A Hybrid Method for Chinese Character Segmentation

J.F. van Wezel

University of Groningen

September 29, 2017

Abstract

This work shows a hybrid method between connected components and horizontal vertical projection for Chinese character segmentation with mixed results. There are some problems with the implementation of the system and these problems are located and discussed. Even with these problems the work shows that a hybrid method is able to achieve segmentation with the best of both worlds.Trudeke

Character Segmentation - Text recognition - Connected Components - Chinese Characters - Horizontal vertical projection - Computer vision - Machine Learning

1 Introduction

Character recognition is an important part of the fields computer vision and artificial intelligence. For a machine to recognize sentences, words ,and characters in a given image is useful for multiple goals. For example to digitize handwritten texts ([8]). digitized texts have a number of advantages over physical texts. An important advantage is that digital texts are more easily distributed and reproduced. It is also easier to search in digital texts. But probably the greatest advantage is storage. It is easier and takes less room to save and backup a few bytes on a disk than it is to physically backup archive the same texts. As is often the case in the fields computer vision and artificial intelligence, compression is the end goal.

An other practical application of character recognition is text translation ([3]). Text recognition for the use of translation is useful for example tourists who use their smart-phone to take a picture of a sign and have that sign translated in their native language by an translation app. Or even better, have their text translated and inserted into the camera feed directly. This form of augmented reality is still in its infantsy but being able to recognize and digitize texts on walls, signs ,and posters for all kinds of purposes is not that hard to imagine.

Usage of character recognition can also be imagined in the retail industry. Here it can be used for label recognition on certain products, currency ,and identification purposes for restricted product groups (like liquor and tobacco).

The automotive industry uses character recognition in their autonomous cars ([6]). They read speed limits and other road signs to form a state of awareness of their surroundings They abstract what the next course of action should be based partly on that gathered state. A normal road sign is not a character but a computer does not know the difference between 'human' characters and a specific image. The important part is that the goal is to find meaning in an image by 'reading' parts of that image. Sign recognition and character recognition are therefor closely related. They differ in that characters tend to come in a sequence forming sentences. Where road signs are mostly isolated.

Staying on the road character recognition is used for numberplate recognition ([5]). It is used by law enforcers to find stolen or unregister vehicles but also as an automated form of speed control. Here a mounted or portable camera is used to take pictures of cars with their number plates and the software recognizes the plates and the characters on them to identify the vehicle.

A character recognition system can be roughly divided into three main components: segmentation, feature extraction, and classification.

Segmentation is where an algorithm tries to find the location of the character or sentences in a given image. Before it can start this task the image often is preprocessed. During preprocessing, an image can be binerized and filtered to reduct small noise and other unwanted large components that might be present in the image. Preprocessing can also include rotating, shearing or other morphological operations.

Feature extraction is where the features of the segmented character images are found and measured. For example if an image is black and white a feature could be the ratio between black and white pixels. An other example of a feature could be the amount of horizontal edges in the image by using an edge detector. The type of features that should be extracted depends heavily on the type of dataset. A western handwritten scroll houses different characters and features than Egyptian hieroglyphs.

Classification is where the, until then unknown, characters get assigned a label. This task can be done by a number of algorithms from the simple K-nearest neighbors, used for classification and regression ([1]), to complicated multilayer convolutional neural networks. An example of such a CNN is LeNet-5 by Lecun ([9]). Which algorithm yields the best results depends on the dataset, the segmentation, and the feature extraction.

This paper will focus on the building of pragmatic preprocessing and segmentation system for a Chinese character dataset. There has been work done in the past on Chinese character segmentation by others. From this work we know there are multiple ways of segmenting the Chinese characters. Examples of methods used in the past are: vertical and horizontal projection based, recognition based, skeleton based, and connected component based [4].

Recognition based uses a learning algorithm to find the locations of the characters, Guo et al. proposes a self learning license plate method ([7]). Skeleton based uses the skeleton of the image to find the individual characters in a image. A variant of this technique has been demonstrated by Nikolaou et al. ([10]). Component based uses the connected components of an image to find the characters. An example of this method is shown by Chen et al. ([4])

For the created system a combination of various prepossessing techniques, horizontal projection, and connected components was used to segment the dataset. This results in a pragmatic segmentation system that was designed for the given dataset but should also be useful for other Chinese character datasets. The segmented images are used for feature extraction and will be shown his paper by Lennart van de Guchte and classification with a CNN of the data with the segmented images by Leon Pater.

This work will use labeled data from the given dataset to validate the resulting segmented images. The use of already used methods that were shown to have worked in the shown related work will likely result in a reliable segmentation system.

This paper will continue by explaining a bit more about the dataset. Then the used methods and used parameters will be further explained. After, the type of experiment and how the segmented images were validated will be explained in the Experiment section. From the experiment the results will be shown and discussed. This paper will end with a conclusion on the work done.

2 Dataset

This section will shed light on the used Chinese character dataset. It will show some metrics, examples of the noise encountered, and other problems that needed to be solved to achieve the segmented images.

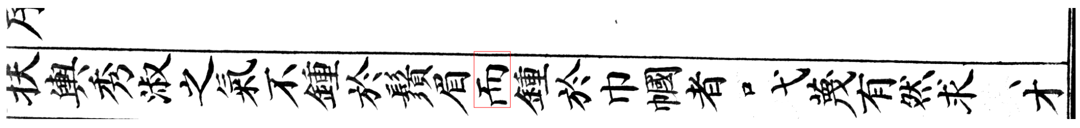


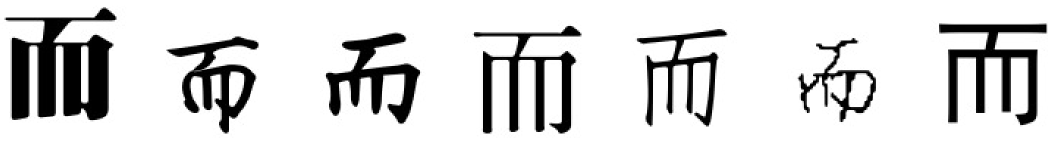
Figure 1: Example of a typical image in the dataset

Table 1: Dataset metrics

Size	Labels	Unique labels	Unique characters	Fonts	μ_{lw}	σ_{lw}
6000	27026	589	6500	7	73.5944 px	8.4463 px

The metrics of the used dataset for segmentation

Table 1 shows some metrics from the dataset. The size (6000) is smaller than the number of labels (27026). The size is based on the number of images the dataset holds. The images in the dataset consist of unsegmented lines of Chinese characters. An example of a typical image is shown in figure 1. The red box in the image shows the location of an a labeled character. The locations of these labeled characters are given in xml files along with the line images. The μ_{lw} and σ_{lw} signify the mean with of the labels and the standard variation in width of the labels in the dataset.



(a) ChaoMing **(b)** CuoKai **(c)** HeiTi **(d)** MingTi **(e)** TeMing **(f)** XinSong **(g)** YanKai

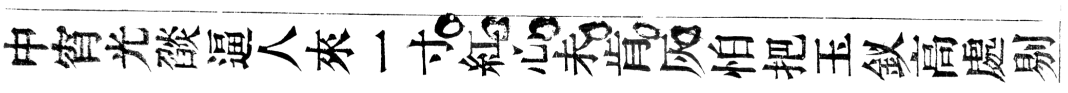
Figure 2: The Chinese printed fonts present in the dataset.

The fonts used in the dataset are shown in figure 2. There are seven fonts in total, which show a similarity with the characters in the images in the dataset. It seems assumable from this observation that the characters were printed instead of handwritten.

The images in the dataset are rows of characters but as shown in figure 1 contain sometimes an extra row with one or more characters in it. Sometimes there is whitespace on one or all sides of the image and sometimes the whole image is noise. The character rows are more or less given but the exact location of the the character row needs still to be found in the image.

The Chinese language is generally written from top to bottom. The orientation of the images is however in the western orientation of the language, from left to right. Because the whole dataset was in the western orientation the dataset will be handled as such. The horizontal line present in the image is to separate the character rows. This line is not present in all images. The image also shows that the rotation of the image is not straight, it is rotated by a few degrees. The characters them self however do not seem to show a curve. This is also not true for all images, some images do show a slight curvature. The rotation is probably a result of the scanning process. This is mostly done by hand and prone to error. The line could be abused to find the correct rotation of the image. But as stated the line is not present in all images and a different approach would be needed for the images without the line.

Figure 1 also shows a 'fat' vertical line on the far right side of the image. This line is also present in most images but not all. Sometimes this line shows up on the far left side of the image or on both sides. This line is not a character and needs to be handled as noise. but it can also be used to find the starting and end point of a character row.


(a) Partial noise

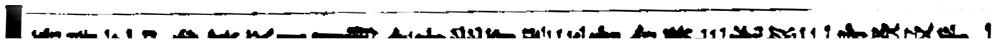

(b) Whole image noise

Figure 3: Examples of noise found in the dataset

The image shown in figure 3a illustrates noise on the center characters often found in this dataset. This noise might be the result of water damage or perforation of the sheet where the characters were originally on. This noise could be resolved with morphological operations and using a dataset specific filter. The dataset also contains some images that are by them self noise. Figure 3b shows an image that is complete noise. This specific image seems to be located to low during the scanning process. This resulted in only half of the original characters showing in the dataset.

3 Method

This section will show what methods where used to segment the dataset. It will do so in a chronological order based on the example image shown in section 2 in figure 1.

3.1 Segmentation

3.1.1 Rotation and binarization

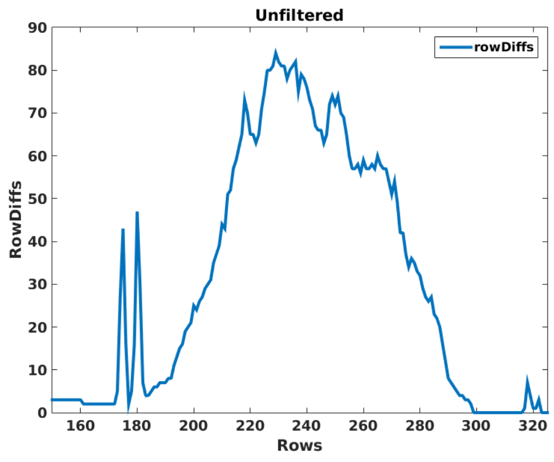
As discussed in the introduction (1) the Chinese characters were supplied in greyscale strokes of characters in the western orientation (from left to right).

In order to segment the characters from an image a combination of methods were used. First the image is binarized. Binarization is done by applying a Gaussian filter on the image to remove some of the pixel noise. Then Otsu's [11] method of finding a threshold is used. The image is then binarized using this threshold.

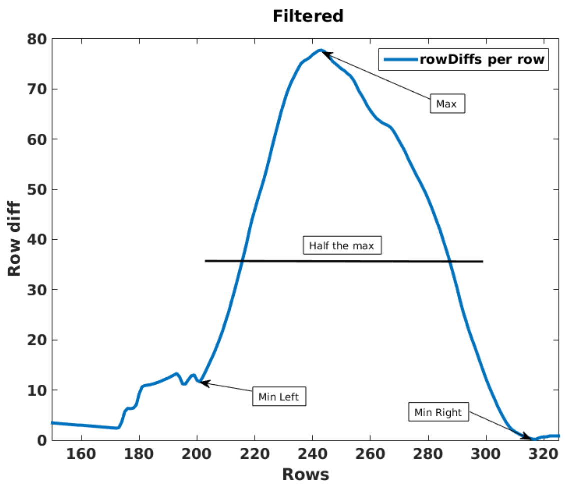
After binarization the image is rotated. This is done because most images in the dataset are titled slightly. As discussed in the dataset section (2) This is probably due to some variations in the orientation of the paper during the scanning process. Rotation is done by resizing the image in the vertical direction to half its size. The idea behind this is that the characters will be squeezed together and form a horizontal line, this line can then be used to find the optimal rotation. This is done by taking the horizontal densities. The maximum densities are saved for the different rotations (between $+1°$ and $-1°$). The rotation with the largest maximum density is where the line is likely to be the most horizontal. This is taken as the optimal rotation. The image is rotated with this optimal angle.

3.1.2 Vertical location

After the characters are rotated the location of the characters is determined on the vertical axes. The used method to determine this location is horizontal projection on the changes in pixels from black to white. On the densities an averaging filter is first applied. Each row is averaged with ten of its neighbor rows. To illustrate this the changes in densities were taken on the example image



(a) Unfiltered density changes



(b) Filtered density changes

Figure 4: Method of finding the vertical location of the characters

shown in the dataset section (1). The effect of the averaging is clearly visible in figure 4. In figure 4a the unfiltered image is show. The plot has many peaks and also the top black line is clearly visible in the densities on the left side of the image. In figure 4b the filtered image is shown. Only the global peaks stay and the noise peak from the black line has almost disappeared.

$$S = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

Figure 5: Sobel opperator

In order to find the location of the characters, the global maximum of changes in densities is taken. This is always the top of the highest peak of the characters. To find the bases on both sides of the peak a Sobel operator is used (shown in figure 5). Sobel's operators can be used to find the gradient in a signal, demonstrated in a paper by Richard O. Duda and Peter E. Hart ([2]) and later by Sobel self ([13]). To find these minima The half of the max is taken and from this half the first occurrence where the gradient crosses zero in the left and right direction is taken as a base of the mountain. The half of the max is used because on the top of the plot sometimes goes up and down even with the average filtering. The half max makes sure the whole height of the characters is taken. After determining the vertical location of the characters the image is cropped accordingly. Whitespace on the left and right side of the image is also removed.

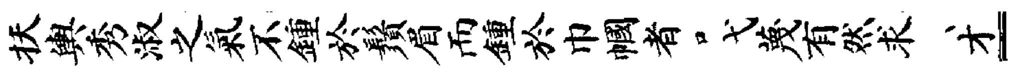


Figure 6: Rotated and cropped image

3.1.3 Connected components

We are left with the row of characters. An example is shown in figure 6. This image is rotated and cropped from figure 1. In order to find the characters in this row the connected components are taken. These are taken with the method described by Sedgewick ([12]). Sedgewick's method will give us a list of all the connected components in the image. If multiple components are above each other we assume that these components are part of the same character. These components are merged. Some components are not directly on top of each-other. It is determent how much the components overlap vertically. If the overlapping width is larger or the same as 0.4 times the width of the smaller of the two components, they are merged. This is done to make sure the components are on top of each other and not some small part of one component is on top of an other component.

After merging vertically we are left with a list of components that might be outliers in terms of width in comparison with the mean width of the characters in the supplied labeled dataset. In order to recognize outliers the mean and standard deviation was calculated from the characters in the labeled dataset, also shown in section 2 and table 1. A component is deemed a small out-lier if:

$$C_w < \mu_{lw} + 2\sigma_{lw} \tag{1}$$

and a big out-lier if:

$$C_w > \mu_{lw} + 2\sigma_{lw} \tag{2}$$

Here C_w is the component's width, μ_{lw} is the mean width ,and σ_{lw} is the standard deviation in width. Both were taken from the labels of the dataset

Small component can be noise or a character. A small component is more likely to be a character if it has an big amount of whitespace left and right from it. Except when it is the first or the last component in the list. To check for this the amount of whitespace on both sides of the small component is measured and added to the with of that component. If the component is the first or the last component in the list the white space is only measured on one side and is added twice to the width. With the added with the component reevaluated for being an out-lier.

The list with components still holds noise components. There are different types of noise present in the images. This algorithm tries to find unwanted 'blobs' and too small components. The unwanted 'blobs' are detected by taking the square of the component and filling it with white pixels. The component itself is then printed on this square in black pixels. The component in determined to be noise if the rectangle consists of more than 40 percent black pixels. This number is based on the labels of the dataset where the amount of black pixels in the rectangle was always lower than 40 percent of the pixels. This method removes components like the black bar shown in figure 6 on the outer right side of the image. An image is deemed noise if the width of the character is smaller than 15 pixels.

The small components that are left in the list are now merged with the smallest of its two neighbor components. If a neighboring component is deemed big a small component will not be merged with it. If a component has a big neighbor to the left and a big neighbor to the right, This small component will not be merged. After each merger the components widths will be evaluated. If two merged components are still small, the algorithm will try to merge them again with one of the (new) smallest neighbors.

After merging all small components are dealt with. The component list still contains big outliers. Some might even been created during merging. These components are spitted by projecting the variations in white pixels to black pixels (or the amount of fluctuations in white pixels). Where there are less variations there is more white space. The characters are split on where the fluctuations of the whitespace is minimal. After splitting the components are reevaluated on width. If a component is determined big it is split again. This is done until the component is not big anymore.

3.1.4 Finding the character box

With the split big components the segmentation part is completed. In order to validate the segmented images the location of the characters need to be determined in the original image. Because the image is rotated before segmentation the locations of the segmented images need to be rotated to point to the correct location in the original image. The location of the segmented characters is given by taking the *x*,*y*,*width* and *height* variables. This creates a rectangle on the original image where a character is determined to be. But because the characters outputted by the segmentation system are (merged) connected components there might be parts of other characters present in the rectangle that are not present in the character outputted by the segmentation system.

The approximation of the rectangle is made by calculating all corner coordinates of the rectangle by taking the max and min row and column of the characters in the rotated image. Then these coordinates are rotated with the following formulas:

$$x' = y \times sin(\alpha) + x \times cos(\alpha) \tag{3}$$

$$y' = y \times cos(\alpha) - x \times sin(\alpha) \tag{4}$$

After rotating the minimum value for x' and y' is taken to determine the upper left corner of the rectangle. Then the width is determined by:

$$width = abs(min(x_1, x_2, x_3, x_4) - max(x_1, x_2, x_3, x_4)) \tag{5}$$

The height is calculated in a similar fashion:

$$height = abs(min(y_1, y_2, y_3, y_4) - max(y_1, y_2, y_3, y_4)) \tag{6}$$

These equations give the *x*,*y*,*width* and *height* variables needed to reconstruct the rectangle. After approximating the rectangle the characters are saved for use in the feature extractor and later the classifier of the character segmentation system.

3.2 Feature Extraction

Because the focus of this system lies on the segmentation and the classification a simple feature extractor was designed. The feature extractor uses boxing and edge detection to construct 22 features.

Each character image is divided into four boxes. (inspired by ... paper) Per box the edges in four directions are calculated with a Sobel or a Prewit filter.

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

(a) $S_{Vertical}$

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

(b) $S_{Horizontal}$

$$\begin{bmatrix} -1 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

(c) $S_{diagonal-left}$

$$\begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{bmatrix}$$

(d) $S_{diagonal-right}$

Figure 7: Sobel opperator in four directions.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

(a) $P_{Vertical}$

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

(b) $P_{Horizontal}$

$$\begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

(c) $P_{diagonal-left}$

$$\begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix}$$

(d) $P_{diagonal-right}$

Figure 8: Prewit opperator in four directions.

The filtered images are binary images with the edges in a particular direction. The percentage of black pixels in the box is calculated and taken as a feature. Per box the percentage of black pixels is taken without filtering. This leads to five features per box in four boxes. Finally the height and the with of the image is taken as the last two features.

3.3 Classification

3.3.1 K nearest neighbor

As a base line of classification the KNN algorithm is used due to its simplicity. KNN stands for K Nearest Neighbor. Here is K the variable that determents the number of closest data-points in the whole dataset to the data-point that is to be classified. The labels of K data-points are known and at classification the majority of the same labels are taken as the winner. If there is a Tie, it is broken by taking a random winner. The winning label is then assigned to the to be classified data-point since most of its nearest neighbors have this same label.

```
Data: k,x,data,labels
Result: label
initialization;
1. Calculate the distances from x to each data-point in data
2. Sort the labels of the data-points ascending by distance to x
3. Take the mode of the top k labels from the sorted set
```

Algorithm 1: The KNN algorithm

The distance measure impacts the KNN algorithm. Therefore multiple distance measures were used. The Mahalanobis measure uses the inverse of the covariance matrix of the dataset to determine the distance.

$$D_{Mahalanobis}(\mathbf{x},\mathbf{y}) = \sqrt{(\mathbf{x}-\mathbf{y})'\Sigma^{-1}(\mathbf{x}-\mathbf{y})} \tag{7}$$

Here Σ is the covariance matrix with: $\Sigma_{x^1x^2} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i^1 - \mu_{x^1})(x_i^2 - \mu_{x^2})$

And μ_{x^i} is written as: $\mu_{x^j} = \frac{1}{N-1}\sum_{i=1}^{N} x_i^j$

$$D_{Cosine}(\mathbf{x},\mathbf{y}) = 1 - \frac{\mathbf{x}\mathbf{y}'}{\sqrt{(xx')(yy')}} \tag{8}$$

The cosine distance measure calculates the angle between the two data-points treated as vectors.

$$D_{Minkowski}(\mathbf{x},\mathbf{y}) = \sqrt[p]{\sum_{i=1}^{n}(\mathbf{x}_i - \mathbf{y}_i)^p} \tag{9}$$

The Minkowski distance measure is a generalization of the Euclidean (for $p = 2$) and the Cityblock (for $p = 1$) distance measures.

3.3.2 Learning Vector Quantization

Next to KNN multiple variants of Learning vector quantization was implemented. LVQ is an method that trains prototypes to represent a class or cluster. Then for classification a data-point is classified by determining the closest prototype. The training face of the LVQ algorithm as proposed by Kohonen is determines the closest prototype to a labeled data-point from the dataset. if the label of the data-point and the label of the prototype are the same, this prototype is moved towards the data-point in the feature space by a certain step-size. If the labels differ, the prototype is moved away from the data-point by the same amount. This is done until the amount of movement of all prototypes converges to zero.

```
**Data:** prototypes, data,labels
**Result:** prototypes
initialization;
1. Calculate the distances from the prototypes to each data-point in data
2. Sort the labels of the data-points ascending by distance to x
3. Take the mode of the top k labels from the sorted set
```

Algorithm 2: The training of the LVQ algorithm

Multiple variants have been made on the LVQ algorithm.

The used LVQ methods were: Generalized LVQ, Relevance Generalized LVQ, Generalized Metric LVQ, and Local Generalized Metric LVQ.

Generalized LVQ is the LVQ algorithm as proven to converge by (REFEREFER) ref

Relevance LVQ uses a vector λ to keep track and improve on the feature relevances at each iteration. The relevances are then used as weights on the features to make more relevant features more prominent. λ is determined during training as follows:

$$\lambda_m = \lambda_m - \epsilon(\mathbf{x}^i_m - \mathbf{w}^j_m)^2 \text{ ,if } (i = j) \tag{10}$$

$$\lambda_m = \lambda_m + \epsilon(\mathbf{x}^i_m - \mathbf{w}^j_m)^2 \text{ ,if } (i \neq j) \tag{11}$$

Here ϵ is the learning rate, x the data-point, w the prototype, i and j are labels. The relevance vector is then used to influence the distance measure:

$$D(\mathbf{x}, \mathbf{y}) = |\lambda \mathbf{x}_i - \lambda \mathbf{y}_j| \tag{12}$$

Metric LVQ uses a matrix to keep track of the relevances and the combinations between the relevances.

Local generalized metric LVQ uses a matrix per prototype.

4 Experiment

This section will show by example how the segmented dataset is validated.

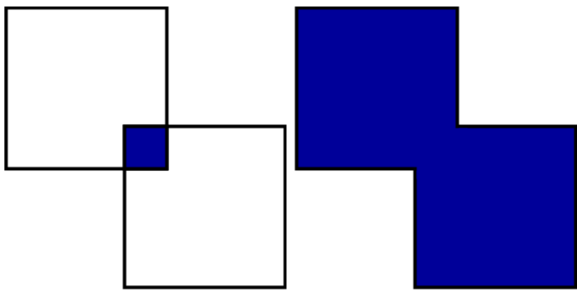
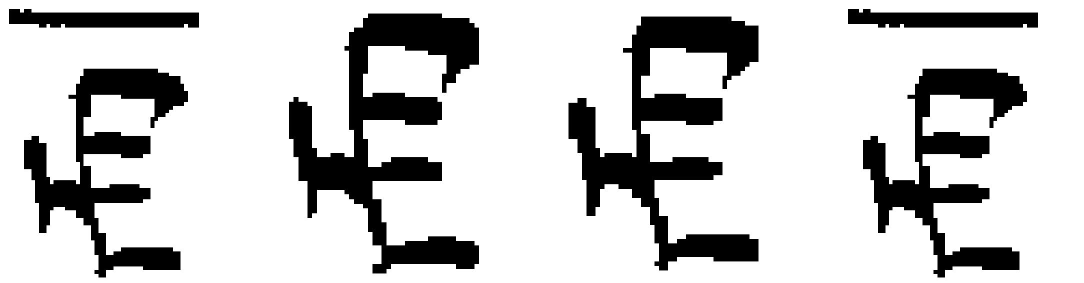

(a) Intersected area **(b)** Area of the union

Figure 9: Method of validating the segmented data (intersection over union)

In order to validate the segmented images obtained by the segmentation the intersection over union is used. As shown in figure 9a the intersection between two boxes is the area that overlaps between these boxes. The union over two boxes give the area both the boxes occupy (figure 9b).

The fraction between the intersection and the union tells us how well the boxes fit on top of each other. With this fraction the performance of the segmentation can be measured.

Because the characters consist mainly of black pixels, and the location of the box is arbitrary, the intersection over union is done only over the black pixels in both boxes. This is done by binerizing the original image and taking its complement. The complement is taken so black pixels will be represented by ones and white pixels by zeros. Now the labeled box and a segmented box are both printed on a white background the size of the original image. This results in two binary images on which an And is the same as the intersection, an Or is the same as a union. Summing the result from these operations give us the needed scalers to perform the intersection over union fraction.



(a) Label image (b) Segmented image (c) Label and Segmented (d) Label or Segmented

Figure 10: Example of intersection over union on a segmented character in the dataset, $IoU \approx 0.73$. The character used is the twelfth character in the image shown in figure 1.

In order to be able to say anything about the performance of the segmentation results will be thresholded from an intersection over union rate from 0.1 to 10. if the intersection over union is lower than 0.1 the boxes are not touching each other enough.

4.1 Feature Extraction

The feature extraction for the experiment was done in two variants: one dataset was generated with the Sobel filters and the other was generated with the use of the Prewit filters. This means that feature 21 (length) and 22 (width) are the same for both datasets. This was done one the segmented dataset created by the segmentation system and on the received labeled dataset.

4.2 Classification

Classification was done by taking the distance measure's described in the method section and running the KNN algorithm with them. For the Minkowski measure, values from $p = 1$ to $p = 6$ were used. This experiment was run on the dataset normalized and non-normalized.

ref

For the KNN algorithm $K = 1$ to $K = 20$ was used. Also multiple runs were done for $K = 1$ to $K = 100$ with the euclidean distance in order to see if a 'large' K has an unexpected influence on the classification results.

In order to gain the classification rates, the 'leave one out' strategy was used. This strategy takes one data-point from the dataset and tries to classify it on the rest of the dataset. This is done for every point in the dataset. It is the extreme case of n-fold data splitting where the maximum number of splits is used.

5 Results

This section will show the direct results of the intersection over union from the segmentation system. It will also show the results from the classification with Knn and the feature extraction method.

5.1 Segmentation

Table 2: IoU Results

0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
0.988	0.973	0.948	0.905	0.841	0.769	0.704	0.631	0.532	0.117

The results from the segmented characters with the labeled data from the dataset.

5.2 Classification

The best result is made bold for all experiments ran.

Table 3: Knn base-set, Sobel

K	Cityblock	Euclidean	Euclidean(norm)	Mahalanobis	Cosine
1	0.6887	0.6689	0.7219	**0.7294**	0.7234
2	0.5900	0.5660	0.6308	0.6517	0.6289
3	0.5517	0.5249	0.5984	0.6173	0.5964
4	0.5217	0.4953	0.5698	0.5926	0.5674
5	0.5018	0.4736	0.5551	0.5822	0.5528
6	0.4958	0.4635	0.5450	0.5754	0.5439
7	0.4943	0.4598	0.5445	0.5761	0.5420
8	0.4927	0.4567	0.5440	0.5734	0.5405
9	0.4933	0.4552	0.5420	0.5730	0.5408
10	0.4914	0.4525	0.5387	0.5726	0.5405
11	0.4871	0.4514	0.5365	0.5701	0.5416
12	0.4857	0.4486	0.5367	0.5680	0.5381
13	0.4856	0.4486	0.5354	0.5674	0.5377
14	0.4829	0.4473	0.5334	0.5658	0.5361
15	0.4824	0.4455	0.5311	0.5639	0.5335
16	0.4835	0.4433	0.5295	0.5618	0.5335
17	0.4823	0.4421	0.5294	0.5608	0.5326
18	0.4815	0.4412	0.5269	0.5586	0.5321
19	0.4811	0.4407	0.5237	0.5568	0.5314
20	0.4794	0.4400	0.5219	0.5563	0.5308

The classification rates of the feature extracted characters from the labeled base-set. Feature extraction was done with the Sobel filter and Knn with K from 1 to 20.

Table 4: Knn base-set, Prewit

K	Cityblock	Euclidean	Euclidean(norm)	Mahalanobis	Cosine
1	0.6509	0.7172	0.7219	**0.7432**	0.6241
2	0.5410	0.6242	0.6308	0.6647	0.5057
3	0.4996	0.5964	0.5984	0.6349	0.4616
4	0.4649	0.5685	0.5698	0.6146	0.4190
5	0.4458	0.5534	0.5551	0.6037	0.3906
6	0.4362	0.5470	0.5450	0.5978	0.3748
7	0.4297	0.5447	0.5445	0.5955	0.3698
8	0.4265	0.5425	0.5440	0.5904	0.3674
9	0.4264	0.5413	0.5420	0.5896	0.3656
10	0.4260	0.5389	0.5387	0.5902	0.3646
11	0.4255	0.5375	0.5365	0.5886	0.3600
12	0.4247	0.5373	0.5367	0.5852	0.3599
13	0.4252	0.5372	0.5354	0.5812	0.3593
14	0.4257	0.5348	0.5334	0.5816	0.3598
15	0.4259	0.5326	0.5311	0.5807	0.3587
16	0.4257	0.5293	0.5295	0.5797	0.3594
17	0.4253	0.5262	0.5294	0.5771	0.3583
18	0.4282	0.5252	0.5269	0.5761	0.3580
19	0.4274	0.5235	0.5237	0.5741	0.3561
20	0.4272	0.5219	0.5219	0.5719	0.3561

The classification rates of the feature extracted characters from the labeled base-set. Feature extraction was done with the Prewit filter and Knn with K from 1 to 20.

Table 5: Knn segmented-set, Sobel

K	Cityblock	Euclidean	Euclidean(norm)	Mahalanobis	Cosine
1	0.8054	**0.8305**	0.8275	0.8179	0.7940
2	0.6865	0.7267	0.7252	0.7149	0.6676
3	0.6274	0.6789	0.6720	0.6621	0.6091
4	0.5911	0.6438	0.6384	0.6322	0.5732
5	0.5748	0.6348	0.6308	0.6159	0.5557
6	0.5672	0.6278	0.6271	0.6106	0.5503
7	0.5693	0.6295	0.6267	0.6093	0.5456
8	0.5666	0.6312	0.6269	0.6067	0.5449
9	0.5665	0.6313	0.6245	0.6056	0.5438
10	0.5671	0.6309	0.6235	0.6018	0.5429
11	0.5644	0.6275	0.6204	0.5991	0.5441
12	0.5601	0.6224	0.6193	0.5964	0.5403
13	0.5601	0.6199	0.6145	0.5960	0.5390
14	0.5594	0.6165	0.6127	0.5960	0.5374
15	0.5593	0.6164	0.6090	0.5929	0.5390
16	0.5580	0.6118	0.6061	0.5929	0.5363
17	0.5579	0.6086	0.6034	0.5908	0.5358
18	0.5557	0.6059	0.6012	0.5897	0.5351
19	0.5545	0.6053	0.5978	0.5880	0.5344
20	0.5552	0.6043	0.5977	0.5861	0.5312

The classification rates with the segmented and feature extracted characters. Feature extraction was done with the Sobel filter and Knn with K from 1 to 20.

Table 6: Knn segmented-set, Prewit

K	**Cityblock**	**Euclidean**	**Euclidean(norm)**	**Mahalanobis**	**Cosine**
1	0.7701	0.7429	0.8229	**0.8254**	0.7456
2	0.6281	0.5903	0.7129	0.7234	0.5925
3	0.5659	0.5149	0.6626	0.6714	0.5223
4	0.5209	0.4623	0.6303	0.6442	0.4702
5	0.4996	0.4319	0.6178	0.6343	0.4396
6	0.4893	0.4223	0.6076	0.6281	0.4257
7	0.4877	0.4167	0.6107	0.6281	0.4222
8	0.4857	0.4106	0.6089	0.6249	0.4146
9	0.4846	0.4066	0.6060	0.6234	0.4126
10	0.4823	0.4062	0.6009	0.6204	0.4137
11	0.4848	0.4028	0.5993	0.6202	0.4140
12	0.4846	0.4022	0.5974	0.6178	0.4121
13	0.4839	0.4023	0.5961	0.6162	0.4107
14	0.4826	0.4002	0.5941	0.6128	0.4100
15	0.4815	0.3991	0.5923	0.6121	0.4077
16	0.4819	0.3964	0.5906	0.6096	0.4076
17	0.4789	0.3947	0.5875	0.6100	0.4067
18	0.4792	0.3943	0.5855	0.6078	0.4068
19	0.4799	0.3935	0.5829	0.6067	0.4067
20	0.4800	0.3916	0.5810	0.6047	0.4069

The classification rates with the segmented and feature extracted characters. Feature extraction was done with the Prewit filter and Knn with K from 1 to 20.

5.3 LVQ

6 Discussion

7 Conclusion

References

[1] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[2] Abraham Bookstein. Pattern classification and scene analysis. richard o. duda , peter e. hart. *The Library Quarterly*, 44(3):258–259, 1974.

[3] Chongmu Chen, Da-Han Wang, and Hanzi Wang. *Scene Character and Text Recognition: The State-of-the-Art*, pages 310–320. Springer International Publishing, Cham, 2015.

[4] Jiun-Lin Chen, Chi-Hong Wu, and Hsi-Jian Lee. *Chinese Handwritten Character Segmentation in Form Documents*, pages 348–362. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[5] Shan Du, Mahmoud Ibrahim, Mohamed S. Shehata, and Wael M. Badawy. Automatic license plate recognition (alpr): A state-of-the-art review. *IEEE Trans. Circuits Syst. Video Techn.*, 23(2):311–325, 2013.

[6] M. Y. Fu and Y. S. Huang. A survey of traffic sign recognition. In *2010 International Conference on Wavelet Analysis and Pattern Recognition*, pages 119–124, July 2010.

[7] J. M. Guo and Y. F. Liu. License plate localization and character segmentation with feedback self-learning and hybrid binarization techniques. *IEEE Transactions on Vehicular Technology*, 57(3):1417–1424, May 2008.

[8] S. Jaeger, C.-L. Liu, and M. Nakagawa. The state of the art in japanese online handwriting recognition compared to techniques in western handwriting recognition. *Document Analysis and Recognition*, 6(2):75–88, Oct 2003.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Intelligent Signal Processing*, pages 306–351. IEEE Press, 2001.

[10] Nikos Nikolaou, Michael Makridis, Basilis Gatos, Nikolaos Stamatopoulos, and Nikos Papamarkos. Segmentation of historical machine-printed documents using adaptive run length smoothing and skeleton segmentation paths. *Image and Vision Computing*, 28(4):590 – 604, 2010.

[11] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, Jan 1979.

[12] Robert Sedgewick. *Algorithms in C*. Addison-Wesley, 3rd edition, 1998.

[13] Irwin Sobel. History and definition of the so-called "sobel operator". 2014.