*Handwritten Character Recognition*

Patrick Rippl  
Rose-Hulman Institute of TechnologyTerre Haute (IN), USA  
[*ripplp@rose-hulman.edu*](mailto:ripplp@rose-hulman.edu)

Brandon Rudolph  
Rose-Hulman Institute of Technology  
Terre Haute (IN), USA  
*rudolpbw@rose-hulman.edu*

Yuankai Wang  
Rose-Hulman Institute of Technology  
Terre Haute (IN), USA  
*wangy16@rose-hulman.edu*

Chen Yin  
Rose-Hulman Institute of Technology  
Terre Haute (IN), USA  
*yinc@rose-hulman.edu*

*Abstract* — Optical character recognition (OCR) is an interesting topic in image recognition. Modern PDF readers are able to recognize standard computer text and thus apply standard operations like search or translation on these documents. This project involves the extension of already existing technology towards recognition of handwritten data. Handwritten data will be preprocessed and fed into AlexNet, a convolutional neural network (CNN), to train and classify different handwritten characters. Given the short timeframe allotted for this topic, there are certain restrictions regarding the types of recognizable characters. The current state of the classifier has relatively low accuracy, but future improvements, like increasing training data or changing the structure of the CNN, would likely improve performance.

Keywords — handwritten characters, binarization, padding, classification, AlexNet

# Introduction

Handwriting is unique to each person, with some people who write clear and distinct letters, and others who write letters that flow together. Optical Character Recognition (OCR) is a method of detecting typeset characters that has been around for a long time. Typeset characters are standardized and consistent, making them relatively easy to distinguish. Basic algorithms like those presented by Zhao and Daut [2] are able to interpret computer writing and classify the characters accordingly.

Recognizing handwriting is a much more difficult, and potentially very useful, task that can be handled with OCR. Successful handwriting recognition has a plethora of applications, like being able to automatically search for particular passages in a document. It can also be used to take images that contain passages of writing and transcribe them to a digital format. Another possible application is the facilitation of forwarding handwritten information to other persons. With the ability to transfer handwritten data into computer fonts, it becomes possible to change the font style or apply translators to documents. Moreover, this technique could be used to search handwritten documents in a more efficient way since specific information in handwritten documents would be found more easily using search functions, like those implemented in basic PDF readers. The availability of large sets of data and the high variability among different people’s handwriting poses a problem well suited to a CNN.

For a CNN to work well, a sufficiently large dataset is needed so the network can determine what features are unique to each character, allowing it to create an accurate classifier. However, some letters are similar and therefore confusing to the network, such as the number 0 and the letter O. Even a 7 can be interpreted as a T in some cases. Figure 1 shows two examples of characters that are occasionally confused by the network. Both letters have an ambiguous stroke, which does not normally appear in standard writing. It was extremely difficult for computer to determine the letter, which was understandable because several human readers also noticed the ambiguity. In Figure 1, the image on the left can be classified as either a T or a 7, and the picture on the right may be classified as a G, or separated in a C and a 7.

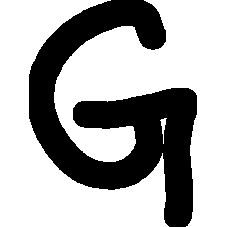
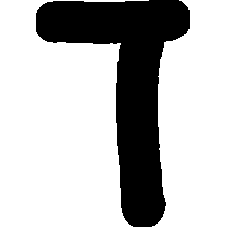


Figure 1: Problematic Letter Images

The primary challenge with the CNN was the processing of the data that was fed into the network. If the initial image contained multiple lines, each with multiple words, the lines had to be isolated and broken down into their constituent words. Within the individual words, each letter had to be identified as a separate image, processed to remove unwanted noise, and resized to be compatible with the CNN. The training images of individual letters were also reformatted using this method. Consistent character dimensions notably increased the accuracy of the classifier.

Due to the varied nature of writing styles, the distance threshold of the line segmentation method needed to be adjusted manually.

The network was trained with a considerably sized set of letter and number images to differentiate between 62 classes (26 letters of the alphabet, lowercase and capital, and numbers 0-9). The input images were taken, pre-processed, and fed into the net to classify.

# Literature Review

Some work has already been done in the field of handwriting recognition by Marti [1], Zhao [2], and Cohen [3].

Marti and Bunke presented a database for handwriting recognition with some segmentation and preprocessing procedures. The initial dataset used in this project is taken from their database. Marti and Buck took texts from a variety of sources (e.g. Press reports, fiction, religious texts) as a training data source, which included linguistic information in addition to the individual words and letters. Linguistic information was not used in the classifier discussed in this paper, but some of the line segmentation processes were applied.

Zhao and Daut investigated a method known as hit-or-miss shape detection that relies on preexisting knowledge of a detect shape. This was applicable to the classifier in this paper because the shapes, in this case letters and numbers, were already known. Variation of shapes complicates the process, but the authors treated imperfect shapes as regular shapes with noise present. They also varied their method to search for shapes within an expected window, which is less flexible, and without an expected window, which is a more robust approach. This method of shape detection was attempted with the assumption that the window for the shape was known, based on the bounding boxes found during preprocessing. Unfortunately, the results did not show any significant improvement to the original approach.

Cohen presented a new character dataset created by modifying an existing set from the National Institute of Standards and Technology known as MNIST. The new set, EMNIST, expanded the original set with extra character data. EMNIST contains character data for uppercase and lowercase letters, as well as numbers, which is exactly the information used in this project. EMNIST is a significantly larger dataset than the one used here and was therefore unrealistic to use due to time restrictions, but the dataset in this project was supplemented with some entries from EMNIST.

# Process

This project is comprised of two main parts: data preprocessing and CNN training.

## Image Preprocessing

By preprocessing the data all images used to train the network were consistently sized, which made extracted features more consistent and thus produced a more accurate classifier. Data augmentation allowed for a slight increase in the amount of available data, but for future implementations of the network a larger dataset would be preferable.

### Black and white images

Most input images began as color photos, which was unnecessary for the letter identification, so they were converted to grayscale. After the conversion all pixels below a certain intensity threshold were removed to reduce noise in the images. The threshold was determined based on the number of black pixels in the images after the grayscale conversion. Connected components with a number of black pixels significantly below the threshold were deleted. Special care was taken for larger dots above connected components because they might represent the dot of the lower letters *‘i’* or *‘j’*.

### Image Segmentation

For test images that contained multiple words and/or lines, the main image was divided into smaller images of single words so the words could be processed individually. To preserve information about different words and lines, each word was organized according to the line containing it. After organizing the words each one was separated into single letters, which were then given to the network.

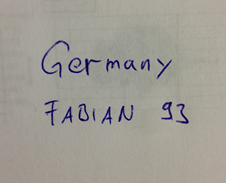


Figure 2: Sample Image of Handwriting

Figure 2 shows a sample image with multiple lines that was given to the network to classify. The background was relatively simple, which was consistent with the images used to train the network. The thresholds were manually selected to get good segmentation. Letters on the same line had similar average values on the x-axis, which indicated there were letters that belonged to the same line. If the difference between two consecutive values exceeds a certain threshold, the image was divided into separate lines. Figure 3 shows the original images after grayscale conversion and line separation.

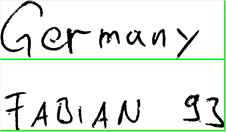


Figure 3: Image after Binary Conversion and Line Separation

The average position of the connected components on the y-axis was then found. If consecutive letters belonged to word, then the y-position increased at an approximately constant interval. When the interval suddenly increased, it was treated as a space between words. Figure 4 shows the second line after segmentation.

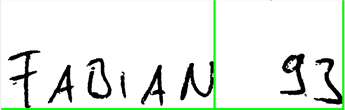


Figure 4: Line Segmentation

Any crossover of letters between boundaries was again removed. Letters were found by searching for connected components. Because not all letters are composed of only one component, specifically ‘i’ and ‘j’, the average position along the x-axis was again found for all the pixels that are not 0.

The average of the average values for position of components was calculated and treated as a standard line for a letter. If a letter consisted of only one component, its average position should be about the standard we set. If we find one component has an average that differs significantly from the average, it was assumed to be the dot of “i” or “j”. The disconnected component was combined with the nearest component according to the position on y-axis. Figure 5 shows an example of how letters within a word are separated.



Figure 5: Bounding boxes for separating letters

When storing the images, the folder hierarchy followed the images moving left to right, top to bottom, consistent with how the words were written.

### Padding Function

To feed the segmented characters and training data into the CNN, the images had to be rescaled to the appropriate dimensions. Special care had to be taken to preserve the shape information of each character.

Since AlexNet was chosen, the input images had to be scaled to a size of 227x227x3. A padding algorithm, which is depicted in Figures 6-8, handled this resizing, while maintaining the overall shape of the input characters. Some letters may be written taller than they are wide and vice versa, which had to be taken into account for the resizing process. The bounding box for the letter was found and resized until the larger of the two dimensions was 227. The resized image was padded with whitespace until the smaller dimension was also 227.

|  |  |  |
| --- | --- | --- |
| Figure 6 Sample image of training data from [1] | Figure 7 Sample image cropped to a height of 227 pixels | Figure 8 Sample image padded to 227x227 |

Using this method, it is possible to train the CNN with characters that retain more information about their shape than they would from just simple resizing.

A drawback of this padding method was confusion between characters that have similar shapes but different sizes, as is the case with several capital and lowercase letters. After being resized to 227x227 some lowercase letters became indistinguishable from their capital counterparts, as shown in Figure 9.

|  |
| --- |
|  |

Figure 9 Capital 'W' on the left; small 'w' on the right.

This loss of information was considered acceptable because the network was able to reach a higher overall accuracy, and these characters are also occasionally confused by human readers. Further development of the process can investigate storing the height information of the input images before applying the padding function to them, thus keeping the information about the relative height between letters. This extra information would then enable the algorithm to specify between capital and small letters in given input images.

## Convolutional Neural Network Classification

Training on the initial dataset, initially 55 images of each of the 62 characters with a high resolution were preprocessed and then fed into the AlexNet CNN for transfer learning.

|  |  |
| --- | --- |
| A close up of a logo  Description generated with high confidence | A close up of a logo  Description generated with very high confidence |
| A close up of ware  Description generated with high confidence | A close up of a logo  Description generated with high confidence |
| A close up of a logo  Description generated with high confidence | A close up of a logo  Description generated with high confidence |

Figure 10 Sample images of dataset [1]

Figure 10 shows some of the variation in the training dataset. To get a consistent dataset, all the letters that were fed into the CNN had to be rescaled accordingly such that the part of the image that did not contain important information was removed prior to the training phase. To do so, the letter itself was cut out of the image and centered using the padding algorithm from II.B.2).

The reason AlexNet was chosen for transfer learning is the simple structure which makes it easier and quicker to train a sufficiently accurate net. The approach of using transfer learning is useful since features will automatically be extracted with only minimal changes for a new classification problem. The last three layers were changed so the network could distinguish 62 classes.

The AlexNet was trained on a NVIDIA GeForce 940MX GPU. There were 3410 images overall used for training. Since they were split into two groups by a ratio of 0.7, a training dataset of 39 images per character and a validation set of 16 images per character was used.

Table 1 Hyper-parameter Ranges

|  |  |
| --- | --- |
| **Parameter** | **AlexNet** |
| Min. Epochs | 1 |
| Max. Epochs | 15 |
| Min. Mini-batch Size | 32 |
| Max. Mini-batch Size | 128 |

After varying the hyper-parameters, the best result was achieved with the combination of 10 epochs and a mini-batch size of 128. The accuracy on the validation set at this point was 94.35%. As previously mentioned, this trained net performed rather poorly on given test images due to the small variations in such a small train set. Table 1 shows the ranges of the varied hyper-parameters.

# Experimental setup

The initial dataset used is the one presented by Marti and Bunke [1], which contains 55 1200x900 images of each character. Since the first dataset did not provide satisfying results, the set was modified and extended by five, manually selected images from the EMNIST set presented by Cohen [3] for each character. This modification increased the accuracy on single character recognition significantly.

Before test images were fed into the network, they were processed to separate words and paragraphs into single characters, which are then classified by the trained CNN.

Several different preprocessing steps had to be applied in order to feed appropriate data into the net for training and classification. In addition to these implementations, the character segmentation was a big challenge since input data has a high degree of variability (resolution, noise, cursive handwriting etc.).

To feed the segmented characters and the train data into our CNN the images had to be rescaled to the appropriate dimensions. To maintain the specific shape of characters, the input images had to be accordingly scaled using a distinct method.

An improvement to this restriction will be given in the

In some cases, an ambiguous character that can be considered one of two characters can be post processed to improve the accuracy of the classifier.

The final CNN used the pre-trained network AlexNet for transfer learning. The final three layers were modified to fit the task of handwriting classification. The classifier assumes that the images being fed into contains only handwriting, with no extraneous shapes or backgrounds.

# Results

The final network used was tested on the authors’ handwriting to evaluate the performance.

A close up of text on a whiteboard

Description generated with very high confidence

Figure 11: Test image of own handwriting

Figure 11 shows a test image of a handwritten input taken by an ordinary phone camera without preprocessing. The network was only trained on images of letters and numbers on a blank background, so the test picture was taken so that it did not contain any difficult illumination traits or shadows. Moreover, the characters are fairly easy to separate and there were no lines that could disturb the connected component analysis.

After this image was fed into the segmentation algorithm and the incorrectly segmented images were removed, the following characters were stored:

|  |
| --- |
|  |
|  |
|  |

Figure 12: Results from segmentation of own handwriting

As can be seen in Figure 12, the segmentation still had some errors due to the connected characters *‘er’* in the word *‘Germany’*. Furthermore, *‘F’* and *‘A’* were not separated and hence were not recognizable by the CNN. The preprocessing could be extended by checking the images again to determine if there is still more than one letter in the input image, and by working recursively on the given input image until all letters are separated. The letter *‘B’* in line two was damaged and looks a lot like *‘3’* afterwards. These assumptions and limitations, as well as the classification results, are depicted in 2.

Table 2 Results on own handwriting

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Germany** (‘Grmany’) | |  | **Fabian** (‘3IAN’) | |  | **93** | |  |
| Ch | Ce | | Ch | Ce | | Ch | Ce | |
| *‘S’* | 51% | | *‘3’* | 33% | | *g* | 63% | |
| *‘j’* | 28% | | *‘I’* | 44% | | *3* | 36% | |
| *‘m’* | 86% | | *‘A’* | 28% | |  |  | |
| *‘W’* | 58% | | *‘U’* | 91% | |  |  | |
| *‘n’* | 96% | |  |  | |  |  | |
| *‘Y’* | 64% | |  |  | |  |  | |

Since the *‘N’* looks like *‘U’*, and the *‘9’* could be interpreted as a *‘g’*, these results could be interpreted as minor mistakes and thus improved through post-processing methods. As a result, an absolute true-positive rate of 58% or 75%, including minor mistakes in the yellow cells, can be achieved for the given segmented image. Considering the overall accuracy with the segmentation error this would result in a true-positive rate of 6/15 (= 40%) and 9/15 (= 60%), respectively.

# Discussion

Even though the CNN reached an accuracy of ~90% on the validation set, the classification on test images was rather poor. To create a more robust system, it was assumed that either a bigger data set for training or a different method of preprocessing the training set would lead to better results. An alternate preprocessing method was tested, but it did not produce a better result.

This approach will be further discussed in section III.A.1).

The results of the trained network were obtained by using images with easily separable letters, as shown in Figure 13.

|  |
| --- |
|  |

Figure 13: Test images

The improved separation function with flexible thresholds segmented the words as seen in Figure 14.

|  |
| --- |
|  |
|  |
|  |

Figure 14: Results from segmentation on test images

All the single characters were nicely separated into individual 227x227 image files with the applied padding algorithm. In the following tables, red cells represent significantly bad results. Yellow cells indicate a result which is reasonable enough to get and could be improved with a better pre- or post-processing algorithm. Green cells represent correctly identified characters, even if they might be wrong considering capital and small letters. Since the padding algorithm lost information about capital and lowercase letters, these ‘false’ classifications can be regarded as correctly classified. As a measure of certainty on correctly classified characters, the equation

was used. The classification of each of the three words above produced the results in Table 3.

Table 3 Results on test images

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **wearing** | |  | **Royal** | |  | **which** | |  |
| Character (Ch) | Certainty (Ce) | | Ch | Ce | | Ch | Ce | |
| *‘N’* | 35% | | *‘T’* | 20 % | | *‘W’* | 96% | |
| *‘C’* | 44% | | *‘C’* | 28 % | | *‘k’* | 63% | |
| *‘Q’* | 86% | | *‘J’* | 65 % | | *‘j’* | 35% | |
| *‘F’* | 57% | | *‘Q’* | 32 % | | *‘C’* | 92% | |
| *‘I’* | 44% | | *‘I’* | 68 % | | *‘L’* | 62% | |
| *‘r’* | 62% | |  |  | |  |  | |
| *‘q’* | 55% | |  |  | |  |  | |

Of the 17 characters in the images, 6 characters were incorrectly recognized with no acceptable error. The certainty of the falsely recognized characters was above 50% in three of those cases. Due to the low true-positive rate of 18%, it would also be difficult to improve the results using post-processing methods, such as built-in probability measures of typical character sequences or built-in dictionaries.

## Improvements of CNN

After reviewing the first results, it could be stated that they were not satisfying enough. Various ideas of different implementations were considered. These ideas will be described in this section.

### Using thinned characters

According to Zhao and Daut [2], the implementation of a hit-and-miss algorithm would increase the probability of finding specific characters and therefore facilitate the single character segmentation of words and paragraphs. A new dataset for the training of another net was created for this purpose. Some of the newly constructed images are shown in Figure 15.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Figure 15: Thinned characters

After training a network on the newly formed training images, the net was given test images that were preprocessed with the same thinning algorithm. This method could not reach a higher performance. This preprocessing method may work with a different network architecture, so it was not entirely discarded.

### Bigger dataset (Augmentation)

Because the dataset of only 55 images did not lead to satisfactory results, another idea was to increase the number of images in the training set. To increase the dataset, five images of each of the 62 characters were manually picked from the EMNIST dataset and modified in the same way as the prior data to maintain consistency. After increasing the number of images for each character to 60, the images were augmented through a mix of small translations, rotations and shear effects.

In Figure 16 the characters are augmented through the application of one single change in either translation, rotation or shear. In the training set used by the network, combinations of these augmentation transformations were included.

|  |  |
| --- | --- |
|  |  |
|  |  |

Figure 16: Samples of data augmentation (shear x-direction, shear y-direction, rotation and translation in x-direction)

After training a new CNN using the augmented training dataset with a ratio of 0.7 for training and 0.3 for validation data, a result of 95.31% accuracy was reached with a mini-batch size of 128 and 15 epochs.

## Data Analysis

Referring to the previous three test images, this network produced significantly more accurate results, as depicted in Table 4.

Table 4 Results on test images using the modified train data set

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **wearing** |  | **Royal** |  | **which** |  |
| Character (Ch) | Certainty (Ce) | Ch | Ce | Ch | Ce |
| *‘W’* | 94% | *‘Q’* | 38% | *‘W’* | 99% |
| *‘Z’* | 28% | *‘Q’* | 39% | *‘W’* | 39% |
| *‘Q’* | 75% | *‘y’* | 68% | *‘t’* | 45% |
| *‘R’* | 63% | *‘Q’* | 81% | *‘C’* | 73% |
| *‘I’* | 40% | *‘t’* | 55% | *‘W’* | 73% |
| *‘N’* | 30% |  |  |  |  |
| *‘9’* | 54% |  |  |  |  |

Of the 17 characters, 4 were incorrectly classified with no acceptable error. Three of those four failures have a certainty of less than 40%. The true-positive rate on the given data is 41%. If the yellow colored cells are considered as acceptably classified characters, a true-positive rate of 76% is achieved. If the functionality of comparing the yellow classified characters is included with a built-in probability distribution of common character sequences, some of the falsely identified characters could be corrected. For example, in ‘wearing’ it is a higher probability that the last character is a letter instead of a number. Implementing this functionality would improve the overall accuracy of the algorithm. Given the ability to compare this data in a second step to a built-in dictionary, the original word could be guessed.

# Key Challenges

The implementation of handwriting OCR was problematic for multiple reasons, primarily segmenting connected letters, recognizing different writing styles, and ambiguous characters.

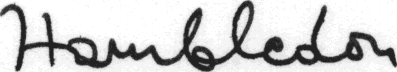
## Different writing styles

### Difficulty with consistent segmentation for connected letters

One of the differences between writing styles is space between words, letters, and connected strokes. Because single letter images were fed to the network and classified, there was not a universal method to segment the letters. The threshold used to segment lines and words is highly dependent on writing style. For each classification, the threshold had to be manually set.

### Recognizing writing style

Some writing styles distort the main features of particular characters. Specifically excluding strokes, connecting every letter, and over simplifying characters often results in failure of classification.



**Figure 17: Example of complicated writing style**

Figure 17 shows a writing style that is difficult to segment and classify. All characters are connected to each other, making segmentation difficult.

## Ambiguous characters

Even with relatively simple writing styles, some characters are difficult to even by human readers. Even with training on every letter of the alphabet, the network made frequent mistakes for some letters.

Figure 18 shows some examples of ambiguous characters.

|  |  |
| --- | --- |
| C:\Users\yinc\Dropbox\TermProject\Coding\TrainImages\u_s\6.png  (a) lower case u | C:\Users\yinc\Dropbox\TermProject\Coding\TrainImages\O\18.png  (b) upper case O |

**Figure 18(a)-(b): Problematic character images**

Figure 18(a) belongs to lower case u class, butit is easy to confuse with lower case n. Figure 18(b) belongs to upper case ‘O’, which is easy to confuse with ‘0’ or lowercase ‘o’. Ambiguous characters are unavoidable in handwriting recognition, so some compromise in accuracy has to be made.

# Future Work

Given more time, there are multiple possibilities that can improve the overall performance of the handwritten recognition algorithm. First, the preprocessing could be improved. Second, the CNN could be adapted to specific handwriting or extended with a larger training set. Third, post-processing methods could be implemented to reinterpret the classified data afterwards and adjusting the results base on given probabilities specific to the language. Finally, a more user-friendly environment could be implemented to enable inexperienced users to operate the system.

## Improve Preprocessing

The preprocessing of the images could be improved regarding the following topics.

### Thin out images, implement hit and miss

Due to the limited time, it was not possible to correctly implement the whole algorithm discussed by Zhao and Daut [2]. Early attempts to implement parts of the algorithm by thinning out the characters for handwritten character recognition did not seem to have a beneficial effect on the results. However, this method could be further investigated and might deliver some features that could be beneficial in preprocessing the images.

### Images with different resolution. Find a way to detect areas with characters in the beginning

To adapt the letter detection technique for practical use, like taking some pictures for the words and turning them into a text file in computer. The current method depends on setting reasonable thresholds to determine the standard of dividing letters. However, it is obvious that using fixed thresholds is not a reasonable way. For any given input image, the word spacing might be different. It is impractical to change the threshold every time when a new picture is given to the network. Ideally the preprocessing algorithm would set the threshold automatically.

Multiple methods were tested to determine the thresholds for lines, words, and letters. In one method, one input labeled ‘lines’ was added to show the number of rows in the picture. This allowed for the calculation of the threshold for lines by the number of rows and size of the picture. It was assumed that the word threshold proportional to the line threshold, because people tend to write with consistent intervals between words. A coefficient was set that multiplies the line threshold to calculate it. It was then assumed that the interval between words is in proportion to the size of the letters, so the same technique was applied to the letter threshold.

Line and word spacing is heavily dependent on the style of the writer. This means the coefficients necessary to set the line space threshold vary for different writing styles. For one style the word threshold was set to be 1.3 times the line threshold, which provided great result, but for other styles 2 is be a more reasonable coefficient.

A method in which the input image was resized to make all the inputs have the same size was test, but it still had a similar problem to the previous method.

Another possible method is to include the information about the portion of the black pixels in the picture. If it is small, there may be fewer words in the picture. The words may be more separated, in which case a large threshold is needed. This approach may not work well on pictures that only have a few letters on it, because it might be influenced by the size of each letter.

Finally, another method is to get a set of pre-defined writing styles and choose the style of the picture to determine the coefficients needed.

### Split the connected characters into single characters

Because connected components are used to determine the boundary box for each letter, if any letters are connected then they will be detected as one letter. It was assumed that all the letters should be “slim”, meaning the number of rows of its bounding box should be larger than its number of columns. If any bounding box detected does not satisfy this condition, then it is divided into two boxes.

Another possible method is to count the number of non-zero pixels in each row. If the number is small, it can be assumed that it is just a connection between two letters and can therefore be separated.

### Specification between text and non-text parts in an image

In practical applications, images of text are not just purely text, but also include graphics or other shapes. One approach to separate given images in possible parts containing text is the usage of a detector CNN to identify areas of text in an image. These areas would then either be fed into another CNN for character segmentation or could serve as input to the segmentation algorithm that was described in this paper. Using this approach, it would be possible to investigate entire documents and search for handwritten passages.

## Train AlexNet on user-specific handwriting

Apart from pre-processing improvements, the CNN could be modified such that it recognizes specific user dependent handwriting. The idea is to train the net on the specific style so that it will be able to recognize their writing in other input words as well. Training the net on one style would mean that the training set has to be extended with enough information about the user’s handwritten single characters.

## Further extension of train data

Higher accuracy on the training net in the beginning could be reached either with improved preprocessing and augmentation, as already mentioned, but also with an extension of the existing train data set. A set of 3720 images for 62 classes is not very large, which limits the overall feature training. Even though augmentation is used, the style of different handwritten characters may considerably. Furthermore, it could be helpful to train different CNN layouts and compare the results of each of them.

## Embed character sequency probablility & dictionary

It could be helpful to ‘mark’ classified characters that could easily be misinterpreted, e.g. ‘9’, ‘g’ or ‘c’, ‘e’. Implementing probabilities of sequences could improve the result significantly. Additionally, a dictionary could then be used to improve the results on word recognition even more. It has to be mentioned that these improvements only work if the input language is given and the input data consists of real words. For a random combination of letters and numbers, this approach would not help.

## Improve Design & Usability

Given more time, a nice feature would be to embed the current MATLAB code within a graphical user interface (GUI). The built-in GUI functions in MATLAB provide quick implementation methods for GUIs. The implementation of a GUI would enable potential future users to easily feed their images in and set certain adjustable parameters without having the knowledge about the programmed algorithms. Inexperienced users would then have a chance to train their net with their handwriting, change threshold parameters, or just feed their own images in without modifying the MATLAB code themselves.

# References

[1] U. Marti and H. Bunke. The IAM-database: An English Sentence Database for Off-line Handwriting Recognition. Int. Journal on Document Analysis and Recognition, Volume 5, pages 39 - 46, 2002.

[2] D. Zhao and D. Daut, Morphological Hit-or-Miss Transformation for Shape Recognition, Journal of visual communication and image representation, Volume 2, No. 3, pages 230 - 243, 1991.

[3] Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. Retrieved from http://arxiv.org/abs/1702.05373.pdf