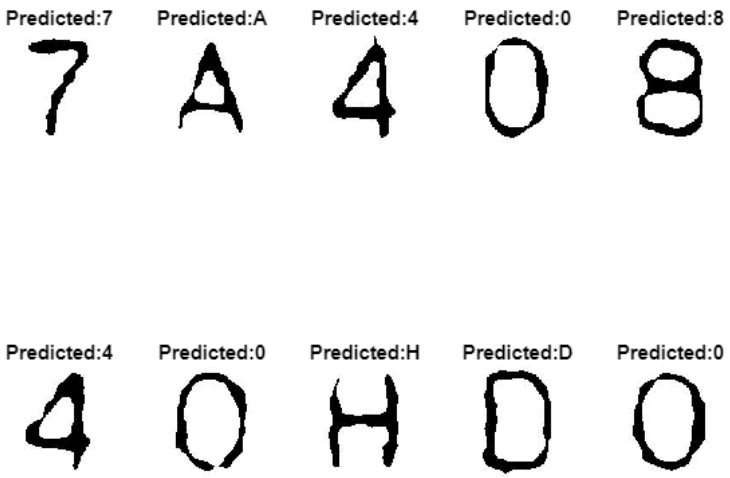


Figure 30, Segmented Labels predicted.

3.4 SVM-based OCR model

3.4.1 Problem introduction

The objective of this section is to construct a character recognition system in MATLAB without using convolutional neural networks (CNNs). The system is intended to classify individual characters extracted from a BMP image of a microchip label. The chosen method for classification is Support Vector Machine (SVM) with binary encoding. This approach is particularly advantageous for our dataset, which consists of binary images. Binary encoders streamline the SVM's task by simplifying the data structure, thus facilitating the determination of the most effective separating hyperplane. SVMs are renowned for their efficacy in creating robust multi-class classification systems. This same dataset, encompassing 1778 binary images of seven distinct alphanumeric characters (0, 4, 7, 8, A, D, H), and standardized to a 128x128 pixel scale, was previously utilized to train a CNN-based classifier.

The decision to utilize SVM over other models stems from several of its inherent strengths. Primarily, SVMs are adept at handling high-dimensional data, as is the case with the 128x128 binary images, where each pixel represents a dimension. Furthermore, SVMs are less prone to overfitting, especially in cases where the number of dimensions exceeds the number of samples, which is beneficial given the relatively small size of the dataset. The binary encoding simplifies the input for the SVM, effectively transforming the complex patterns of the characters into a format that is more manageable for the SVM to analyze and classify. This binary approach aligns well with the SVM's binary nature, which inherently categorizes data into two classes. The extension to multi-class problems, as required for character recognition, is then achieved by combining several binary SVM classifiers, a technique for which SVMs are particularly well-suited. The selection of SVM for this task is also supported by its successful track record in similar classification tasks, indicating its capability to generalize well from training data to unseen data, a critical requirement for robust character recognition.

3.4.2 SVM Model Architecture

The architecture of the SVM is divided into feature extraction and classifier training. The initial segment is dedicated to identifying and extracting pertinent features from the input data. The subsequent segment utilizes these extracted features as a foundation for training the classifier.

3.4.2.1 Feature Extraction

For the task of feature extraction within the SVM model, the MATLAB computer vision toolbox was employed to assess the viability of several algorithms. After a thorough comparison among Sobel, PCA, Histogram of Oriented Gradients (HOG), and Local Binary Pattern (LBP), the HOG algorithm emerged as the superior choice for our specific requirements. This was tested by implementing different extraction methods. As shown in Table 7, HOG significantly outperforms the other considered algorithms with an accuracy of 99.1%, while Sobol, PCA, and LBP lag with accuracies of 87.8%, 58.5%, and 36.1%, respectively. Figure 31 displays the confusion matrix of Sobol, CA, and LBP to visualize the accuracy and losses.

This stark contrast in performance can be attributed to the specific characteristics of the HOG algorithm and the nature of your dataset. HOG's method of segmenting the image into cells and computing gradients captures the form and structure of alphanumeric characters very effectively. Since the dataset comprises binary images, where the characters are delineated by clear-cut edges against a uniform background, HOG can exploit this clarity to accurately outline and describe the characters. The gradient-based descriptors are less sensitive to variations in illumination and can discern the shapes of characters which are essential for identification.

Table 7, Accuracy of different extraction methods

Extraction Algorithms	**HOG**	**Sobol**	**PCA**	**LBP**
Accuracy (%)	99.1	87.8	58.5	36.1

In contrast, PCA focuses on reducing dimensionality based on variance, which may not capture the essential edge features needed for character recognition in binary images. LBP, while effective for texture classification, may not be suitable for high-contrast binary images where edge information is more significant than texture patterns. Sobel performs better than PCA and LBP but

still falls short compared to HOG, likely due to its simpler edge detection that doesn't encapsulate the character's shape as effectively.

The HOG's robustness against geometric and photometric changes, except orientation, is crucial for identifying the alphanumeric shapes that remain consistent across various transformations. This robustness, coupled with the one-vs-all classification scheme of SVMs, establishes a potent combination for high accuracy in character recognition. The one-vs-all scheme aligns perfectly with the binary nature of the images, allowing the SVM to determine distinct decision boundaries for each character.

We employed a [20 20] cell size for the HOG feature extraction, despite the general preference for finer grids, which was influenced by the dimensions of our image data. Given the substantial resolution of our 128x128 images, a larger cell size offered a pragmatic compromise, enhancing computational efficiency without unduly sacrificing detail which is crucial for accurate character recognition.

3.4.2.2 Classifier Training

MATLAB's Machine Learning Toolbox was utilized for crafting a multi-class classifier and acquiring the associated class labels. The dataset was partitioned into training and validation subsets, with 75% dedicated to training and the remaining 25% reserved for validation. The 'fitcecoc' function was employed, which is tailored to train support vector machines (SVM) for multi-class problems using the error-correcting output codes (ECOC) technique. The Error-Correcting Output Codes (ECOC) technique is a method for multi-class classification that constructs a binary learner’s ensemble, where each learner is trained on a different dichotomy of the classes, providing a robust error tolerance and improved classification accuracy. This allowed effective training of the model and the assignment of labels to each distinct class.

3.4.3 Result of SVM model

The SVM model achieved an impressive validation set accuracy of 99.1%, which is nearly on par with the 99.9% accuracy secured by the CNN model. Figure 32 illustrates the labels predicted by the SVM model, providing a visual representation of its classification capabilities. To evaluate the performance, confusion matrices and charts were used, as depicted in Figures 13 and 14,

respectively. The SVM model was then tested with the segmented dataset where it displayed an accuracy of 100 percent. Figure 35 displays the SVM model classifying the segmented images correctly.

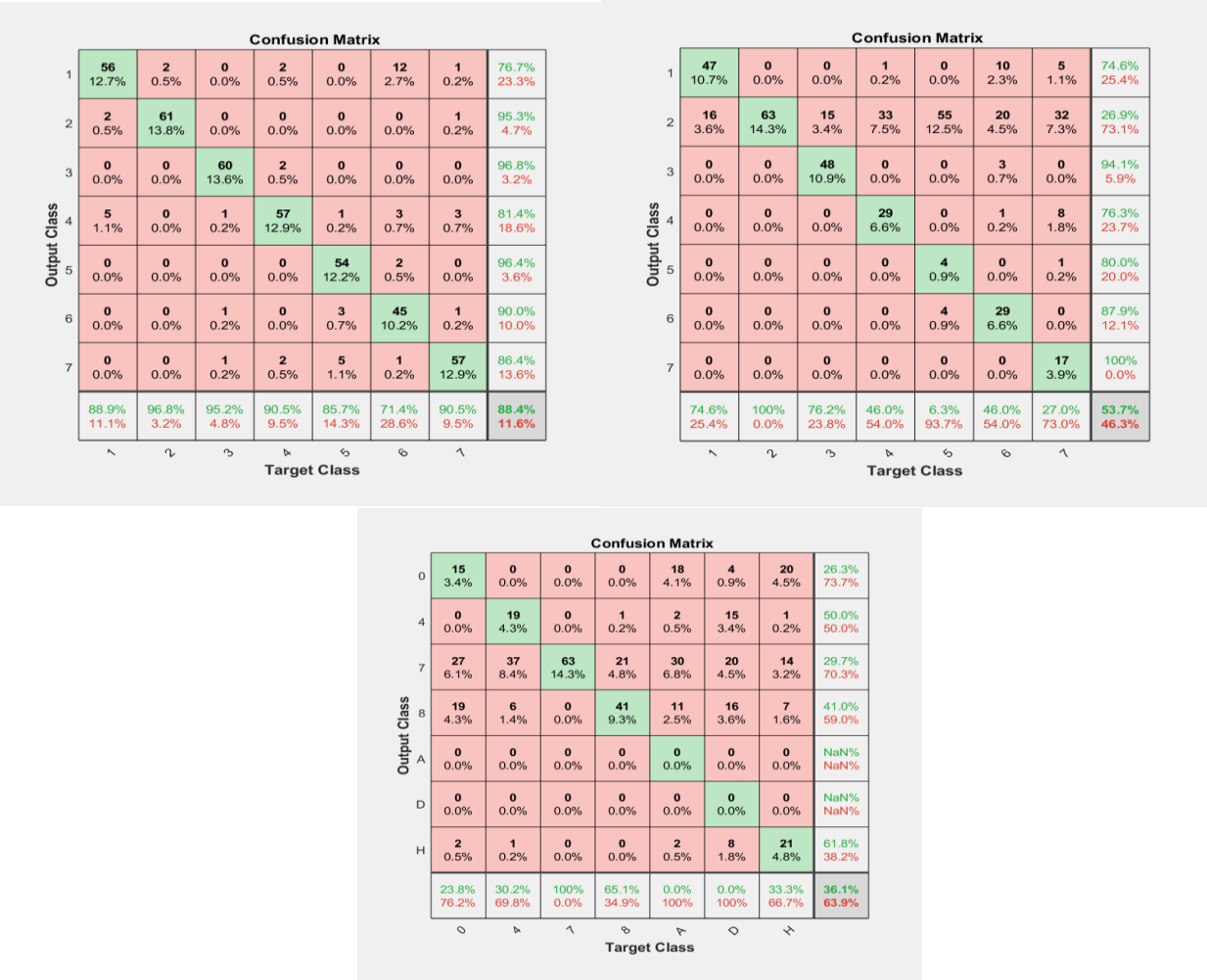


Figure 31, Confusion matrix of Sobel (top left), PCA (top right), and LBP (bottom)

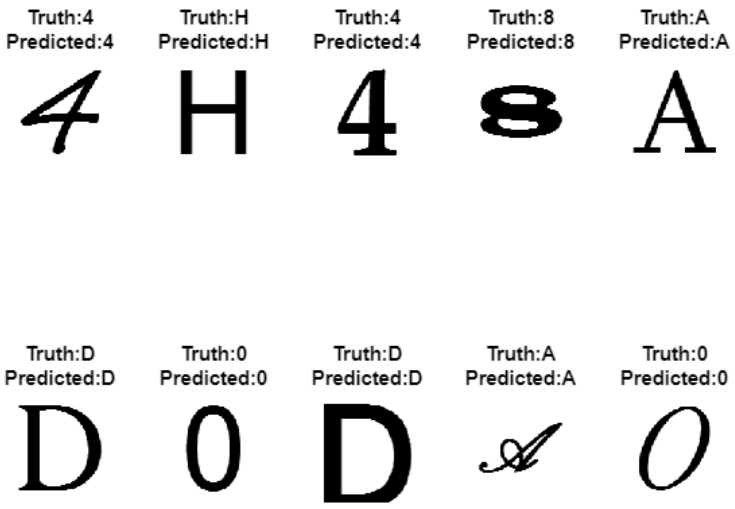


Figure 32, Truth vs predicted label of the SVM model.

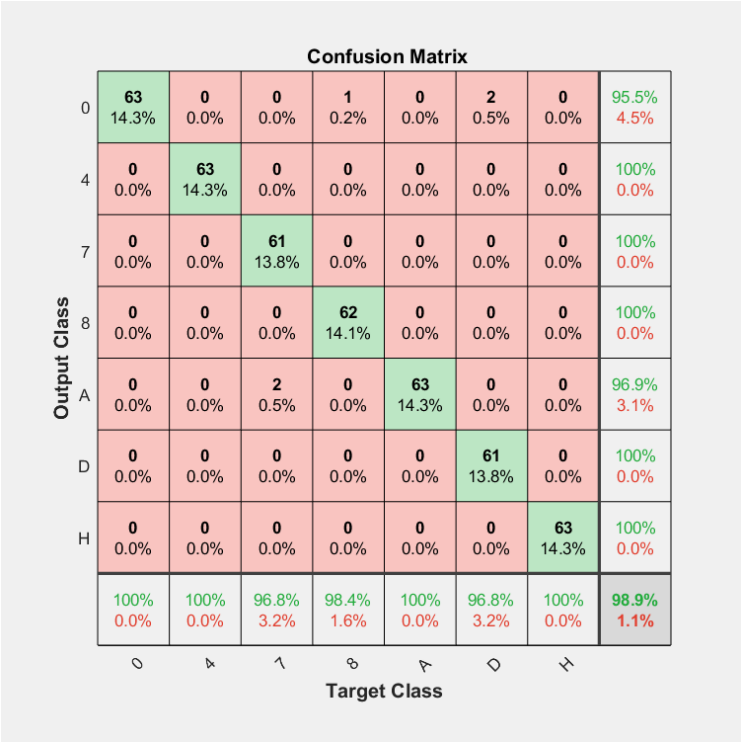


Figure 33, Confusion matrix of the SVM model

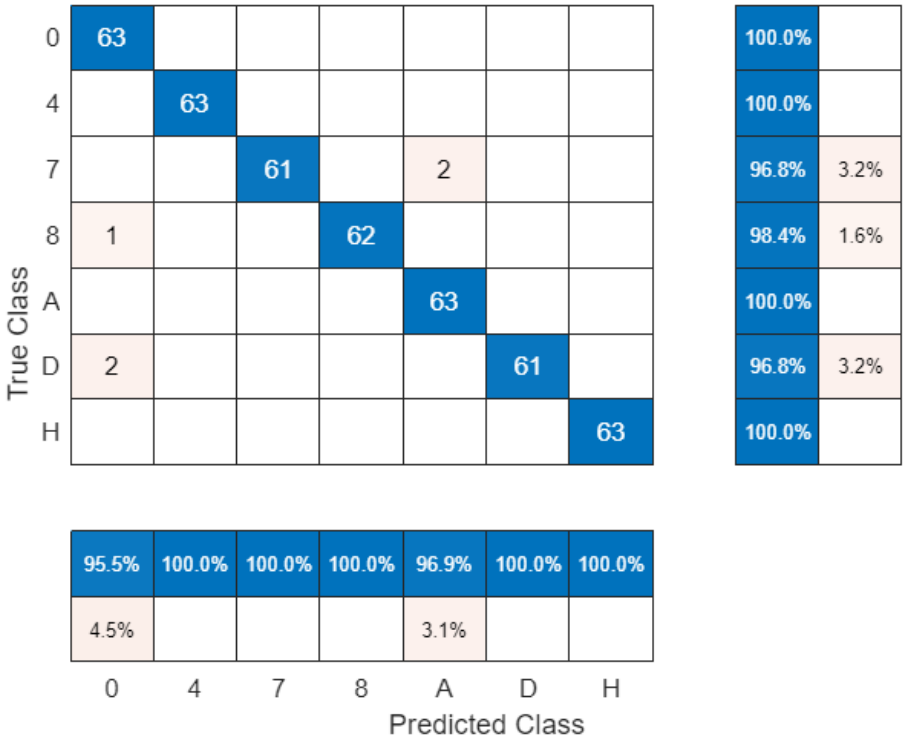


Figure 34, Confusion Chart of the SVM model

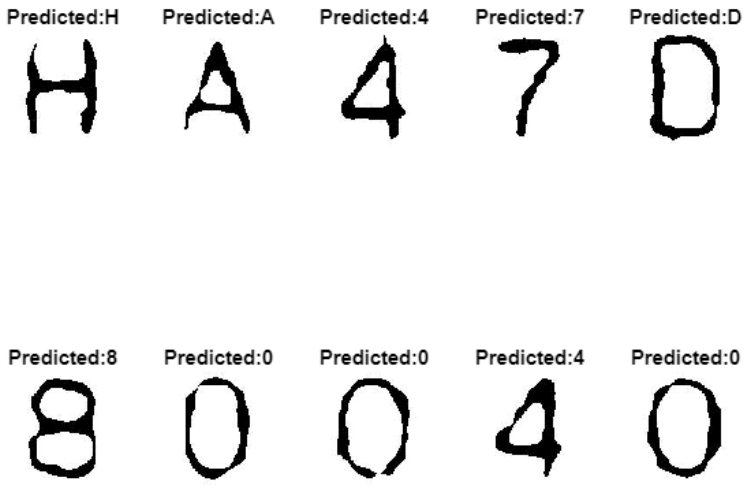


Figure 35, Predicted Label of the SVM model

3.5 Comparison of CNN and SVM model

Comparing the CNN and SVM models based on the accuracy figures provided in Table 8, we can discuss their relative performance and characteristics.

Table 8, Accuracy of CNN and SVM

Classifier	**CNN**	**SVM**
Accuracy (%)	99.9	99.1

Although very minor, it seems that CNN edges SVM in character recognition. This could be due to several reasons. To start off, Convolutional Neural Networks (CNNs) are highly proficient in automatic feature extraction, a key aspect when dealing with image data, such as high-resolution (128x128) binary images. They excel in capturing hierarchical patterns and intricate details through their convolutional layers. This ability to discern and analyze various levels of visual detail allows CNNs to be particularly effective in applications where fine-grained image recognition is essential. The CNN architecture is designed to progressively extract and intensify relevant features, making them ideal for tasks that require a deep understanding of complex image structures, such as identifying subtle variations in patterns or textures within an image dataset.

Additionally, CNNs can learn spatial hierarchies of features, which is extremely beneficial for recognizing alphanumeric characters where the spatial arrangement of pixels is pivotal. The layered learning approach enables CNNs to differentiate and categorize alphanumeric characters based on their unique pixel arrangements, enhancing accuracy in character recognition tasks. By understanding the spatial relationships between pixels, CNNs can effectively handle variations in font, style, and even slight distortions in characters, ensuring robust recognition across diverse image scenarios.

Furthermore, the use of non-linear activation functions enables CNNs to capture the non-linear complexities present in image data. The incorporation of non-linear activation functions such as ReLU equips CNNs to model the intricate and non-linear patterns often encountered in image data, thus enhancing their predictive accuracy in complex visual tasks. This nonlinearity is essential for

adapting to the varying dynamics of image content, allowing the network to differentiate between a wide range of visual cues and patterns.

Lastly, CNN's ability to recognize localized features and remain invariant to the position of these features in an image is crucial for accurately recognizing characters that may vary slightly in position or scale. The inherent design of CNNs to recognize localized features, irrespective of their position in the image, provides a significant advantage in dealing with images where the subject's placement and scale might vary. This spatial invariance is particularly useful in applications like character recognition, where slight shifts or scaling of characters within images are common, ensuring consistent recognition performance regardless of these variations.

3.5.1 Limitation of CNN and SVM

Despite both models achieving impressive accuracies, there can still be limitations. Convolutional Neural Networks, while powerful, are not without their drawbacks. A significant limitation is their tendency to overfit, especially when the training dataset lacks diversity or is too small. Overfitting occurs when a model learns the training data too well, including its noise and outliers, leading to poor performance on unseen data. This issue is exacerbated in CNNs without proper regularization techniques. Another challenge is the requirement for substantial computational resources and time. As CNN architectures become deeper and more complex, the computational cost and the time required for training increase significantly. This makes them less feasible for applications with limited computational capacity or those requiring quick deployment.

In the context of your SVM model, it's important to address the limitations of the Histogram of Oriented Gradients (HOG) feature extraction method. HOG, while effective in capturing the structure and appearance of objects in an image, struggles with variations in illumination, pose, and occlusions. Its performance significantly drops in scenarios where the subject's appearance deviates from the norm, such as changes in lighting or angle. Additionally, HOG features are not rotation invariant, meaning they can fail to recognize objects in different orientations. This limitation requires either augmentation of the dataset with rotated images or the development of more sophisticated feature extraction techniques that can handle such variations. The computational cost of extracting HOG features can also be a concern, especially for high-resolution images or large-scale datasets.

3.5.6 Discussion

Recent advancements in the field of image classification have seen the emergence of innovative approaches that combine the strengths of different models. A notable example from the literature is the hybrid model proposed by Agarap, which integrates Convolutional Neural Networks (CNNs) with Support Vector Machines (SVMs), covered earlier in Section 3.2. This model uses a linear SVM as the final layer classifier instead of the conventional softmax function, effectively combining the powerful feature extraction capabilities of CNNs with the robust classification ability of SVMs. Research, including Agarap's work, has demonstrated the effectiveness of this hybrid approach in various datasets, particularly in the MNIST and Fashion-MNIST datasets. The significant accuracies achieved by this model underscore its potential in tasks requiring refined classification, benefiting from the linear SVM's superior generalization and decision boundaries for the features learned by the CNN.

The integration of CNN and SVM for image classification represents a significant step forward in tackling the challenges of high-dimensional data analysis. In situations where overfitting is a major concern, particularly with limited training data, the hybrid model shines. The SVM component, with its margin maximization principle, refines the decision boundaries established by the CNN, potentially leading to enhanced performance on new, unseen data. This approach not only capitalizes on the individual strengths of CNNs and SVMs but also addresses their respective limitations, creating a more robust and versatile classification system. The decision to use either a standalone CNN or SVM, or a hybrid of the two, should be influenced by factors such as the specific characteristics of the dataset, the available computational resources, and the unique demands of the classification task. This tailored approach ensures that the chosen model or combination of models is best suited to the specific challenges and requirements of the image classification project.

4. Conclusion

To recapitulate, this report covers two pivotal components, image processing and AI recognition. We first implement the image display, smoothing with averaging and rotating mask, sub-image generation, binary image creation with thresholding on histogram, edge detection, and character segmentation for the given image. These processes are essential and lay a foundation for the second part of the project, character recognition. For the second section, two deep learning models, CNN and SVM, were developed to train with the given dataset. The accuracy for these two models are 99.9% and 99.1% respectively. With this excellent training result, we succeeded in recognizing all the characters on the given chip image.

5. References

[1] R. Datla, "Adaptive fast spatial averaging filter," *8th International Conference on Computing, Communication and Networking Technologies (ICCCNT),* pp. 1-6, 2017.

[2] P. C. Y. Chen, "“ME5411 Robot Vision and AI Part II: Deep Learning for Computer-Vision in Robotics," in ME5411 Slides AY2324 Sem1," *National University of Singapre ,* 2022-2023.

[3] A. F. M. Agarap, "An Architecture Combining Convolutional Neural Network (CNN) and Support Vector Machine (SVM) for Image Classification".

[4] Y. G. a. P. Zhang, "Research on MNIST Handwritten Numbers Recognition based on CNN," *J. Phys.: Conf. Ser.,* vol. 2138, no. 012002, 2021.

[5] M. R. Phangtriastu et al., "Comparison Between Neural Network and Support Vector Machine in Optical Character Recognition," *Procedia Computer Science,* p. 116, 2017.