Hierarchical Font Recognition

Letter Snippets — Visual Words in Font Recognition

Diploma Thesis

Thesis advisor Prof. Dr. Eyke Hüllermeier
Department of Knowledge Engineering & Bioinformatics
Philipps University of Marburg

Germany

written by
Jakub Tomasz Lidke

May 19, 2010

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

$Marburg, Mai 2010$

Jakub Tomasz Lidke

Contents

Chapter 1

Introduction

1.1 A Thousand Fonts are not enough

One of the most important reasons for font design can be found in theory of media. Font plays an important but subtle role, while communication takes place between receiver and sender. According to Uses and Gratifications Theory [7], the way information is understood, depends, among other things, on style of the message, receivers expectations and certain surrounding circumstances. A thin fluent script will introduce other prospects about a message than a square Gothic font. This will affect subconsciously the way a message is understood. Knowing this, it is not surprising that today there exist over a hundred thousand different fonts. While books and newspapers often use less than 10 different fonts, magazines and advertisements offer a wide range of different typefaces. Movies, serials, games and companies make use of uniquely designed fonts, which will be instantly associated with them. Because of the huge number of available fonts, today optical font recognition (*OFR*) is more important than ever before.

In the following we will introduce several applications for OFR: First, a font designer wants to create new fonts. Therefore, there is a need for being able to verify differences between prototypes and other fonts. A font search engine seams to be an essential tool, since manual comparison is too time consuming. Second, detection of font license violations becomes applicable in large scale, if a font search is available. On the other hand, it would be possible to learn whether there is a free font available, which is very similar to an expensive commercial type. Furthermore we can think of applying font recognition as a module in existing software. There are projects where printed media (newspapers & magazines) are archived. The content must be fully searchable. Among other things correct text block alignment and good optical character recognition (*OCR*) are needed. In both cases font

Figure 1.1: An overview of most important font features.

recognition is a great help. For example, we know that companies use unique fonts, therefore advertisement can be recognized by OFR. In magazines and books different fonts also indicate different content. Again, OFR offers a convenient way to increase the performance of text block alignment. OCR applications usually apply multi-font recognition software, which is in most cases a favorable choice. Recognition errors occur if unknown or exotic fonts are used or image quality is poor. If the font is known before OCR is applied, the exact shape of each letter is known, therefore character recognition may achieve higher precision.

The main contribution of this diploma thesis is the introduction of a new OFR method and its evaluation on a huge database of 9809 fonts, which exceeds to the best of our knowledge the size of the largest font database, mentioned in related works by more than 3.5 times. Further, we explore importance of input length and content with respect to recognition rate.

Based on part of our results, we have also written and submitted a paper to the International Conference on Pattern Recognition (*ICPR*) 2010 which has been accepted.

1.1.1 Anatomy of Fonts

This section will introduce the structure of fonts and how they can be distinguished. Technically there are over 37 different features, which conjuncted may identify most fonts. We will briefly explain a small selection of features, which will enclose the problem of font recognition.

Apart from the font size, the *font style* can be changed by a word processor. There are 4 standard presets *italic*, **bold**, ***bold-italic*** and normal. We do not count variations of these as new fonts because they are created by a word processor, which is a form of post processing. Nevertheless a font can be designed e.g. italic or bold, the important thing is which is the designated appearance.

Figure 1.1 shows some of the most important features, which are used to distinguish fonts manually. The area in which a character is placed is separated by four lines. The lower height *(a)*, the baseline *(b)*, the x-hight line *(c)* and the upper height *(d)*. Obvious font features are the *x-height* (10), the *ascender* (11) and the *descender* (9). The relation between *capital-height* (12) and *x-height* is often a good attribute to create general groups of fonts. One of the best known features are *serifs* (1). If they exist in a font, serifs can be oriented to only one side or both. Further their shape can differ in many ways. The angle in which horizontal strokes hit the baseline is important too (3). The thickness of the *stem* (4) is also a good attribute for font recognition. The contrast (2) is a very important attribute of a font. The contrast is defined as the relation between the thin stroke (*hair line*) and the thickest line stroke in a font (more examples in Sub-Figure 1.2a). Usually the character *H*, is a good example to measure the contrast of a font. But it is not always constant. There are fonts which change the contrast dynamically. Garamond is such a font. Depending on the applied *weight*, the contrast differs. Here *weight* expresses the mean thickness of a font. If we would use a quill to paint a letter, all attributes of it will be needed to describe the first part of a stroke. The angle, the pressure, the shape and the thickness of the used quill will determine the shape of the *starting stroke* (6). Although it is a very small part of the shape, it is a very discriminative feature. Of course the finish of a line is similar to and also a powerful attribute (7). A very specific but deciding feature is the orientation of the diagonal of the letter *O* (8). Very similar to the *O*-feature is the eye in the character *e* (see Sub Figure 1.2b). The eye is the shape of enclosed area by the character *e*. Another attribute is given by the shape and size of the *bowl* (5) in a letter. Bowls are closed areas in letters. Another related feature is the angle and the shape of the *stroke entry* and *stroke exit* of a bow. This is a feature, which is used to distinguish very similar fonts. As mentioned before, there are many more features which can be used. But they are mostly very specific and hard to recognize.

To summarize this section, we want to emphasize two observances. First, most of the listed features are very difficult to extract with respect to the topic of pattern recognition. The most difficult point is, that we need extensive knowledge about the data and its shape before we can even extract a desired feature. This makes many possibly powerful features useless for automatic font recognition or narrows their applications.

Second, there are two kinds of features. There are features which describe relative differences in size, thickness and angle of letters. These features can be extracted from all letters from a font in general. In contrast to those we can find many features, which cannot be extracted from each character. In particular they can only be captured if we examine certain regions of distinct letters.

(a) Contrast examples

(b) The eye of *e*

(c) start stroke 1, end stroke, start stroke 2, bowl, serif shape

Figure 1.2: Figures show examples of one global and five local features. Images from [4]

We can derive two possible approaches. At First, we can introduce methods, which make use of general attributes – *global methods*. Second, we can distinguish fonts by comparison of detailed characteristics – *local methods*. In the following Chapter 2 we will introduce works on the subject of font recognition which can be assigned to local or global approaches.

Chapter 2

Related Work

This chapter will give an overview to several methods, which were applied to the challenging topic of font recognition.

It is important to note that the notation *font* can be misleading. Most approaches are applied to very small numbers of fonts. Usually 5-20 different fonts are used, but often much higher numbers are reported. The reason for that is the way *font* is defined. Often the number of fonts is the product from font style (**bold**, *italic*,...) typeface (Arial, Comic,...) and font size (10pts, 12pts, ...).

First, shapes can be described from a distant or close point of view. By choosing the view, we also choose the way we will describe a shape. Usually descriptive attributes of distant objects will refer to global properties. While details and local parts of a shape will be the subject from a close perspective. If we choose the most descriptive attributes to specify objects, we should be able to distinguish several objects by referring only to these attributes. The process of collecting these properties is called *feature extraction*.

Generally there are two different paradigms, we can do local or global feature extraction. In this case local refers to information gained from parts of single letters. Hence global features refer to information, which are extracted from entire lines, pages and words. The advantage of local features is their flexibility in real-world applications. A well working local feature extraction relies in most cases on a reliable character segmentation. Character segmentation is used to extract single letters from word images. Global approaches are more forgiving with respect to noise and segmentation errors. Usually more data for training and recognition is required than in local approaches. However feature classification is usually done by well known classifiers like Neural Networks, SVMs, Bayes classifiers or appropriate distance metrics.

By using topological and morphological features, which were extracted from text lines,

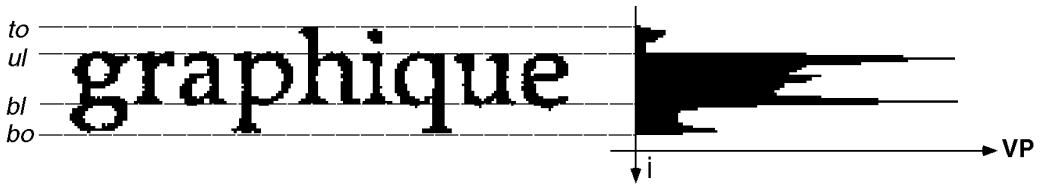


Figure 2.1: Typographical lines from a vertical projection profile.

Zramdini and Ingold [25] realized a statistical method, which allowed to identify typeface, weight, slope and size from unknown fonts. Since entire text lines are processed, it is a global feature approach. Applied features are vertical and horizontal pixel projections (see Figure 2.1 which shows an example from this paper), which were used to create a font model base. Each model has been created from the average of over 100 text lines of each font (font size, typeface and style), which resolved into 2800 text lines for training. They have also done experiments with different line size for test data, but with about $\frac{1}{4}$ of the original dataset. However recognition rate drops to 64% at a text length of 25 words and improved from 96% to 99% with 400 words per input line. The disadvantage of this approach is dependency on content and length of input lines. Further application of this method is severe narrowed because of its requirement for large data sets.

In method presented by Schreyer [1] fonts are described by texture elements called Textons. Textons are based on the assumption that every texture can be reconstructed by a combination of basic elements. For more detailed information we refer to [19]. Basic geometric features and thresholds were used in order to detect font style (bold, italic, normal) within four different fonts and three font sizes. However, the main contribution of this publication is the Texton structure itself.

Lee and Jung [12] presents a Non Negative Matrix Factorization (NMF) for feature extraction. Letter images are segmented and rescaled to a small fixed resolution. For each font a profile representation is created, which is generated with an agglomerative hierarchical clustering algorithm. Distances are assigned by the Earth Mover Distance, while Tanimoto Distance is used as ground distance. Recognition is done on six fonts, four styles and on lower as well as on upper case letters. Letter segmentation and parameter settings are very important to this approach, and have to be manually adjusted for each dataset. Reached recognition rates are favorable.

Sun [20] applies stroke templates which are generated from characters of a font and stored in a database. Given input stroke data a Bayes classifier decides about the most likely font. Classification is done with 5 English and 5 Chinese fonts, which can be found

Figure 2.2: First five eigenimages from character a. Intensity values are normalized.

in four different styles (normal,bold, italic, bold-italic). Classification rates are for certain fonts very good. However, not every font can be processed by the proposed stroke templates which make the feature unpractical for real-world applications.

A different interesting method is introduced by Solli and Lenz [18], where recognized text is used as an additional clue for font recognition. Grey value images of single letters are processed by principal component analysis. The resulting covariance matrix, provide eigenvectors and eigenvalues. These vectors describe a low-dimensional subspace. Original images are projected on this space. Their coordinates in the subspace are used as new descriptors. Referring to the idea of Eigenfaces [22], the descriptors are named Eigenfonts (see Figure 2.2). Recognition is done on a database of 2763 different fonts with a best-in-five selection. While this approach achieves convenient results at a large dataset, it has also the strongest prerequisites – it cannot be used without working OCR and character segmentation. Besides, this approach made us realize that there are no other approaches which can compare with it, since they applied a much larger dataset then others. Which was concurrently the priming to our work.

A Neural Network classifier is applied by Jung [10]. Input nodes are fed with vectors, which a representing topological features. Which are very similar to features described by Zramdini and Ingold [25]. However recognition rates are satisfying at seven fonts with font sizes between 9-18pt.

A global feature approach is introduced by Zhu [24]. This does not require character segmentation. This method uses uniform text blocks for feature extraction, hence spaces between words and text lines will be set to a small fixed value (see Figure 2.3). Font recognition is transfered into a texture recognition problem. Texture features are extracted by a multi-channel (4 orientations and 4 frequencies) Gabor filter from preprocessed text blocks. Final font identification is done by a weighted Euclidean distance classifier. The dataset consists of 6 typefaces and 4 font styles for Chinese fonts and 8 typefaces and 4 styles for English fonts. The average performance of this approach is very convenient. However the need for a large datasets for training as well as for testing, narrows possible applications to real applications.

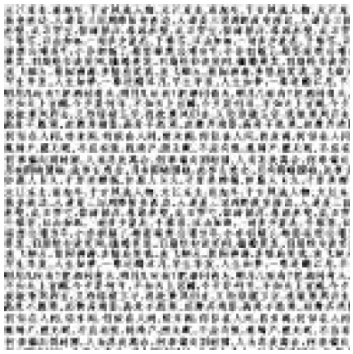


Figure 2.3: A uniform text block for feature extraction.

Doermann [13] apply Gabor filter on dictionary pages to distinguish between two arbitrary script types as Hindi and Roman or Arabic and Roman and their style. Filter settings are the same as in [24], but identification is done by a different distance measurement. In contrast to the other Gabor filter based methods, text blocks were generated from single word copies not from whole text lines. Real world applications of this approaches are limited, since it is designed for dictionaries. However results were satisfying for this application.

Avils-Cruz [6] proposes another global texture analysis method. Before feature extraction uniform text blocks are created. Afterwards high-order statistical moments (3^{rd} and 4^{rd}) are employed for feature extraction. 3^{th} order moments describe the skewness of a distribution, while 4^{rd} describe its abruptness (kurtosis). Then principal component analysis is performed to the extracted features. Additionally, an expectation maximization algorithm provides an unsupervised method to find the number of fonts and their parameters. Finally, a Bayes classifier is used for classification. While this approach shows accurate results on a small dataset of 8 fonts and 4 styles, it has very high computational cost. Thus, for the desired application of Internet font recognition it is not applicable.

Ma [14] presents a biologically inspired method. Here, the Grating operator is used for feature extraction. The Grating operator is similar to a Gabor filter, but with a strong precedence to regular patterns. Further details can be read in [11]. Moreover it is combined with a back-propagation Neural Network to classify between 5 fonts and 3 styles.

We have learned that there are many methods for OFR on small datasets. Since we are not aware of a favorable method for font recognition with respect to large datasets, there is still research to be done. Our method will introduce a new approach in this direction.

Chapter 3

Local Patch based Font Recognition

As we have seen in the previous chapter, there are several approaches to font recognition. Therefore it is important to put our method into context. We want to introduce a new method for Internet font recognition. As we have learned in the related work Section 2, many methods have been applied to small datasets. In contrast not much is known about font recognition on large datasets, hence this topic needs more exploration. Our work is a contribution to these efforts.

The following pages of this chapter will first give an introduction about some general concept ideas. Next we will discuss features, learning methods and an abstract view of our approach. Subsequent sections will introduce our method in detail. In Section 3.3 we describe the preprocessing phase. Applied features will be introduced in Section 3.4 and classification of fonts will be explained in Section 3.5.

3.1 Overview

This section is organized as follows: First we will discuss possible features, which are locally or globally oriented (Section 3.1.1). Second we will give a brief overview about possible learning systems (Section 3.1.2).

3.1.1 Type of features

Next we describe feature types with respect to their practical applications and their realization. As described in the related work section there are two paradigms from which we can choose:

Global features
Here we assume that single letters do not carry enough information about any font for sufficient recognition. Given that we choose enough normal distributed letters from an arbitrary font, a unique but font specific pattern will emerge. That way font recognition is transformed to a texture recognition problem. In order to create suitable input data, many uniform text blocks (illustration 2.3) have to be created. Before text blocks can be created, word and line segmentation has to be performed. While text lines have usually no skew, this is a very fast and easy operation. Further preprocessing is usually not needed, which is the biggest advantage of global feature approaches. These feature extraction methods cannot be applied in most applications, because of their need for high numbers of test and training data samples. Often there is only a small number of data samples available.

Local features
They can target parts of characters up to word parts. Usually global features move font recognition towards texture recognition; local features can be designed to target typographical features, texture statistics, shape matching or combinations of these. Clearly they are much more flexible than global methods, also there are more applications available for this kind of method. According to the "No-Free-Lunch-Theorem" there is a price to pay for this flexibility. Local approaches usually require a reliable character segmentation, which in some cases is a challenging task.

We assume local features will perform better than global ones with respect to large numbers of fonts. This is because many very similar fonts can be found in large datasets, where global features may be too coarse to be able to distinguish them. Further local features are not as restrictive as global features with respect to application spectrum. Furthermore local features have shown very good overall performance in the field of Computer Vision which is an additional motive for applying local features.

3.1.2 Learning systems

There are three different learning systems available. The following paragraphs will give a brief introduction on these. After that we will explain our choice.

Unsupervised learning methods can be understood as closed systems, which posses a limited number of tools to process input data and put it into relation to each other. These methods create an order or another perspective to the data. Most important is the fact that the system is neither aware of any environment information nor does it receive any

information about the meaning or real relation between data itself or its relation to any environment. Based on the system tools, data will be processed and results will be returned. A well known exponent of these systems is the PCA.

Reinforcement learning systems are interacting with their environment, which generates negative or positive feedback to the general performance. The system creates adjusting actions to the environment, which aim for maximization of the positive feedback. Hence the environment is an abstract collection of one or several systems. This method is generally a good choice in order to find optimal parameters in a working process. Especially if there are many parameters and the complexity of the environment is too high to find a decision manually. A full evaluation of data can be very time-consuming, because this kind of system usually has to run several cycles until a target performance is reached. Since we want to develop a method which works with a large dataset, a reinforcement learning system could be too slow to start with.

Supervised learning systems adapt models, which are modified by pairs of data — input and its desired output. A good model is able to reproduce the corresponding output, when a previously shown or similar input example is provided.

In many recognition tasks supervised learning approaches are applied. For two reasons this seams to be the best system to work with. First we can make use of all information we can extract. Second we get a direct feedback about chosen methods, which will make further evaluation of each decision in the process easier. Since we have decided in favor of a local feature approach, there is a difficult task to solve. How do we extract most of the local features (Section 1.1.1)? Several specific feature extractors would be very slow and therefore not applicable. We need a flexible extractor, which can capture most of the mentioned features. Thus we cannot be sure, what feature will be captured exactly. Also it will be difficult to assign consistent labels to data. The reason for that is the similarity of some local shape parts. For instance horizontal or vertical line strokes are the same among many fonts. Consistent examples are essential for a successful classification in a supervised environment. If we want to proceed with this idea, we need a different learning system. We need a algorithm, which is able to organize extracted features with respect to their content. This task can be accomplished by an unsupervised approach. Therefore we have chosen the *Bag of Visual Words* method. The next section will introduce its concept.

3.2 Bag-of-Words

This section will introduce the concept of *visual vocabulary*, which enables us to search for content using indexed local features. The idea is taken from text retrieval, where it has been successfully applied. First we describe how the visual vocabulary is created, second we describe how *visual words* are used and finally we will define several characteristics of this approach.

3.2.1 Creating a Visual Dictionary

In the topic of text retrieval documents can be identified by collecting a number of words, which are used as a summary from a given text. Words statistics of documents can be used to create unique fingerprints. A selection of words from these fingerprints, is also a very good summary of the original document. By selecting a few key words from a summary, documents can be found very quickly.

First there seams to be a problem to unify the world of words with the 2-D world of images. How do we translate high dimensional images into single words? Words are very limited in their appearance. In contrast images can contain thousands of different shapes and colors. A common way to discretize real-values is to create value windows, which will contain values between discrete borders. This is also applicable in the case of images. In terms of 2-D images we need a local feature extraction. A collection of image feature descriptors constitutes a feasible discretization of the image space. But we also need a discretization of the feature space, therefore we need a fixed number of prototypes, where features can be assigned to. Data prototypes can be created by application of a clustering algorithm. The entire set of constructed prototypes is the visual vocabulary/dictionary or also called codebook, which gives is a quantized feature space. From now on it will be used to translate feature vectors into visual words.

In practice we will create a representative dataset. We will apply feature extraction to it and afterwards we will quantize the features. The resulting codebook will be the basis of further steps. Hence first dataset may be discarded.

The next step is to translate each entry of the real dataset. Therefore each feature value is translated using the dictionary into a *visual word*. Each word is represented by its unique index. Usually we will extract a collection of features from one entity, hence we will also receive a collection of word indexes.

The list of *visual words* per data example is analyzed. Frequencies of each word are counted and assembled to a *bag-of-words* histogram. These histograms summarize the

data, while the dimensionality of it is much smaller than the original data dimensionality, several examples of our visual words are illustrated in Figure 3.1.

Again if we assume an image to be a document, then we can clearly see the analogy to the previous text retrieval idea. We summarize text documents with a collection of key words. So it can be recognized using only these words.

The document recognition is performed very similar to the former creating step of the vocabulary. New examples are translated using the previously generated vocabulary. Once more we create a *bag-of-words* histogram for the new data. Finally we have to compare the frequencies of each word with the occurrences in the former phase. The first practical approach of this idea was implemented by Zisserman and Sivic [17], who indexed video frames in order to search for particular objects.

3.2.2 Important factors at Bag-of-Words

Since we have decided to apply a bag-of-words approach, we should be aware of influential factors to this method. There are several factors, which are important to this approach and all have a strong impact on the outcome.

Interest points define positions for feature extraction. It is an important and difficult task to find these points. Often optimal coordinates are unknown. Salient regions in images are not necessarily good extraction regions. Also regions, which appear to be important from the semantic point of view, are not always appropriate. Hence, if regions are chosen badly from the beginning, the approach will not yield convenient results.

Point density raises two problems: First, should there be overlapping between regions? This is important because overlapping can be a double-edged sword. On the one hand we get a very detailed scanning of the data. On the other hand overlapping information can interfere with each other, which may result in noise and false data. Second, how many regions need to be extracted? The more features are extracted the more precise a recognition can be done, concurrently this will raise the time complexity of the approach.

The Feature itself is the most important factor as in any other approach. Features have to describe the data well enough to allow a differentiation to some extend, especially if we compare smaller regions. Regions assigned to the same visual word, should also be related to each other. Besides, best possible parameter settings, it is always a good choice if the applied feature is scale and rotation invariant.

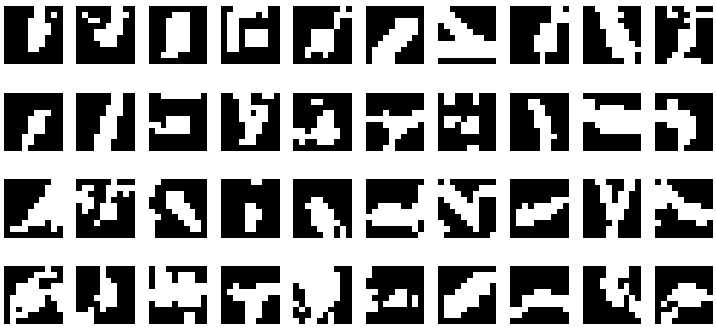


Figure 3.1: Examples of visual words.

Loss of geometric relations is a fact which is bonded to the bag-of-words idea. While we disassemble an image into small regions, overlapping or not, we do not use any information about their positions. Furthermore data is arranged in a unsupervised manner, which also ignores any order, which may have been given by the order of the data samples. Depending on the importance of geometric relation, this may be compensated by a post processing step.

The visual vocabulary is the second most important factor to a bag-of-words method. Hence, the dataset used to create the vocabulary is very important too. First of all it has to be representative for the entire feature space. Second, number of visual words must be high enough to translate features in a appropriate way. At the same time it must not contain too many words. Since the *Curse of Dimensionality* lies in the number of visual words. If number of visual words is too high, single examples will be represented by sparsely assigned visual words. This will lead to comparison difficulties, where bags-of-words cannot be distinguished, since they are all equally different. A good vocabulary can be used for different datasets, with convenient results. However best results are achieved, if test and training data samples are used to create the vocabulary.

To summarize the previous sections, we will emphasize the most important facts. We want to expand knowledge in the topic of font recognition on large datasets. For the sake of flexibility we have decided to use a local feature extraction on single letters. Because of problems with inconsistent data samples, standard approaches using supervised learning seam to be difficult to apply. In this case an unsupervised learning system appears to be the best choice. We have decided to apply a Bag-of-Visual-Words approach.

(a) Newspaper color scan

(b) Newspaper binarized

Figure 3.2: Here we can see an example for document binarization. The method applied was proposed by S. Lu and C.L. Tan of the Institute for Infocomm Research in Singapore. They won the Binarization Contest arranged by ICDAR 2009.

3.3 Preprocessing

Preprocessing is needed to prepare data before feature extraction can be successfully applied. This section will explain adopted preprocessing methods in detail.

Preprocessing is built up as follows:

1. Image Binarization
2. Character segmentation
3. Image sampling

3.3.1 Image binarization

Binarization maps pixel values from an arbitrary color space to 0 and 1. Usually it is applied as the first step to each image. On the one hand image contrast will be maximized this way, which is most important for following operations. On the other hand computational time and memory usage are lower that way. Although there are plenty of methods for image binarization available, for the sake of simplicity we apply simple threshold binarization.

As we work on synthetic images, we do not have to deal with light conditions, shadows or bad image quality (see Figure 3.2). Experiments showed that simple threshold based binarization performs well in our case, therefore we did not apply complex methods. Pixels which have gray values above 124 are considered full black, while the others are considered white. Finally images are inverted, which is needed for further processing.

(a) Horizontal projection (b) Vertical projection

Figure 3.3: Here are two examples of horizontal (a) and vertical (b) pixel projection, which can be used for word and line segmentation.

3.3.2 Character segmentation

Character segmentation is usually the second step in most font recognition approaches. Usually text is arranged in blocks. Blocks can be separated into text lines, which again can be separated into words and finally they can be split into single letters. This split operation is done by character segmentation. It is to a certain extent necessary. If we would apply a global feature method, we would probably need a line and word segmentation. Usually local approaches need character segmentation. Spacing between single letters are often very small. Hence local feature extraction could consider values from other letters in direct proximity to a target letter position. This additional context would pollute the feature extraction, which would complicate the recognition process unnecessarily.

Most fonts have a distinct space between letters, strict vertical fonts are not a challenge because we can create vertical and horizontal pixel projections, which will indicate letter or line spacing with zero values (see Figure 3.3). However italic fonts are more challenging. They can be left or right slanted with different angles. Thus vertical projections will only work for word segmentation, while characters cannot be separated that easily.

Here we will introduce briefly a method to character segmentation: *Connected Component Labeling* scans input images and investigates the neighborhood of each pixel. Pixels which are directly connected, become grouped to a certain label. The simplest version of this algorithm will return groups, assuming each group is a single letter. In addition, heuristics and thresholds can be used to enhance the algorithm. Although there are many fonts which may be difficult to segment (see Figure 3.4). If appropriate settings have been chosen, even challenging fonts can be segmented. However image segmentation is a well researched field, and a subject for its own.

3.3.3 Image sampling

Image sampling is the third step and most crucial part of the preprocessing phase. This step computes positions for feature extraction. As mentioned in the beginning we apply

Figure 3.4: An example of a difficult font for segmentation, where connected component labeling has been applied.

a local feature approach, which in this case demands knowledge/decision making about regions to choose from. The optimal positions for feature extraction depend strongly on the feature itself and the dataset. There are two strategies to solve this problem. First, we choose a regular sampling of a shape. This way we will cover the entire shape and most important regions will be covered this way. Although we could receive an unnecessarily large number of extraction points. Second, another feasible method is given by an extraction of the shape skeleton, which again results in a regular sampling of the shape, with an additional local constraint, which demands that a given point is located in the middle of a shape. We have applied four different methods for image sampling:

Random pixel selection The most simple and probably worst way to get extraction points is random selection. Random points tend to create small groups or lines within the letter. This often leads to missing representation of image regions. While some shape parts get many representatives other get none.

Information Theoretic Vector Quantization (ITVQ) We have also applied an implementation of ITVQ for binary images. Since this method tries to estimate a statistical model for the shape of an input image, it may result in better results than the k-Means algorithm. The idea is as follows: A number of sampling points can be calculated, by minimizing the distance between a statistical model of the original image and a simplified one, based on the returned points.

We are interested in a good representation of the image by a finite number of points $M = x_1, \ldots, x_m$, where their number is much smaller ($M \ll N$) than in the number of pixels in the input image. The image function $I : X \rightarrow \{0, 255\}$, where X is the set of pixel vectors, can be described by a probability density function *(pdf)* $p_1(x)$. In order

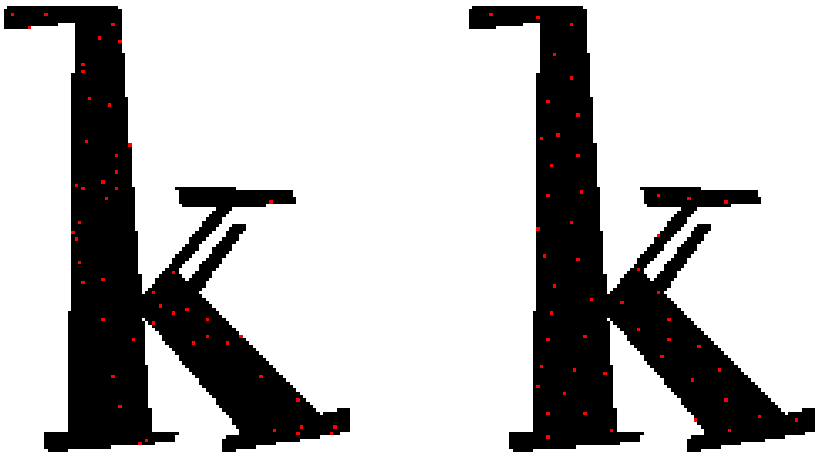
Figure 3.5: An example of the ITVQ sampling method.

to estimate pdf $p_1(x)$, the Parzen estimate is calculated based on the image vectors. A Gaussian is chosen to be the Parzen window. In order to estimate the pdf we create a sum of Gaussians over the data point vectors. Further, we initialize a number of random points, which will shape the quantization we are searching for. Now we can estimate another pdf $p_2(x)$, which describes vectors from M. These two pdfs need to be put into relation, in other words we need a comparison measure. One convenient way to do that, is the Cauchy-Schwarz (CS) pdf divergence (Equation 3.1).

$$D_{CS}(p_1, p_2) = -\log \frac{\int p_1(x)p_2(x)dx}{\sqrt{\int p_1^2(x)dx \int p_2^2(x)dx}} \tag{3.1}$$

Where p_1 dedicate the pdf of the original image, the p_2 is the pdf of the quantization. The minimum is obtained if the pdfs are equal. Further we replace p_1 and p_2 with their Parzen estimates. After solving the integrals from D_{CS} and rearranging the terms, we can find an expression for fixed points, which will be used as a update rule for points in M. Hence the integration constant contains a cross entropy term in the nominator, which works as a counterbalance in the process. It prevents the number of points from collapsing to very small numbers. After several update iterations (we run 150) a convenient result is reached (illustration in Figure 3.5). Though this approach delivers more stable and representative results than random selection or k-Means clustering, we have little control over the number of points, which will be returned. This method is in the same complexity class as k-Means clustering.

k-Means clustering A more reliable method is the well known k-Means clustering [15]. The general idea is to group similar objects and to find one representative for each group. The process of grouping is called clustering. While we do not need to add additional in-



(a) Random (b) K-Means

Figure 3.6: Comparing random selection to k-means clustering of foreground pixel coordinates, while k was set to 50. k-Means shows in general a more regular distribution than random selection.

formation besides the data itself, k-Means belongs to the unsupervised learning methods.

K-Means clustering creates a k partitioning of an arbitrary number of input vectors. The goal is to minimize the objective function 3.2.

$$J = \sum_{i=1}^{k} \sum_{x_j \in S_i} ||x_j^{(i)} - u_i||^2 \tag{3.2}$$

Where $\| x_j^{(i)} - u_i \|^2$ is a chosen distance metric between $x_j \in S_i$ and u_i a centroid position. Thus the variance of vectors represented by an arbitrary partition $S_i \in S$ tends to be low, where S is the set of all portions generated by the algorithm.

Partitions are generated as follows (an illustration can be found in Figure 3.7): First k random samples are generated. These are the first cluster centers. Second, each vector is assigned to the nearest cluster. Third, a new centroid for each group is calculated and saved as group representative. Fourth, the algorithm starts a new iteration from the second step. The algorithm proceeds unless no changes occur or a preset maximum number of iterations is reached. k-Means will converge after some time, but it depends on the amount and complexity of the data. There is no guarantee to find an optimal solution. Mary Inaba [9] has shown, that for a fixed number of clusters and dimensions k-Means Clustering reduces to polynomial time complexity, while in general it is *NP-hard*.

Each centroid can be understood as a prototype of all vectors, which are assigned to it.

If we had new vectors, we could identify their membership to a centroid by computing the distance to all centroids, the center with the smallest distance represents the corresponding partition. This step is also referred to as Vector quantization *(VQ)*.

Hence we obtain sampling points by feeding k-Means algorithm with coordinates of all foreground pixels, which are building up our letters. The resulting codebook contains the coordinates of the centroids, which are later used as extraction points. In our approach k was scaled dynamically with the image size. Nevertheless k-Means clustering is computationally expensive so we limit the iteration number to 5. Most of our experiments have been done with k-Means clustering of image pixels. Point distribution is more regular than random selection, because selected points combine local values of the letter (compare Figures in 3.6).

k-Means clustering can be also understood as a discrete version of the Expectation Maximization algorithm (EM). The basis idea for EM is adopted from k-Means. EM estimates parameters of a statistical model, which describes a data distribution, given a set of examples. While k-Means calculates a discrete model of a dataset, an EM algorithm computes parameter of the maximum likelihood estimate for a statistical model. First, observed data samples are used to calculate data dependent parameters of a probability function, which corresponds to the second step in the k-Means algorithm. The next step finds parameters, which maximize the probability function. This corresponds to the movement of the centroids in the third k-Means step. Alike k-Means, EM algorithm cannot guarantee to find an optimal solution.

Laplacian-Distance-Transform In contrast to the other sampling methods this one tries to extract points on the skeleton or also called medial axis [3]. First we create the distance transform [16] of an input image. The distance transform is a map of the input image, thus it has the same size as the original image. As figure 3.8 indicates, each entry in a distance transform holds the minimal distance to the next white pixel. Note that it can be only computed correctly, if the input image is binary, which is not a drawback in our case, because we need binary images anyway for other steps. Geometrically speaking it builds a height map from a binary image, where peaks mark positions with the largest distance to planes (illustration in Figure 3.10a). Any Distance Transform has very powerful predicates, it is rotation invariant. Horizontal and vertical pixel neighbors differ at most by one at any scale. Of course values will change, if the image is resized, but relative value differences will be conserved, which gives us a kind of scale invariance as well. Hence the relative positions of each maximum from a distance transform are scale invariant, which could be used for skeleton generation. We tried to extract pixel positions at maximal values

from the Distance Transform. But there are either not enough or there are too many of them. Figure 3.9 illustrates this problem while using letter images as input. Examples show that letter shapes vary in stroke thickness too much. The extraction of pixels with highest Distance Transform value does not lead to aspired results. A possible method could be the application of a shape sub-devision method.

This leads to a local consideration of the shape. There is a faster and more purposeful method, than dividing the image into sub regions. We realize that the search for the highest values of the DT is also an edge detection problem. The first derivative from the Distance Transform will reveal extreme values. We calculate it in x-axis and y-axis direction and calculate the sum of both, which is the Gradient of the DT. (Equation 3.3).

$$G\left[f(x,y)\right]) = \begin{pmatrix} G_x \\ G_y \end{pmatrix} = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} \tag{3.3}$$

In digital image processing derivatives can be approximated by differences.

$$f^{'}(x_i, y_i) \approx f(x_{i+1}, y_i) - f(x_{i-1}, y_i) \tag{3.4}$$

The Gradient indicates changes of a vector field and can be used as an edge detector. Hence zero values will reveal positions on the shape skeleton, which is an edge on the DT. Additionally we get points on the border of our shape, which can be ignored. We observed that zero values on diagonal lines from the skeleton, are often overwritten in the summation. Reason for that is the DT itself. Its property of slow value changes between neighboring pixels makes it difficult to localize edges with the first derivative.

Instead it is more precise to apply the second derivative. Figure 3.11 shows this with a 1-D example. The second derivative is more sensitive to noise than the first derivative. This also is a reason, why it is not applied very often in other applications. If an image contains noise, it is important to smooth the image first. One possible way to do that is the application of a Gaussian filter. In our case however there is no need for that, because we work on synthetic images. The 2-D equivalent to the second derivative is the Laplace Operator. We apply the approximation h_1 of the Laplace Operator (see Equation 3.5), which is not rotation invariant. A rotation invariant approximation is shown in Equation 3.6.

$$h_1 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad h_2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.5}$$

$$h_3 = \begin{bmatrix} 2 & -1 & 2 \\ -1 & -4 & -1 \\ 2 & -1 & 2 \end{bmatrix} \tag{3.6}$$

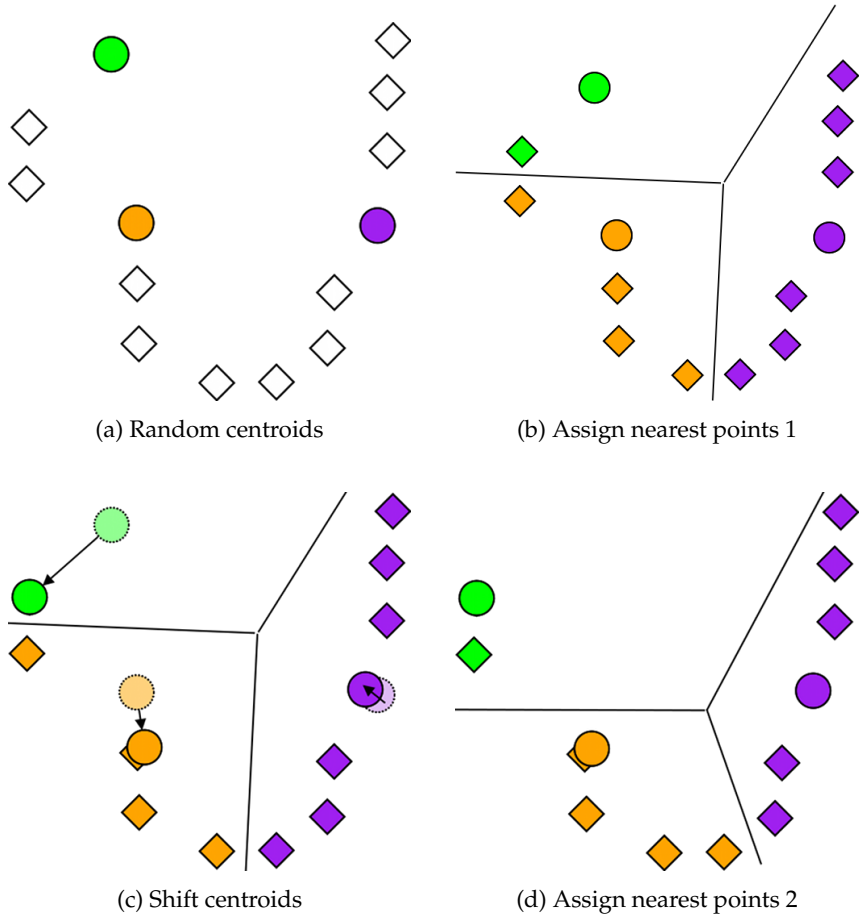
We have compared the experimental results form h_1, h_2, h_3, and the Sobel operator h_4 (see Equation 3.7), which is also applied for edge detection.

$$h_{4_x} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad h_{4_y} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{3.7}$$

Preliminary experiments showed that the 4-neighborhoods Laplace Operator (3.5 h_1) has a convenient response, while the other approximations h_2 and h_3 are responding too strongly to edges; we received too many points. h_4 responded too weak to edges, which means we received not enough points in general.

Too many extraction positions will result in later processing in too high computational costs. Therefore we are not interested in using every point of the median axis. It is sufficient to extract a small fraction of possible points. Hence we decide to apply h_1. There are some rare cases in which the 4 neighborhoods Laplace Operator does not produce enough points, but we can catch that, with an additional rule in the algorithm. If there are not enough skeleton points found, we switch to the 8 neighborhood expression. Another option is to lower the acceptance threshold, in these rare cases. In general we are interested in points, which got at least a response of $x <= -2$. The reason for that is that the second derivative will give greater response in cases where the shape border is very near. Since we want skeleton points, which are not near the border, response above -2 can be ignored. Figure 3.11 illustrates this in a 1-D example. This way we discard nearly all responds to character edges, accepting only those which are truly within the shape. As we can see in Figure 3.12c negative values (blue color) appear where we expect them to be. In very rare cases this threshold discards too many points, where a dynamic increased threshold is activated until an adequate number of points is reached. Another solution for that problem would be a switch to the 8-neighborhoods operator. There is also a third possibility. We could apply Laplacian of Gaussian (LoG), which would need a different threshold. Since it smooths the image first, response will be broader and less accurate. On the other hand thresholding can be applied much finer than in previous methods.

To summarize this chapter: We took a look at the time complexity of our four quantization methods and some visual examples, which can be seen in Figure 3.13. While random points are determined nearly instant, ITVQ and k-means have both polynomial time complexity. LDT has linear time complexity, which is a great advantage in our case, because we need to apply it very often. Further information about performance of these methods will be given in Chapter 4.



(a) Random centroids (b) Assign nearest points 1

(c) Shift centroids (d) Assign nearest points 2

Figure 3.7: k-Means Steps – (a) Initialization with random centroids, (b) assign vectors to the nearest centroid, (c) moving centroids to new mean of points position, (d) reassign vectors to new centroids

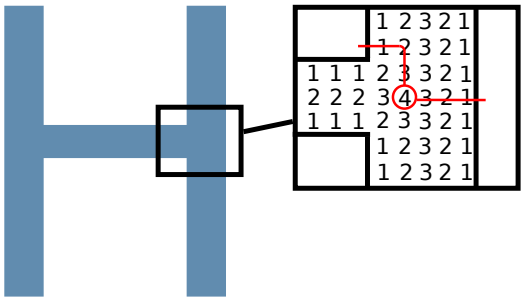
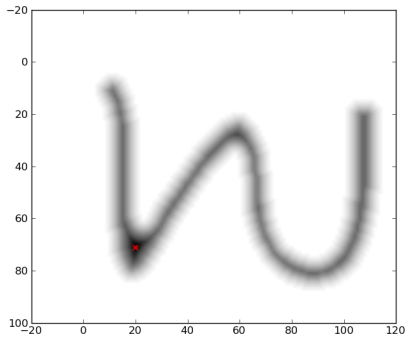
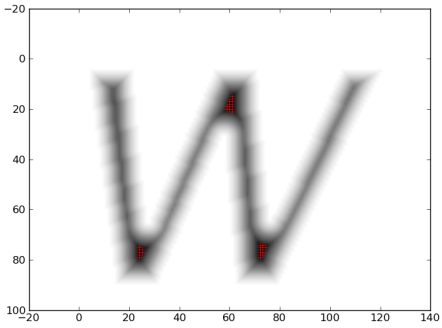


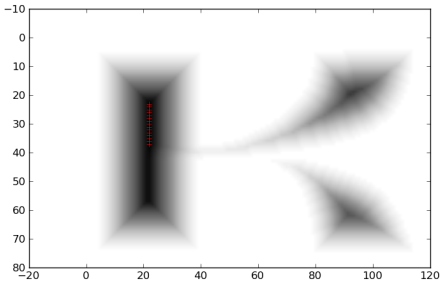
Figure 3.8: The distance transform is a map of the size of the input image, where at each pixel position minimal distance to the next white pixel is stored. This example uses the L1 distance. Other distances can be applied as well.



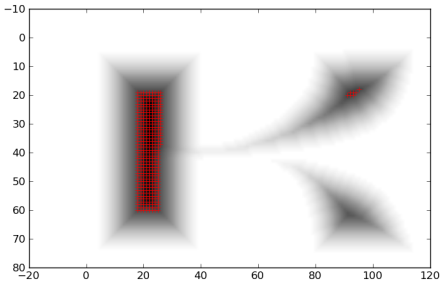
(a) One



(b) Three

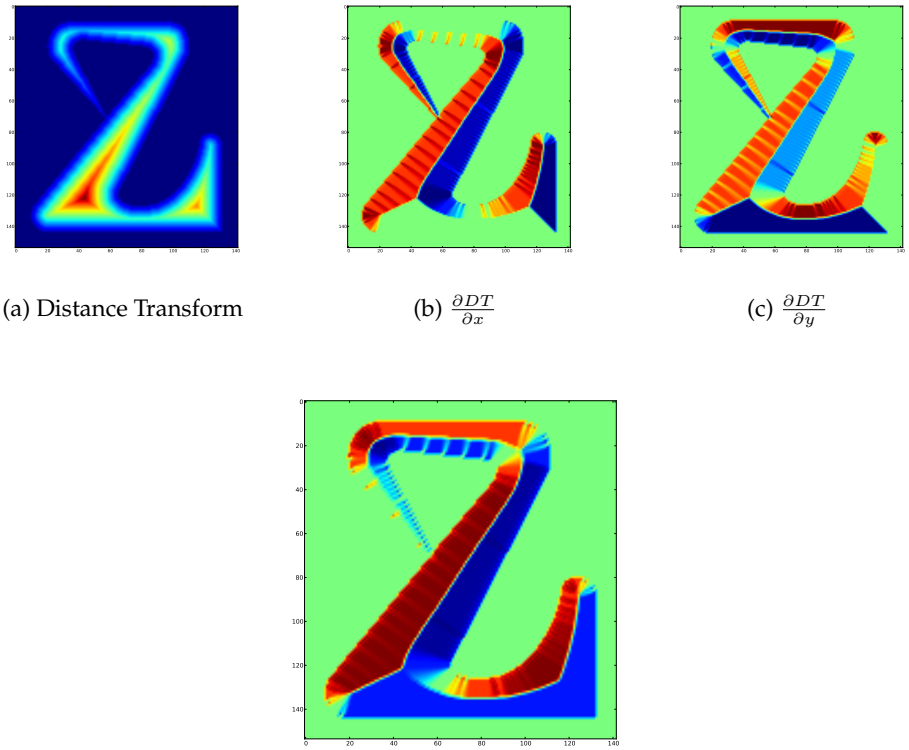


(c) Some



(d) Too many

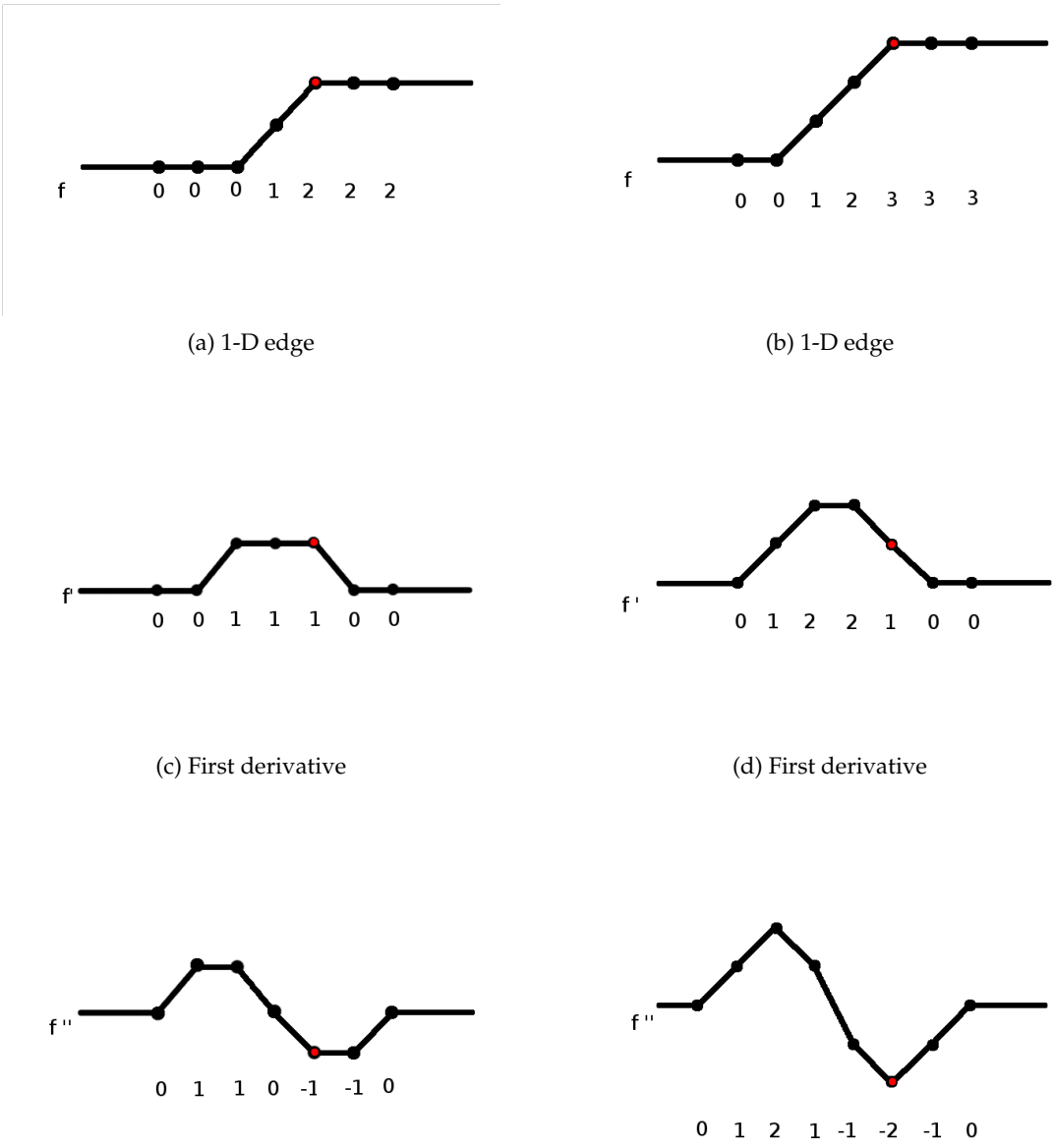
Figure 3.9: Here are some examples for extracting thresholded maximum values from a distance transform. Picture (a) shows only one maximum, where acceptance threshold is set to 0. In picture (b) and (c) a threshold was used to accept additional pixels with 10% smaller values than the maximum value. Picture (d) shows what will happen if the threshold will be set to a lower value.



(a) Distance Transform (b) $\frac{\partial DT}{\partial x}$ (c) $\frac{\partial DT}{\partial y}$

(d) $\frac{\partial DT}{\partial x} + \frac{\partial DT}{\partial x}$

Figure 3.10: Representations of a distance transform and their derivatives. (a) is its distance transform and their first derivative in x-axis (b) and y-axis (c). Image (d) is the sum of (b) and (c). We are interested in zero values within the letter, which denotes maximal values of the distance transform and the center of line stroke.



(a) 1-D edge

(b) 1-D edge

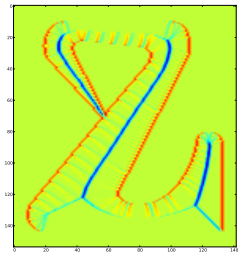
(c) First derivative

(d) First derivative

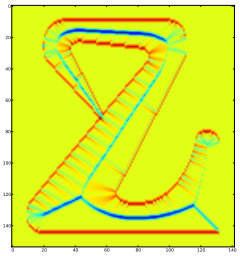
(e) Second derivative

(f) Second derivative

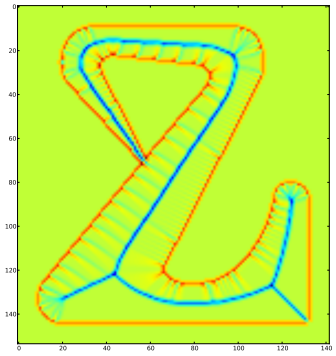
Figure 3.11: Two 1-D example of edge detection via first and second derivative. a), c), e) Edges which are near to the border get -1. b), d), f) while edges which are deeper in the shape get at least -2 responds in the second derivative.



(a) $\frac{\partial^2 DT}{\partial x^2}$

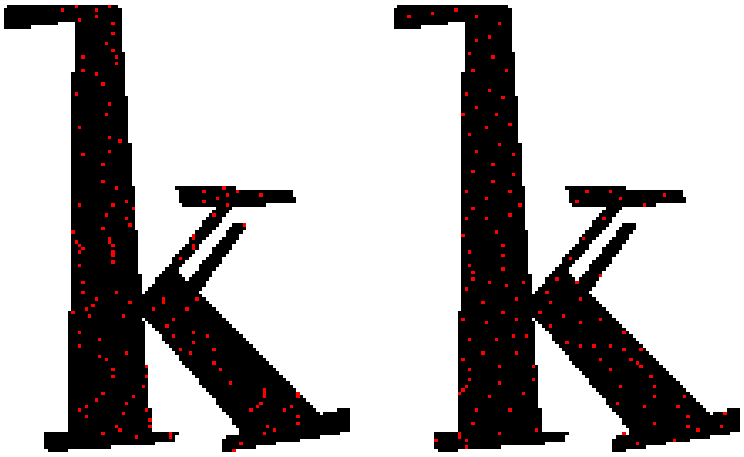


(b) $\frac{\partial^2 DT}{\partial y^2}$



(c) $\frac{\partial^2 DT}{\partial x^2} + \frac{\partial^2 DT}{\partial y^2}$

Figure 3.12: Sub figure (a) shows the second derivative of the x-axis of the distance transform and (b) the corresponding y-axis. (c) is the Laplacian of the distance transform, which is the sum of (a) and (b). Now we can extract skeleton points (see blue line in the middle of the line stroke).



(a) Random (b) K-Means

(c) ITVQ (d) LDT

Figure 3.13: Comparison of sampling methods. Images were sampled with 134 points random, k-means and Laplacian Distance Transform (LDT), while ITVQ example was generated with 150 iterations.

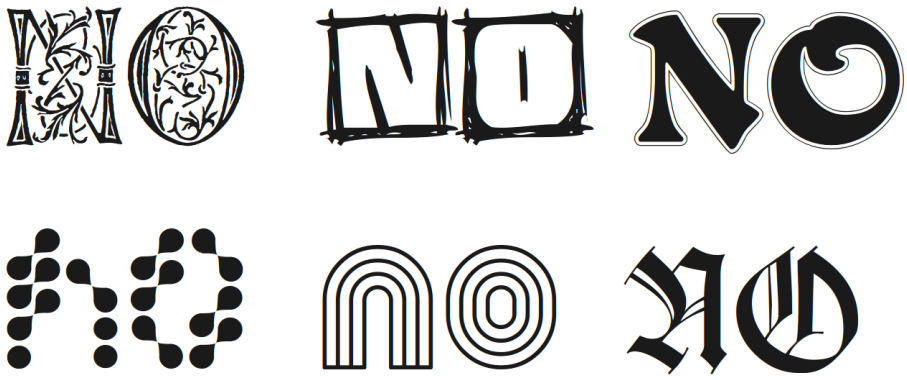


Figure 3.14: 6 fonts which will be a problem for stroke template feature extraction. Most of them would be discarded.

3.4 Features

As has been mentioned before, local features were chosen for this approach. Therefore, three local features will be discussed which could be found in other approaches. Subsequently this section will introduce features applied in the main approach and one additional feature which has been tested.

Stroke Templates An interesting approach introduces stroke templates for font recognition [20]. The authors believe that the stroke shape is the most essential attribute of a font. While they did recognition on several English fonts against Chinese fonts, this assumption becomes true. The idea is also true for many English fonts.

The template extraction is created in several steps. First, the letter image is thinned, which results in a close representation of the letter skeleton. Second, the image is scanned from left to right and from top to bottom. This step is needed to find start positions for further tracking of the strokes. Third the thinned line is followed until a junction is reached. If the processed length is large enough ($L_{min} = 0.27H$, H= letter height), the original letter part is saved as a template. Letters with loops in their shape are discarded. A template is saved only, if there is no other template similar to it, where similar templates have 5% and less mismatching pixels while overlapped.

This feature should be a good idea in most standard font applications. But there are hundreds of fonts which cannot be handled by this approach. For example Gothic print fonts often contain loops in their letters. Most decorative fonts have loops. Since loops are discarded, fonts containing these (see examples in Figure 3.14) cannot be recognized.

Since we want to use a method, which is not restricted to any font, this feature is not an option.

Nevertheless the thinning procedure from this approach was the thought provoking impulse to our LDT image sampling method, which was described in Section 3.3.3.

NMF Letters Another local feature extraction method was introduced by Lee & Jung [12]. Entire letter images are transformed into feature vectors. Those vectors are calculated by Non Negative Matrix Factorization *(NMF)*. NMF creates two matrices **W** and **H**, whose product will approximate the original input matrix **V** (Equation 3.8). Where column vectors of **W** denote basis vectors and column vectors of **H** denote the factorized encodings of the basis. Hence NMF can be understood as a projection from n-dimensional into r-dimensional vector space. Notice that the number of basis vectors r is manually chosen. In the training phase the matrix **W** is adjusted until the approximation of the input is sufficient good. In the event of testing new vectors v_i are projected into the r-dimensional space using the basis **W**, resulting in new coefficient vector h_i. More detailed informations about NMF can be obtained from the paper written by Weixiang et al. [23].

$$V = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ a_{21} & \cdots & a_{2m} \\ \vdots & \vdots & \\ a_{n1} & \cdots & a_{nm} \end{pmatrix} \approx \overbrace{\begin{pmatrix} w_{11} & \cdots & w_{1r} \\ w_{21} & \cdots & w_{2r} \\ \vdots & \ddots & \vdots \\ w_{m1} & \cdots & w_{mr} \end{pmatrix}}^{r} \times \overbrace{\begin{pmatrix} h_{11} & \cdots & h_{1n} \\ h_{21} & \cdots & h_{2n} \\ \vdots & \ddots & \vdots \\ h_{r1} & \cdots & h_{rn} \end{pmatrix}}^{n} \tag{3.8}$$

NMF seams to be a good choice in order to reduce dimensionality of data. Since we want to apply a finer feature extraction, we will explore the performance of this method further by extracting letter parts. Experiments and results are described in Section 4.3.

Eigenfonts The method for feature extraction presented by Solli and Lenz is very straight forward. Feature extraction is done by applying PCA to letter images. Resulting eigenvectors are used as descriptors. Before three Sobel filters are applied to the images (horizontal, $2\times$ diagonal). To the best of our knowledge it is the only method applied to a large dataset [18]. In addition to the mandatory character segmentation, this method has also a very strong prerequisite. Each character must be recognized before processing. In our opinion this is a too restrictive pre-condition. Therefore we will not apply this method.

3.4.1 Hierarchical Recognition Features

The following two paragraphs will introduce features we have chosen for our method. The first feature described, is a filter which limits the numbers of candidates, while the second extracts image patches for final recognition.

Typological pixel coverage

Inspired by the work of Zramdini and Ingold[25] we have implemented an feature to our approach, which calculates the pixel coverage of a font. For each letter's bounding box we calculate the relation between the number of covered and empty pixels. From these we can compute the mean, the highest and lowest value for each font. Which define the weight (see Section 1.1.1) of a font. This feature is used to create a weight hierarchy of the fonts.

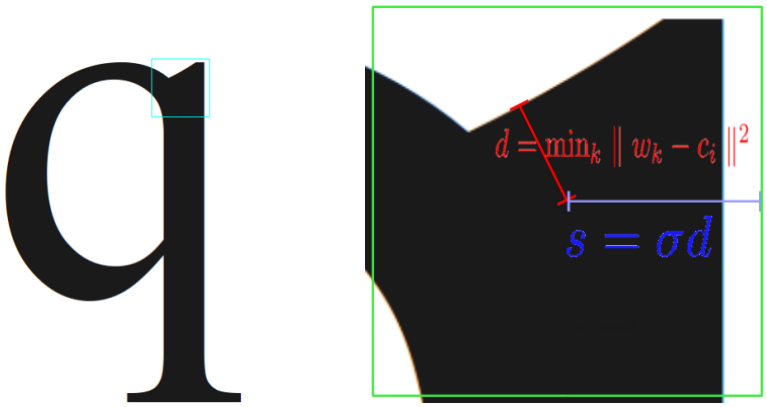
Since the other applied feature is scale invariant, fonts with same line strokes but different weight cannot be distinguished. Therefore the removal of differently weighted fonts will restrain the number of false candidates, which is the main objective of this feature.

Letter snippets

In the preprocessing phase we have computed coordinates $C = c_1, \ldots, c_j$ for feature extraction. Centered around each position c_i we extract a rectangular image patch p_i. The sizes $S = s_1, \ldots, s_n, s_i \in \mathbb{R}$ of the extracted binary patches are automatically determined. We measure the s_i using the distance between c_i and the nearest white pixel and set $s_i = \sigma \min_k \parallel w_k - c_i \parallel^2, w_k, c_i \in \mathbb{R}^2$, where the w_k are the white pixels of the shape image and σ denotes a constant scaling factor, see Figure 3.15 for an illustration. All extracted binary patches are finally scaled to a uniform size, e.g. of 5×5 or 10×10 pixels, using a simple image resizing procedure.

The patch size defines the number of details this feature can capture given a certain scale. While the scaling factor σ is a trade-off parameter. If it is chosen to high, the feature captures most context around extraction position, but concurrently it looses all details. Hence, if it is chosen too small, only details are captured, but details become with no context information worthless.

After the parameters σ and size are set to fixed values. This feature has two important attributes. First, the amount of extracted context depends only on the thickness of the shape. Therefore, patches extracted from border regions will capture details and patches from regions deeper in the shape will describe a more general shape appearance. This leads to the second attribute, which is scale invariance. Given the same shape in different



(a) Extraction region (b) Patch close up

Figure 3.15: Centered on black pixels, binary image patches are extracted. Patch sizes are determined automatically and lead to variations in the amount of context a patch encodes.

sizes, same patches will be supplied if extraction centers are at same relative positions. At this point we have to notice that this is also an unfavorable attribute with respect to font recognition, thus we loose information about the thickness of a shape, which refers to the font weight and contrast (see Section 1.1.1 & Figure 1.2a). Further, this feature captures general shape appearance. It may capture font features or other shape features, it is not dedicated to font recognition only.

The number of extracted patches depends on the size of provided images. Since several patches are extracted from each letter, their overall number is very high. For example, if we extract about 50 patches per letter, the number of patches is above 12×10^6 on our largest dataset. Comparison of vectors of this quantity is not possible in a short time. Further they would require about 18 GB RAM, if loaded at once (assuming 10×10 patch size). Therefore it is essential to create a common alphabet of patch prototypes. This way the number of patches will be lowered to a tractable amount.

As described before in Section 3.2 we create a vocabulary of visual words. First we need a representative dataset as basis. Therefore $10 - 20$ randomly extracted patches from each letter will be stored. Patch positions obey the applied sampling method. In order to compute the needed vocabulary, patches are clustered. Prototypes are computed with the well-known k-Means algorithm, which is explained in detail in Section 3.3.3. The k value is empirically determined. The number of prototypes is much lower than the number of individual patches, $k \ll$ #Patches. Especially in the case of overlapped patches, many of them are very similar, which results in drastic reduction. Obtained k-Means clusters are

saved as the new vocabulary. By applying Vector Quantization to patches, corresponding vocabulary indexes are calculated.

3.4.2 Histograms from Distance Transforms

In separated experiments we evaluated histograms from Distance Transforms as feature for font recognition. The Distance Transform (see preprocessing Section 3.3.3 and Figure 3.8) has been applied for shape recognition, before. One method to recognize shapes with a Distance Transform is the Chamfer Matching [5]. Basically this method recognizes an abstract edge template. This method is not appropriate for font recognition. It is too simplistic to capture small details in the shape as is needed in font recognition.
However, in this case patch extraction cannot be done. Resize operations are not defined on DT. Therefore we consider the entire bounding box of characters for histogram calculation.

We create histograms of the value distribution in a Distance Transform. Depending on the highest value we partition the value range into a fixed number of bins b_i.

$$b_{i+1} = \left[c_i \frac{\max D_j}{B}, c_{i+1} \frac{\max D_j}{B} \right], \quad c_i = 0, \ldots, B-1 \tag{3.9}$$

Where B is the number of bins and D_j is the set of DT values of the image j. Therefore each bin of the histogram represents the number of pixels in the given range. In order to be able to compare histograms we extract values from fixed size images and fonts. Experiments and their results are described in Section 4.2.1.

3.5 Classification

This section will describe classification methods we have applied in our experiments. Histograms and distance metrics, voting decisions and k-Nearest Neighbor have been chosen for classification.

3.5.1 Histograms and Metric

In order to get a single descriptor for a complete font or a single text-snippet, we compute histograms of certain prototypical shape patches. As we describe in Section 3.4.1, the shape prototypes (or vocabulary vectors) are computed during training by applying k-Means clustering.

Given a number of patches, a histogram $\mathrm{H}(\vec{p})$ for all vocabulary vectors $[\vec{c}_1, \ldots, \vec{c}_n]$ that were clustered during the training phase is defined as a mapping

$$\mathrm{H}(\vec{p}) : \vec{p} \rightarrow \mathbb{N}^+ \cup 0, \mathrm{H}(\vec{p})_{\vec{c}_i} := \mathrm{occ}(\vec{c}_i, \vec{p}) \,, \tag{3.10}$$

where $i = 1, \ldots, n$, and n denotes the total number of words in the vocabulary, and $\mathrm{occ}(\vec{c}_i, \vec{p})$ denotes the number of occurrences of $\vec{c}_i$ in $\vec{p}$. Finally, we normalize the histograms. Illustrations of two fonts and their normalized histograms are shown in Figure 3.16. The represented font is described as a distribution over a set of visual words. We do this for two reasons. First, histograms are independent of the number of patches or letters or words, thus we can compare histograms of arbitrary content. Second, certain distance metrics are defined on distributions only. A normalized patch histogram $\phi(\vec{p})$ is then defined as follows:

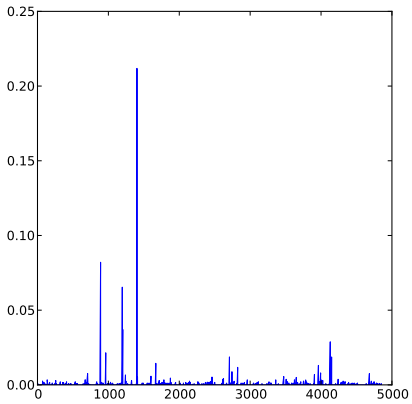
$$\phi(\vec{p}) = \frac{\mathrm{H}(\vec{p})}{\mid \mathrm{H}(\vec{p}) \mid} \tag{3.11}$$

Given the normalized histogram $\phi(\vec{p})$ for a number of patches $\vec{p}$, the classification is performed according to the minimum distance D to a set of normalized training histograms $\phi(\vec{f_i})$ of font $\vec{f_i}$.

$$j = \underset{i}{\operatorname{argmin}}\, D(\phi(\vec{p}), \phi(\vec{f_i})) \tag{3.12}$$

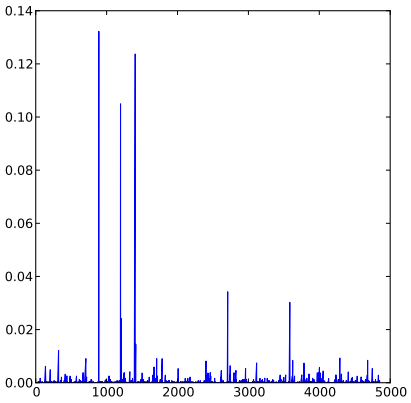
Metrics

We have applied four distance metrics: First, we applied the L1 distance (D_{L1}) and second the L2 distance (D_{L2}), both are very fast to compute. Third, we employed the Kullback-Leibler divergence (D_{KL}). It provides a measure for the disparity of two probability distributions. It is not a true metric, since it is not symmetric. Finally we implement the



(a) Font JumpTroops Histogram

(b) Font JumpTroops Alphabet



(c) Font Dromon Histogram

(d) Font Dromon Alphabet

Figure 3.16: Fonts and their visual word histograms.

Bhattacharyya distance (D_{BA}), which describes the similarity of two discrete probability distributions.

$$D_{L1}(h,p) = \sum_{i=1}^{n} \mid h_i - p_i \mid \tag{3.13}$$

$$D_{L2}(h,p) = \sqrt{\sum_{i=1}^{n} (h_i - p_i)^2} \tag{3.14}$$

$$D_{KL}(h,p) = \sum_{i=1}^{n} h_i * \log_2 \frac{h_i}{p_i} \tag{3.15}$$

$$D_{BA}(h,p) = \sqrt{1 - \sum_{i=1}^{n} \sqrt{h_i * p_i}} \tag{3.16}$$

3.5.2 k-Nearest Neighbor

Some of our experiments apply the k-Nearest Neighbor (*KNN*) classifier to character patches directly. KNN is a simple supervised instance learning method. It can be described in five steps.

1. We have to determine k, which will decide about the number of neighbors.
2. We calculate the distance D between a query instance and the learning dataset.
3. In order to determine the k smallest distances, they have to be sorted first.
4. We gather the labels of the k smallest distances.
5. A majority voting decides about the label of the query instance.

There is no method known to determine the k value automatically. Usually it is set after several empirical tests e.g. 5 or 10. Also the effect of certain distance metrics is not predictable. Furthermore, it is unknown which attributes should be used for comparison exactly.

During testing KNN needs to calculate distances to the entire training dataset, therefore its computational cost is high. Further its recognition performance raises proportional with the size of the training dataset. Therefore, there is a trade-off between recognition rate and computational time. An example where a k-Nearest Neighbor classifier has been applied very successfully to a vast datasets is given by the publication of Torralba, Fergus and Freeman [21]. It shows that even a simple classifier can be a powerful tool. Given training dataset provides a dense coverage of feature space. In mosts application datasets yield a sparse coverage of the feature space, especially if images are involved.

3.5.3 Patch Majority Voting

Some of our experiments apply a majority vote in order to determine the font name. Font names $C = c_1, \ldots, c_m$ are determined through a majority voting from patch prototypes $p_{i,j}$. In training, for each font a list of prototypes indexes is created. While testing, a new list of patches is translated into a list of patch prototypes which are evaluated in the voting. The voting process for an example k is described by the following equation

$$c_k = \underset{\forall j \in C}{\operatorname{argmax}} \; occ(p, j) \tag{3.17}$$

where C is the set of font names, the function occ returns the number of occurring patches p assigned to font j. The font with the highest number of votes is selected.

3.6 Work flow

This section will put previously introduced methods into a brief work flow description of our final approach. Generally we can distinguish between preprocessing, training and testing phase.

Preprocessing First, input text images had to be transformed into binary image space. Second, single characters are extracted separately. Third, positions for feature extraction are found by a image sampling method. We have four methods to choose from: Random selection, k-Means, ITVQ or LDT. Fourth, image patches are extracted at certain positions which were calculated before. Finally, weight of the character is calculated.

Training First, by clustering of a large number of image patches visual vocabulary is created. Second, visual words are calculated for each font and its letters [a-z]. Their word indexes are stored in a normalized histogram. Third, calculation of the minimum and maximum thickness value for the entire alphabet for each font.

Testing First, input's entire weight is calculated. Second, all fonts which does not fit into acceptance window are skipped. Third, image patches are translated into visual words. Fourth, indexes of visual words are summarized in a test histogram. Finally, distances between candidates and test histogram are calculated. Figure 3.17 sketches extraction of three patches and their assignment to prototypes.

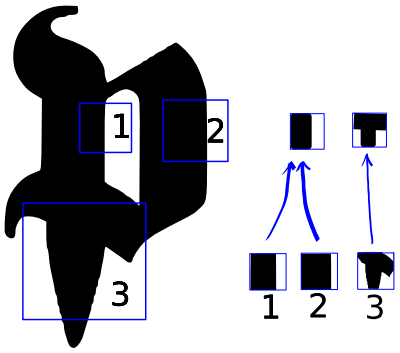


Figure 3.17: Draft of patch extraction and prototype assignment.

Chapter 4

Evaluation

In order to evaluate our approach and features to automatic font recognition we have used four datasets in several experiments. This chapter has following structure:

First, we will introduce our datasets in Section 4.1. The second Section 4.2 will evaluate the application of Distance Transform histograms to OFR and OCR. In the third Section 4.3 results to NMF feature extraction in combination with majority vote will be discussed. Finally, we present extensive experiments and evaluation of the main approach which can be found in Section 4.4.

4.1 Datasets

All four datasets have two things in common. First, lower case letters $[a-z]$ are considered only. Thus, we skipped German Umlauts and other language specific characters, but also capital letters. The reason for this is simple; most letters encountered in the wild are lower case, and not all true-type fonts provide uppercase letters, numbers, or special characters. Therefore, adding other characters would unbalance the training data.

Further it should be noted that (if not described differently), all characters were created in the same size (90 pixels height).

Dataset I The smallest dataset contains 42 handpicked graffiti fonts. Fonts in this collection show distinct differences which should make further recognition easier.

Dataset II The second graffiti font dataset contains 72 different graffiti fonts, the additional 30 fonts were also handpicked from free available sources.

Dataset III Here we collected 747 free fonts, which were provided by the Ubuntu package system. In this collection are standard, decorative, graffiti and also more exotic fonts.

Dataset IV The last and largest font dataset contains 9809 fonts. These were acquired with a self designed and implemented web harvester, which downloaded all free available fonts from `http://www.searchfreefonts.com`. Our program could find over 13000 true type fonts in 45 different categories. However in this collection most fonts are members of decorative, capital or digibats categories.

Unfortunate many files could not be used in further processing. Several did not contain all characters and many could not be processed properly. Further we noticed that Python image library *PIL* is not working properly with some font files. Examples for that are cropped, displaced letters or we received corrupted symbols instead the desired characters. In our case the free available software *ImageMagick* was more reliable than PIL with respect to create single letter images from true type files. However, in the end we could keep 9809 fonts.

4.2 Distance Transform Histograms

This section evaluates histograms of Distance Transforms as feature for font recognition and character recognition.

4.2.1 Distance Transform Histograms for OFR

Distance Transforms has been applied to shape and object detection [8] before. As we explained in Section 3.4.2 known methods cannot be applied here. In the following we present experiments which evaluate the use of Distance Transform histograms for font recognition. Experiments have been carried out on font dataset IV. Test and training examples were created separately, where size of test data images was 20% larger than training. Applied distances between histograms were calculated by the Bhattacharyya metric. If the correct font was within the $5, 10, 15$ smallest distances, it was counted as recognized.

First manually generated histograms seamed promising. Distance measurements indicated distinct differences between font histograms (illustration in Figure 4.1). For initial testing we generated full alphabet histograms instead of words or single characters. Table 4.1 shows the recognition results with various bin partitions. Unfortunately experimental results did not meet our expectations. Apparently the manually generated examples were lucky choices, since overall results of our experiments were insufficient. In a second experiment we introduced a fixed image size of 82×82. Before processing each image was

bins	top 5	top 10	top 15
10 bins	11.9%	16.6%	19.9%
20 bins	7.8%	10.3%	12%
40 bins	4.1%	5.2%	6%

(a) Test word: acfghjstu

bins	top 5	top 10	top 15
10 bins	18.5%	23%	24.1%
20 bins	8.6%	10.6%	12%
40 bins	4.2%	5.1%	5.9%

(b) Alphabet

Table 4.1: Distance Transform Histograms applied for OFR recognition rates with different bin numbers. Dataset IV was used.

Top X	5	10	15
Recognition	6.2%	8.2%	9.4%

Table 4.2: OFR results using histograms of Distance Transforms of rescaled font images to fixed size on dataset IV. Bin number set to 20.

rescaled to this size. Table 4.2 summarizes the results for the second trial with best-in-five scoring. As can be seen precision was even lower than in previous experiments.

Distance Transform Histogram for OCR

In previous experiments we have observed similarities between histograms of equal letters. An example for different fonts but equal characters is shown in Figure 4.2. In order to investigate these observations further we have carried out several experiments, which are described in this section.

In order to probe DT histograms capabilities for OCR purpose, we have chosen 10 simple and rather similar fonts. So the recognition task is simple enough to show the potential of this method, if there is any. Test dataset consists of 20% larger images than data used for the character prototypes. In order to reach equal conditions, we extracted each character and rescaled it with a simple resize operation to a fixed size of 80×80 pixels.

For each lowercase letter a histogram was calculated. Since single values were very dominant we calculate their logarithm values. According to the classification with histogram distances method (Section 3.5.1), prototypes of the letters $[a, \ldots, z]$ were created through the sum of their class members which were in this case the fonts. In contrast to the previous experiments, there is no selection among the fife smallest distances: Since there are only 26 letters we accept only direct hits as positive outcome.

Number Bins	20	30	40	80
Recognition	12.3%	17.7%	18.1%	27.3%
Number Bins	160	320	640	1280
Recognition	36.9%	42.3%	49.6%	53.1

Table 4.3: OCR results using histograms from Distance Transform of font images. 10 similar fonts were used.

As Table 4.3 shows, results were not satisfying. The best precision of 53% was reached with 1280 bin histograms, larger histograms did not show any improvement. Further testing revealed that applications of smaller images result in lower recognition rates. Larger images were not tested, due to very high computational cost.

To summarize this section; Usually simple methods for classification similar to measure distances of histograms, return promising results as long as the chosen feature is feasible. Other classification methods can be applied instead, in order to improve results. In this case however overall precision was too low. It does seam unlikely that histograms of Distance Transforms can be successfully applied to OFR or OCR. However it has to be noted that this results and conclusion are restricted to this specific method as we have designed it.

4.3 NMF and Majority Vote

This section describes experiments which have been carried out in order to investigate if visual words can be used for font recognition applying a majority vote.

Experiments were carried out on Dataset I. The chosen fonts offer rather large differences in their appearance if compared to standard fonts. We assume this makes the recognition task slightly easier. Further, if correct font received enough votes to be in the best-in-five ranking, it has been counted as recognized.

Visual words have been created from small image patches by a NMF algorithm (external implementation). Positions for patch extraction have been calculated by k-Means clustering of black pixels which is described in detail in Section 3.3.3. Classification has been performed by majority vote of patch prototypes (see Section 3.5.3).

Recognition rates were rather low. We tested several parameter settings (number of basis vectors, patch size, scaling factor and voting conditions) but recognition rates did

not exceed 34%. Last option was to exchange the interest point algorithm. k-Means generates a regular sampling of a shape, instead we wanted to apply a method which is more sensitive to the local shape. For that purpose we applied a external implementation of Information Theoretic Vector Quantization (detailed description in Section 3.3.3). Based on the new sampling we received patches which were more descriptive with respect to shape information. Unfortunately, this method resulted in even lower recognition rates. Reason for that was the much lower number of generated patches, hence the resulting number of votes also was smaller and decision through major vote become less accurate.

As we were not able to find proper parameter settings and results did not improve, we inferred that majority vote is not an appropriate method for font recognition with visual words as features. Further we were convinced that we can achieve better results if we introduce another classification method.

4.4 Letter Snippets – Visual Words in Font Recognition

In this section we will present the evaluation of the proposed approach to font recognition.

Classification was performed by comparison of distances between histograms of visual words. Visual words were provided by clustering of image patches with the well known k-Means algorithm. Since implementation details of previously applied NMF algorithm were unknown, replacement of NMF with a simple but well known algorithm seamed to be a logical step in order to produce well-founded results. As we have decided to apply a bag-of-words approach, we have to investigate influential factors which can increase or decrease the recognition performance (all factors were introduced in Section 3.2.2). Among other thinks the following sections we will evaluate our method with respect to several of those factors.

We began with evaluation of parameter settings in Section 4.4.1. The following Section 4.4.2 presents results to word length, segmentation and impact of language specifics. Further the approach was evaluated on the largest dataset, the corresponding results can be found in Section 4.4.3. In Section 4.4.4 we present results which show the impact of the second feature to overall precision. Further Section 4.4.5 presents results of a separated serif – sans-serif recognition based on raw image patches. Finally, Section 4.4.6 illustrates examples of successful and failed font recognition.

4.4.1 Parameter

This section presents experiments whose purpose was to investigate the influence of feature parameters to recognition rate. We will also present experimental results, which will illustrate the importance of choosing a appropriate size of a visual dictionary. Finally, we investigated the influence of different distance metrics to the recognition rate.

Recalling Section 3.4.1 there are two factors which influence the performance of the feature itself. First, the scaling factor σ is responsible for the amount of context captured by each patch. Second, patch sizes define the feature complexity and ability to capture details from extracted context.

Patch Scaling Factor

In several test runs on dataset I the scaling factor σ did not show extraordinary impact on recognition rates, if it is chosen $2 \leq \sigma \leq 4$. However factor $\sigma = 3$ yielded slightly better results than 2 and 4. Further random tests for higher σ revealed most of the patches to be too large, since they enclosed the entire letter. Further this would result in great losses of shape details. Furthermore $\sigma = 1$ is not an option. The reason for that is the patch size definition $s = \sigma d$, where d denotes the smallest distance to the border and s denotes the half width of a patch's extraction context. If $\sigma = 1$ the patch would contain only black pixel. σ-values $1 < \sigma < 2$ produced patches with nearly no context information, therefore they are also not an option. An illustration is given by Figure 4.3.

Patch Overlapping

We did not evaluate feature extraction with non overlapping image patches in detailed experiments. Since several manual examples showed the necessity for overlapping patches in our approach. There are two reasons for that;

First, the feature as we apply it ($\sigma = 3$), would be unable to recognize thick shaped fonts. Since thick shape lead to large context extraction which usually is overlapping. If these are not allowed to overlap most of found patches has to be discarded. This would result in $1 - 4$ patches per character, in this case the recognition outcome would depend strongly on dedicated regions for patch extraction and still be very sensitive to noise and unfortunate prototype assignment.

Second, to avoid small numbers of patches we would need to change the feature scaling factor to $1 < \sigma < 2$, then recognition would be very difficult since single patches would contain very little shape context. In order to compensate this lack of information too large numbers of patches would be needed. An illustration of these examples can be found

in Figure 4.3. Therefore, application of overlapping patches seams to be the appropriate choice.

Based on these examples we believe that non overlapping image patches will not perform well for font recognition.

Patch Size

In order to preserve local shape information we have to decide about the dimension of the patches. We have tested four rectangular patch resolutions $5 \times 5, 10 \times 10, 15 \times 15, 20 \times 20$. Experiments were done on 42 fonts. The test dataset contained 20 different words printed in each font. Character segmentation was not applied. Result Table 4.4 shows our results. As we can see patch size 10×10 showed the best results, while 15×15 got the second best recognition rates. This is also convenient with respect to computational time, since each patch is represented by 100 values only.

Size of the Vocabulary and Distance Metrics

This section evaluate the importance of the size of the visual vocabulary and different distance metrics. Further we present results which show the necessity to apply a best-in-five selection for recognition. Here we present experimental results from a dataset containing 72 different fonts. Further we did not apply character segmentation. For testing we created 20 distinct word images, their length varied between $4 - 6$ characters.

As Table 4.4 indicates, a large number of prototypes raise recognition rates. Since the vocabulary is crucial to the outcome, we have to investigate this topic more carefully. Further we need to examine various distance measurements.

For feature translation we considered six codebooks, where the number of visual words is 100, 250, 500, 800, 1750 and 2000. For recognition we applied four distance metrics, which have been introduced in Section 3.5.1. Table 4.5 contains the experiment results, in addition Figure 4.4 illustrates the results in histogram plots.

Number of visual words

Given this results we notice two important facts. At first, precision raises proportional with the number of visual words until a certain boundary is reached. If the dictionary is large enough, features which describe *similar values* are assigned to visual word v_i, while other features are assigned to visual words v_k, where $k \neq i$. Here the *Curse of Dimensionality* can be found, therefore the vocabulary should be chosen only as large as needed. Second, too large vocabularies result in histograms being sparsely occupied. They do not

show significant distances, hence they cannot be distinguished. This can be seen if we take a closer look at recognition rates from vocabulary 1750 and 2000 and compare them to 1500, which is illustrated in Figure 4.4. Precision begins to drop if the codebook is larger than 1500. Unfortunately there is no known method to estimate the required number of visual words preliminarily. This number has to be assigned manually.

Distance metrics
As we compare the overall recognition rates from these experiment. Bhattacharyya Distance 3.13 is clearly the best choice here. It also has a geometric interpretation, where it describes the square of the angle between two position vectors. This is a more abstract way to compare two vectors than direct distance comparison, which is in our opinion a reason why it performs better than other measurements. Several other reasons why it is a convenient choice are given in the publication from Aherne and Thacker [2].

Precision rate – best-in-x
Finally, as precision values are compared between direct hit, best-in-five and others, it can be seen that increasing the acceptance window by 5 raises recognition rates differently. Between best-in-fifteen and best-in-ten precision increase by 5%, if we compare best-in-five with best-in-ten we precision increased by 9%, but if we compare direct hit with best-in-five recognition rate increased by 35%. Thus, the correct font is usually in the best-in-five section. Since, we choose from rather similar fonts it seams an acceptable choice to accept font as recognized if it is within the five smallest distances. It has to be noted that we referred to results achieved with 1500 visual words and applied Bhattacharyya Distance.

To summarize the most important results in this section; the chosen feature extraction method achieves best results, if patch size is set to 10×10 and scaling factor is set to $\sigma = 3$. Further, the number of visual words must be found empirically with respect to the size of the current dataset. It should be noted, that a large vocabulary leads to sparse representation and comparison of histograms may become difficult. However, later we will show an indication to the existence of an upper bound on codebook size to our approach (more in Section 4.4.3). Finally, experimental results showed that the favorable metric for histogram comparison is the Bhattacharyya Distance.

4.4.2 Segmentation, Word Length & Language

Several local features can be extracted from specific characters only. Others can be captured at particular locations only. Consequently, there are characters which carry more

font properties than others. Further the number of characters provided to the recognition process is important as well. Knowing this, we have to ask the following questions: Which characters carry most font characteristics? How many characters are needed in average? How many do we need in best or worst case for accurate OFR results?

We can go further and compare languages on the basis of their relative character frequencies. In addition to those questions we have also investigated the importance of character segmentation to our approach.

To answer these questions we have conducted a lot of experiments, which have been carried out on 747 fonts from dataset III. We have created large amounts of synthetic test data (in total more than 500000 images). To avoid artifacts, following experiments have been repeated several times - results are averaged over multiple trials. In order to receive the best possible visual vocabulary, several different versions have been tested. Their size varied between $1490 - 5900$ patch prototypes. Best results could be achieved with the number of 4853 prototypes, which was the result form clustering of $420k$ patches and 5000 centroids. Following experiments were carried out with that codebook.

Single Characters First we evaluate OFR for single used character. Table 4.6 summarize the results. Interestingly, the recognition rates differ strongly among individual characters. The letters 'd,p,q' ($> 70\%$) seem to be best suited for font recognition, whereas 's,v,x,z' ($< 30\%$) gave the worst results.

Word Length & Segmentation Second, experiments were done with different length of words. Table 4.7a summarizes results from experiments with no character segmentation. While precision increased with more letters, recognition rates above 85% were not reached on average. As we repeated the experiment with applied character segmentation recognition rates increased significantly. Table 4.7b summarizes the results. Here precision above 90% was reached with four letters in average. It can be seen that segmentation is indeed an important aspect of OFR, although it does not lower the results drastically. However further increasing of the word length improved the recognition rate only slightly. Precision in case of no character segmentation is lower because of additional noise in single patches. Since we worked on synthetic data this noise was created by near other letter shapes, which were captured while patches were extracted.

Random vs. Regular Sampling Also, we repeated the random word recognition experiment from the previous section, but instead of k-Means sampling we considered random point selection. Here precision was in average 10% lower than with k-Means image sam-

pling. If we had an application where speed is more important than precision, random sampling becomes an option.

Best & worst words Further experiments considered recognition with best and worst possible character combinations, their results are presented by Table 4.7c and Table 4.7d. Interestingly worst combinations need 6 characters to reach precision above 90% which is two more than the average case.

Language Finally, we evaluate the influence of different languages to OFR. Therefore we choose, according to the relative word frequency of English and German, the 100 most popular words. From these collections we inducted 200 datasets; one word per trial. Based on these generated words, we estimated 87.99% as an average recognition rate for German words, English came up to 86.50%. However, it should be noted that the 100 most frequently used English words are on average shorter than the ones in the German language. Additionally, we give an estimate for OFR dependent on the derived letter statistics combined with the known letter frequency in a given language. Results for the five most frequent letters are summarized in Table 4.8. Here, English gave the best results, followed by Spanish, Italian, German, and French. Although with respect to letter frequency western European languages are rather similar, we could still observe differences in recognition.

To summarize the most important results; in the optimal case, using the most discriminable characters (d,p,q), three characters can already lead to a precision of 91% on 747 different fonts. However, in the worst case (the letters v,x,z) precision achieves only 58%. On average (random selection), three distinct characters lead to a precision of 87%. Often only 3-5 characters are sufficient, using more than 5 characters often leads to rates above 90%. Thus, assuming a sufficiently large text snippet, font recognition can be done reliably with a very high best in five accuracy, i.e. the correct font is within the list of the five best matching fonts.

4.4.3 Expanding the Dataset

In previous sections we have investigated the importance of parameter settings of the feature itself. Further we explored the influence of input length and information value of different characters to our approach. Here we present evaluation of our approach on a large dataset of 9809 fonts (dataset IV).

Here experiments conducted the vocabulary from previous experiment section, where dataset III has been applied. There are two reasons for that. First, the previous dataset

is representative to most typefaces therefore its vocabulary stays suitable for other larger datasets. Further new elements are not likely to be found, since the old vocabulary provides 4853 prototypes which is a high number if we consider binary images of the size 10×10. Second, histograms created with this codebook were still sparsely occupied, therefore we assumed it will be suited to the new dataset too.

Average Precision Experiments were carried out on various word lengths and several randomly generated words for each trial. Table 4.9a summarizes the averaged results. It can be seen that recognition was difficult if the number of considered characters was $2-5$ only. If the number of characters was higher than 5 we reached convenient recognition rates in the very best-of-fife selection. We could also make an interesting observation if we compared this results with Table 4.7b from previous much smaller dataset. Recognition rates were about 1% lower than on previously presented results if words of length 9 were considered. However comparison of shorter words showed distinct difference in recognition. If only $2-4$ characters were supplied precision varied between $43\% - 76\%$ on our large dataset, where on the smaller dataset of 747 fonts, 2 characters achieved 76% and 4 characters already exceeded 90% precision. However this was not surprising since many fonts offer similar characteristics and more information is needed for certain fonts.

Skeleton sampling vs. k-Means Further experiments considered a new interest point extraction, which returns locations from the character skeleton. Its details are introduced in Section 3.3.3. These experiments were carried out to show if a regular sampling performs better than a locally restrained one. In this case we expected LDT to perform better. Patches which are extracted from the character skeleton, are most descriptive and concurrently their location is stable in different scales. Therefore comparison should be easier than in the other case where patches are extracted from arbitrary positions. It should be noted that test and training data were created in different sizes (10% difference in height), since LDT returns exactly the same points, if supplied images are the same size.

Table 4.9b summarizes obtained results. As can be seen recognition results were similar to the previously obtained results but in average about $6\% - 18\%$ lower than before. In average we did not exceed 75% with LDT sampling method, where regular sampling exceeds 92% recognition rate with same number of characters supplied. If 9 characters supplied locally restricted patches resulted in average precision of 74%, where the previously introduced method needed only 4 characters.

However it has to be noted that, these experiments demanded a newly generated vocabulary. Since regular sampling feature extraction produced different image patches

than a skeleton oriented patch extraction. The conducted vocabulary contain 4182 prototypes while previous vocabulary consists of 4853 visual words. Since vocabulary is used sparsely we assume that the size difference between the two vocabularies has very little influence on the outcome, if any.

While we created the new vocabulary, we experienced an interesting property of the LDT patches. In the process of clustering the number of patches collapsed several times to a number near to 1200 prototypes. Since it was not possible to apply all extracted patches for computational reasons, we had to repeat clustering with different patches. This may be also the reason why the LDT patch extraction method did not outperform regular sampling. If number of centroids collapsed in the clustering process we could infer that large number of patches are very similar. Hence too many visual words were too similar, which finally resulted in lower recognition rates. Further we did not repeated experiments with 1200 visual words vocabulary. We are sure this would result in very low recognition rates, since we needed a larger vocabulary for 72 fonts in experiments before4.4.1.

False Vocabulary Finally we have also conducted several experiments applying an inappropriate vocabulary. These were meant to indicate the importance of a proper vocabulary. However recognition results drop about another 20%, if patches were extracted based on the skeleton sampling and translation was performed with the codebook from regular sampling. Therefore we can see that it is more important to generate a new codebook if patch positions change, than if new data samples are provided.

4.4.4 Adding shape thickness feature

In Section 3.4.1 we have indicated a weakness of our feature: shape thickness cannot be captured. The additional feature should balance this weak point. The following experiments present its impact to the approach. Fonts which do not fit into the acceptance window of $\pm 10\%$ of input thickness are considered as irrelevant.

Table 4.10a and Table 4.10b summarize the results obtained with features extracted based on regular and skeleton sampling. The comparison with previously received results reveals an overall improvement of 2% in recognition. Further we noticed a significant speedup in the comparison of histograms. As can be seen in the second table, in average 64% of font candidates were removed by the font weight feature. It has to be noted that the acceptance threshold of 10% has been estimated empirically. More accurate threshold values require knowledge about the input length and an estimate of possible errors related to the number of characters.

To summarize these experiments; If few characters ($1 - 4$) are provided recognition is difficult ($< 80\%$) on dataset IV. However if more characters are provided convenient results are achieved with high recognition rates above 92%. If a shape thickness feature is added, precision can be increased by further 2%. Further regular shape sampling shows better recognition results than skeleton (locally restrained) sampling of the character shape.

4.4.5 Serif or Sans-Serif Classification

The most known font categories are serif and sans-serif fonts. Serifs can be found in upper and lower regions of a character.

One uprising question was, whether it is possible to recognize serif and sans-serif fonts by direct comparison of their image patches. As serifs does not occur in the middle of a low-case character, we considered patches from the upper and lower third of a letter only. Extraction positions were given by the LDT sampling method, explained in Section 3.3.3. The following experiment has been introduced to single image patches which were extracted from 330 serif and sans-serif fonts, which were obtained from dataset IV. These are over 17000 image patches where each consist of 10×10 pixels.

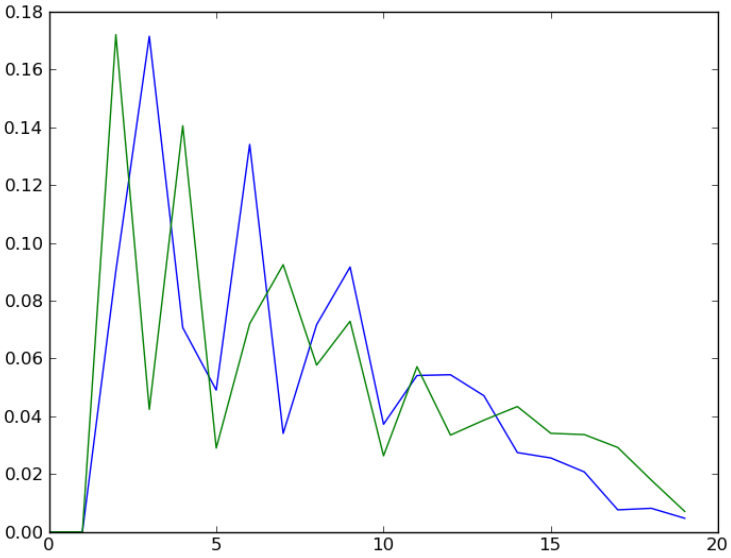
Classification has been performed with a k-Nearest Neighbor classifier, which has been introduced in Section 3.5.2. Table 4.11 summarizes the result from 5-cross-folds validation tests of serif font recognition. As can be seen, recognition of a serif font could be done with an average precision over 75%. Concurrently sensitivity and specificity are at 80% and 70%. We have repeated these experiment with smaller patches of the size 5×5 and received similar but about 5% smaller recognition rates. However it has to be noted that, we considered single letter snippets from certain image regions only. Several of the patches were not extracted from proper positions, further the letter *o* does not contain serifs. Therefore better extraction method would probably increase precision.

If we consider agglomerated patches from one letter, simple majority vote will result in a convenient serif font classifier. Which could be applied as an additional hierarchy level into the recognition process.

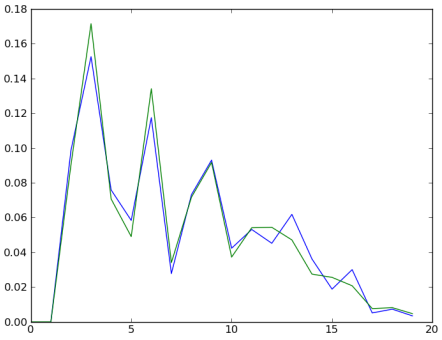
4.4.6 Illustrations of best-in-five results

To give a better impression of our approach, this section illustrates two results where our approach has found the correct font within the best-in-five selection and two examples where the correct font was not in the selection. Figure 4.5 visualizes these four examples, where the first line in each picture shows the font for which we were looking. The illus-

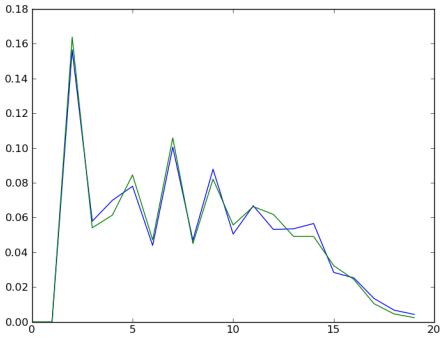
trations show that fonts within the best-in-fife selection are similar to each other, which is the desired result. Further investigation of our results revealed that if a font is not recognized, it is often a sans-serif font with a very simplistic shape (illustration Figure 4.5c). Since these fonts are very difficult to distinguish by shape descriptors only, this result is not surprising.



(a) Two different fonts (Angleterre-Book & Chancery-Cursive)



(b) Font Chancery-Cursive & word „ *afgjtux* “ in the same font.



(c) Font Stonesans 2 & histogram of word „ *afgjtux* “ in the corresponding font.

Figure 4.1: Histograms of distance transforms - Example words and their font histograms

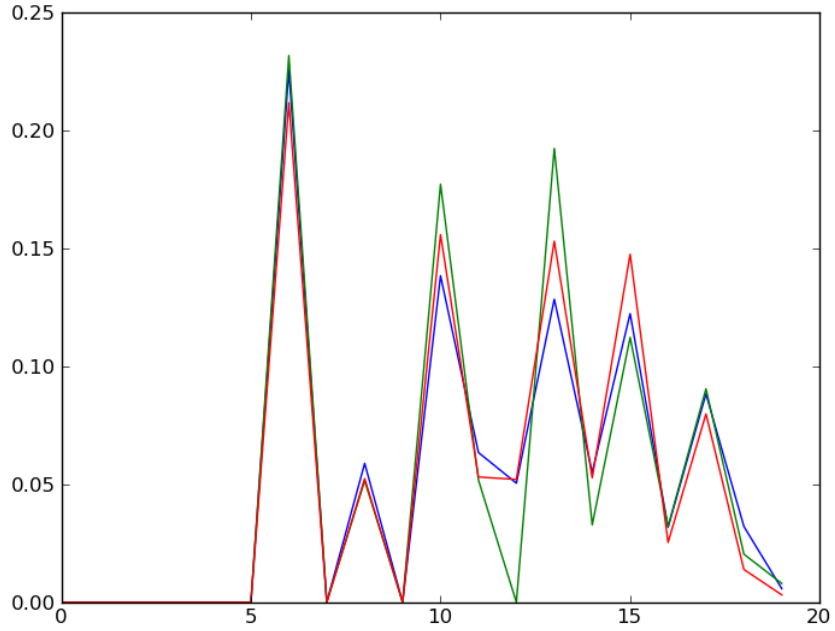
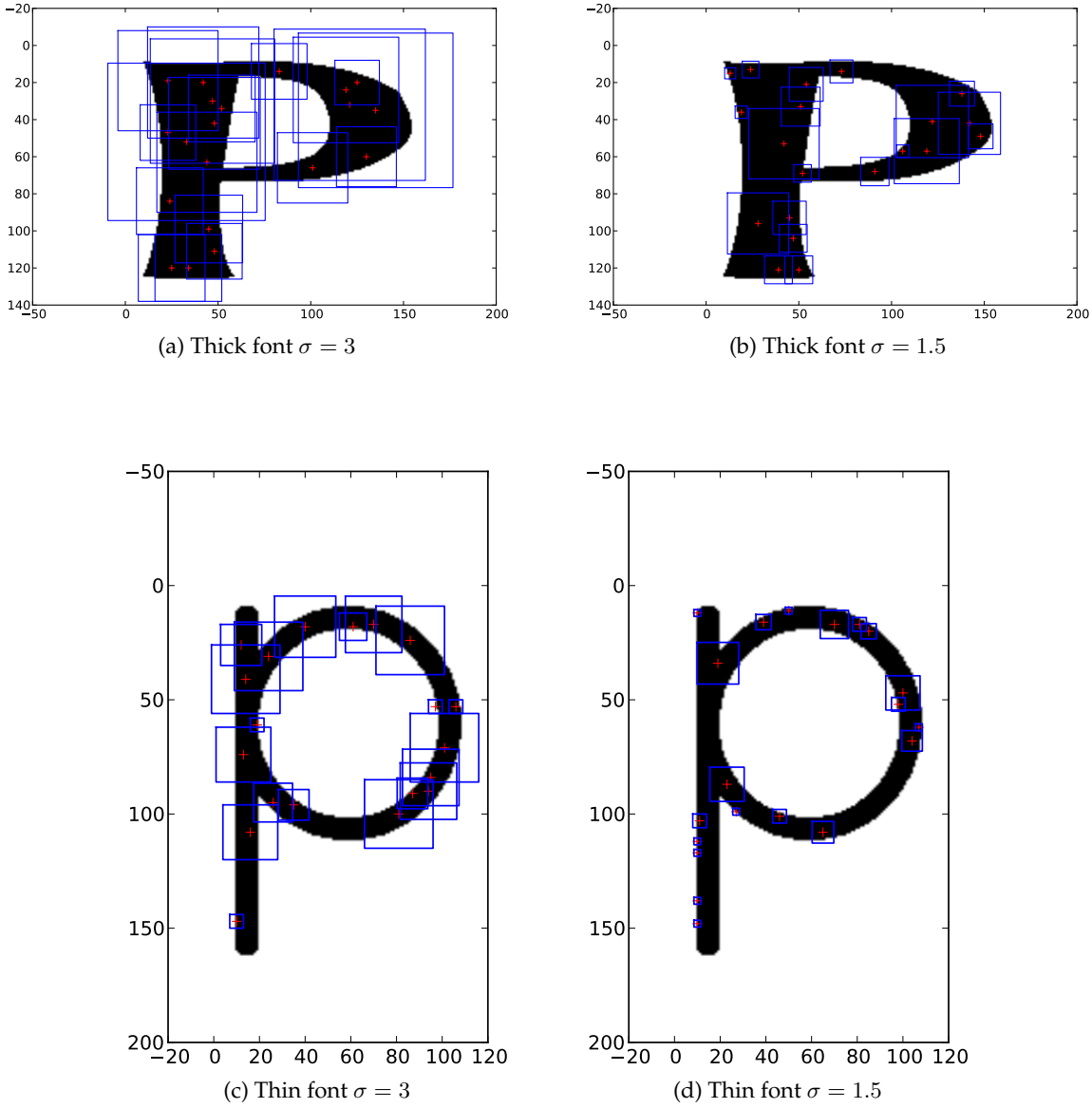


Figure 4.2: 20 bin histograms of the letter w from three different fonts.



(a) Thick font $\sigma = 3$

(b) Thick font $\sigma = 1.5$

(c) Thin font $\sigma = 3$

(d) Thin font $\sigma = 1.5$

Figure 4.3: Extracted regions by image patch feature for different scaling factors.

Patch size	5 x 5	10 x 10	15 x 15	20 x 20
Top 5	56.0%	57.7%	59.6%	**60.5%**
Top 10	79.0%	79.2%	**79.9%**	79.9%
Top 15	90.8%	91.0%	**91.7%**	91.7%

(a) Result on 42 fonts with different patch sizes and 50 prototypes

Patch size	5 x 5	**10 x 10**	15 x 15	20 x 20
Top 5	68.3%	**68.4%**	67.7%	64.0%
Top 10	**87.4%**	87.0%	85.8%	84.8%
Top 15	94.6%	**94.8%**	94.4%	93.2%

(b) Result on 42 fonts with different patch sizes and 250 prototypes

Patch size	5 x 5	10 x 10	15 x 15	20 x 20
Top 5	69.5%	**71.2%**	70.6%	70.6%
Top 10	87.6%	**88.6%**	87.1%	87.3%
Top 15	94.6%	**95.5%**	95.4%	95.0%

(c) Result on 42 fonts with different patch sizes and 500 prototypes

Table 4.4: Testing patch size. Subtables show results from several tests where various numbers of prototypes (50, 100, 250 and 500) and patch size ($5 \times 5 - 20 \times 20$) were set. Dataset I was used.

words	direct	top5	top10	top15
100	0.10%	0.40%	0.66%	0.80%
250	0.13%	0.43%	0.66%	0.80%
500	0.18%	0.56%	0.76%	0.86%
800	0.21%	0.58%	0.79%	0.88%
1500	**0.28%**	**0.62%**	0.80%	**0.89%**
1750	0.25%	0.60%	0.78%	**0.88%**
2000	**0.28%**	**0.62%**	0.80%	**0.89%**

(a) Euclidean Distance

words	direct	top5	top10	top15
100	0.14%	0.47%	0.70%	0.82%
250	0.18%	0.56%	0.76%	0.85%
500	0.33%	0.56%	0.76%	0.85%
800	0.35%	0.72%	0.85%	0.92%
1500	**0.46%**	**0.81%**	**0.90%**	**0.95%**
1750	**0.46%**	**0.81%**	**0.90%**	**0.95%**
2000	0.45%	0.79%	**0.90%**	0.94%

(b) Bhattacharyya Distance

words	direct	top5	top10	top15
100	0.10%	0.40%	0.67%	0.81%
250	0.13%	0.44%	0.67%	0.80%
500	0.21%	0.59%	0.78%	0.87%
800	0.25%	0.61%	0.81%	0.90%
1500	0.34%	0.70%	**0.86%**	**0.93%**
1750	0.32%	0.69%	0.85%	0.92%
2000	**0.35%**	**0.71%**	**0.86%**	0.92%

(c) Manhattan Distance

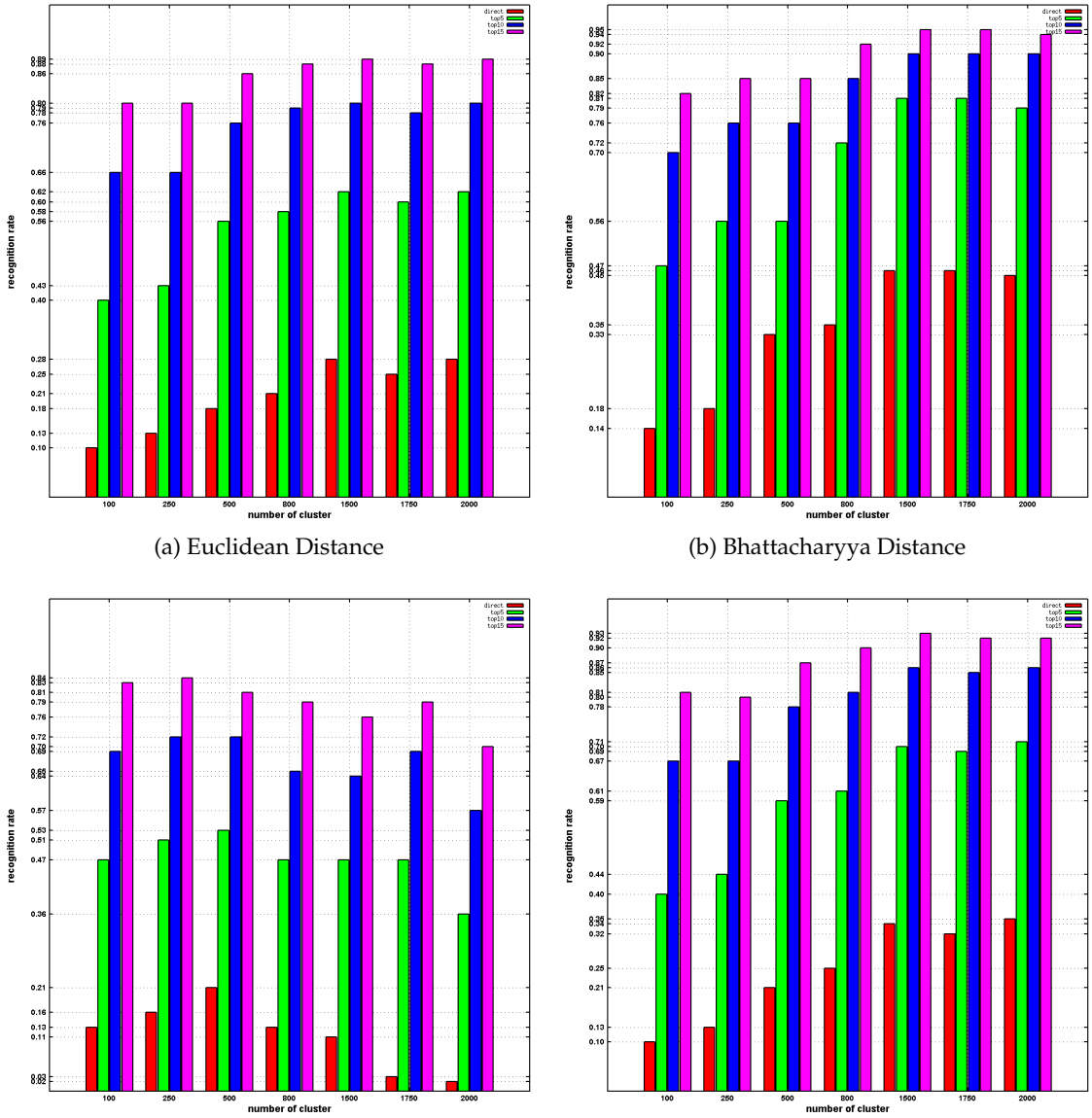
words	direct	top5	top10	top15
100	0.13%	0.47%	0.69%	0.83%
250	0.16%	**0.51%**	**0.72%**	**0.84%**
500	**0.21%**	0.53	**0.72%**	0.81%
800	0.13%	0.47%	0.65%	0.79%
1500	0.11%	0.47%	0.64%	0.76%
1750	0.03%	0.47%	0.69%	0.79%
2000	0.02%	0.36%	0.57%	0.70%

(d) Kullback-Leibler divergence

Table 4.5: Application of diverse distance measurements and distinct vocabulary sizes on dataset II.

d	**74**%	a	56%	k	48%	w	30%
p	72%	f	55%	c	46%	s	26%
q	70%	n	54%	e	42%	v	25%
b	69%	m	52%	y	41%	x	20%
r	60%	h	50%	j	39%	z	**16**%
g	59%	u	49%	l	35%		
o	57%	t	48%	i	33%		

Table 4.6: Single letter recognition results on dataset III.



(a) Euclidean Distance

(b) Bhattacharyya Distance

(c) Kullback-Leibler Divergence

(d) Manhattan Distance

Figure 4.4: Application of diverse distance measurements and distinct vocabulary sizes. Showing recognition rates for direct hit, top $5 - 15$. Dataset II was used.

Word length	2	3	4	5
Recognition	60.8%	68.9%	73.3%	77.2%
Word length	6	7	8	9
Recognition	78.7%	81.8%	82.0%	82.4%

(a) Unsegmented random characters

Word length	2	3	4	5
Recognition	76.0%	86.1%	90.6%	91.9%
Word length	6	7	8	9
Recognition	92.7%	93.1%	93.4%	93.7%

(b) Segmented random characters

Word length	2	3	4	5
Recognition	89.6%	91.0%	92.6%	93.4%
Tag length	6	7	8	9
Recognition	93.7%	93.7%	93.9%	94.3%

(c) Optimal case character selection

Word length	2	3	4	5
Recognition	33.2%	58.3%	75.1%	84.0%
Tag length	6	7	8	9
Recognition	91.1%	92.0%	92.4%	92.4%

(d) Worst case character selection

Table 4.7: Importance of segmentation and specific characters to OFR. Dataset III was used.

English	German	French	Italian	Spanish
$5.3 \rightarrow$ e	$7.2 \rightarrow$ e	$6.4 \rightarrow$ e	$6.5 \rightarrow$ a	$7.0 \rightarrow$ a
$4.5 \rightarrow$ a	$5.3 \rightarrow$ n	$4.4 \rightarrow$ a	$5.6 \rightarrow$ o	$5.7 \rightarrow$ e
$4.3 \rightarrow$ t	$4.2 \rightarrow$ r	$4.0 \rightarrow$ r	$4.9 \rightarrow$ e	$4.9 \rightarrow$ o
$4.3 \rightarrow$ o	$3.9 \rightarrow$ a	$4.0 \rightarrow$ n	$3.8 \rightarrow$ r	$4.4 \rightarrow$ d
$3.7 \rightarrow$ n	$3.8 \rightarrow$ d	$3.6 \rightarrow$ t	$3.8 \rightarrow$ i	$4.1 \rightarrow$ r
⋮	⋮	⋮	⋮	⋮
1.54	1.42	1.42	1.42	1.46

Table 4.8: Language specific font recognition estimates.

Word length	2	3	4	5	6
Recognition	43.3%	69.4%	75.7%	80.9	84.6%
Tag length	7	8	9	26	
Recognition	87.8%	88.0%	92.4%	94.5%	

(a) Image sampling performed with k-Means.

Word length	2	3	4	5	6
Recognition	37.1%	51.1%	58.2%	65.5%	67.9%
Word length	7	8	9	26	
Recognition	71.6%	73.2%	74.2%	82.8	

(b) Image sampling performed with LDT.

Table 4.9: Average recognition results on dataset IV.

Word length	2	4	5	6	8
Recognition	45.6%	75.8%	82.6%	86.0%	89.9%

(a) Image sampling performed with k-Means. Preselection based on font weight.

Word length	4	5	6	7	8
Recognition	60.7%	67.3%	70.8%	73.1%	75.4%
Removed	6076	5978	6284	6570	6846

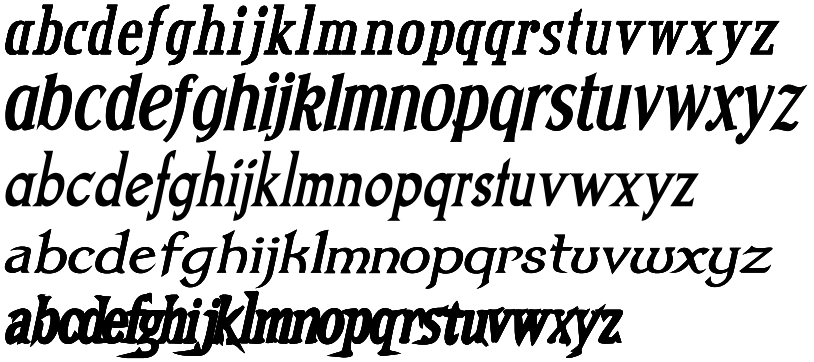
(b) Image sampling performed with LDT. Preselection based on font weight.

Table 4.10: Average recognition results on dataset IV. Preselection of possible fonts applied.

Neighbors	5	10	15	20
Precision	73.5%	75.1%	75.7%	75.5%
Sensitivity	76.9%	79.1%	81.9%	81.2%
Specificity	70.0%	71.1%	69.4%	69.6%

Table 4.11: 5 cross-fold validation results from serif font recognition on 660 fonts from dataset IV.

(a) Positive — Care-Bear



(b) Positive — Credit-Valley-Bold-Italic

(c) Negative — Castor-Gate

(d) Negative — Jolt-of-Caffeine

Figure 4.5: Illustrations from best-in-five selection while main approach was applied. Each first line is the font which was searched for.

Chapter 5

Conclusions

We noticed that many recent publications in the field of document analysis present results on handwriting recognition, which may have drawn attention and interest of many researchers. However, automatic font recognition seams to be forgotten after several methods for small datasets have been published, leaving OFR on large datasets a rather unexplored subject.

Besides, there are practical reasons which may be daunting. One reason may be the lack of free and large datasets. Freely available fonts are not easy to acquire in large numbers. Another problem is the ambiguous name space, which makes it difficult to create reliable training datasets.

We have overcome these obstacles and presented an evaluation of letter snippets as a new feature to the topic of automatic font recognition. By adding a second feature, a two level hierarchic approach has been formed. The joining of applied features balanced their drawbacks and resulted in an overall increased precision of the method. With respect to a visual word approach, we showed in our experiments that a regular shape sampling achieves better results than a locally restricted sampling. In fact, under certain conditions font recognition seems to be an ill posed problem as there is a lack of information when only few letters are supplied. Also, if inconvenient letters are chosen, we cannot expect reliable recognition results. We could show, if supplied number of characters increases, the task becomes more and more manageable, leading to precision higher than 94% for 9809 different fonts. Finally, we have investigated the influence of language specific character statistics to font recognition.

However, comparison of our work with other approaches is very difficult. One reason for that is the disparity of goals, where the task is the same. Where others concentrated on high precision on small numbers of fonts, this work made a point to font recognition

on huge font datasets which demanded another approach to the topic. Hence, the chosen approach relies on a best-in-five selection, there is still room for improvement. Fortunately, our method is flexible enough for further extensions. Nevertheless, we have presented a feasible font recognition approach to a huge dataset which achieved convenient results.

Chapter 6

Future Work

A further increment to the hierarchy of the approach should compensate the loss of local geometry which was introduced by the bag-of-words idea. This can be achieved by tracking the neighborhoods of all visual words. Where the neighborhood is described in probabilities. This additional information can be used to introduce weighted-patches. Where the weight is based on the probability of a patch being member of a given neighborhood. Therefore patches which may contain noise or are simply misleading will receive a lower rating than others and have a smaller impact on the final histogram. Further we could introduce a adaptive patch exchange, where single outliers could be replaced by patches with the highest probability to be there. We believe that an extension of the presented approach with patch neighborhoods will result in an increased recognition rate.

Chapter 7

Appendix

Implementation For implementation we used *Python* as programming language. Further we have made use of some integrated algorithms from two Python libraries. We adopted a fast k-Means implementation from scientific tool library *SciPy* which is based on several optimized *C* methods. Also we applied k-Nearest Neighbor algorithm from the *orange* data mining framework. The most important tool was the module *h5py* which allows storage of vectors and other data in one container file using the hierarchical data format *HDF5*. The reason for its importance is simple; single file access become extremely slow, if we worked with several ten thousands of files. Further we experience a additional speed up while reading data if files were stored with average compression ratio.

Skeleton extraction for other applications As experiments in Section 4.4.3 showed, our skeleton extract method LDT (see Section 3.3.3) did not perform as well as we expected. Still there may be another application for this method. Application which do pose estimation may benefit from our method. It is very fast and could be applied to foreground segmented images in real time applications. Figure 7.1 illustrates an example.

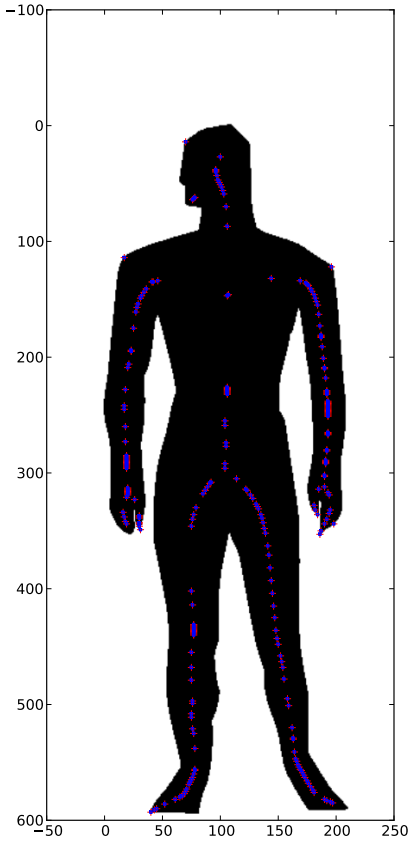


Figure 7.1: Human silhouette processed by LDT sampling method.

Bibliography

[1] G. M. A. Schreyer, P. Suda. A formal approach to textons and its application to font style detection. In *Document Analysis Systems: Theory and Practice*, volume Volume 1655/1999 of *Lecture Notes in Computer Science*, page 739. Springer Berlin / Heidelberg, 1999.

[2] F. Aherne, N. Thacker, and P. Rockett. The Bhattacharyya metric as an absolute similarity measure for frequency coded data. *Kybernetika*, 32(4):1 – 7, 1997.

[3] H. Blum. A transformation for extracting new descriptors of shape. In W. Wathen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, Cambridge, 1967.

[4] J. Böhringer, P. Bühler, and P. Schlaich. *Kompendium der Mediengestaltung*. Springer-Verlag, 2008.

[5] G. Borgefors. Hierarchical chamfer matching: A parametric edge matching algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6):849–865, November 1988.

[6] R. A.-M. R. E.-P. Carlos Avils-Cruz, Juan Villegas. Unsupervised font clustering using stochastic versio of the em algorithm and global texture analysis. *Iberoamerican Congress on Pattern Recognition*, Progress in Pattern Recognition, Image Analysis and Applications Volume 3287/2004 Springer Berlin / Heidelberg ISBN 978-3-540-23527-9:275–286, 2004.

[7] D. F. Elihu Katz. On the use of the mass media as ‚escape': clarification of a concept. In *Public Opinion Quarterly*, volume 26, page 377. American Association for Public Opinion Research, 1962.

[8] D. M. Gavrila and V. Philomin. Real-time object detection using distance transforms. In *Proceedings of Intelligent Vehicles Symposium*, page 998, 1998.

[9] M. Inaba, N. Katoh, and H. Imai. Applications of weighted voronoi diagrams and randomization to variance-based k-clustering. In *Proceedings of the tenth annual Symposium on Computational geometry*, pages 332–339, New York, NY, USA, 1994. ACM.

[10] M. Jung, Y. Shin, and S. Srihari. Multifont classification using typographical attributes. *International Conference on Document Analysis and Recognition*, 0:353, 1999.

[11] P. Kruizinga and N. Petkov. Nonlinear operator for oriented texture. *IEEE Transactions on Image Processing*, 8(10):1395–1407, Oct 1999.

[12] C. W. Lee and K. Jung. Nmf-based approach to font classification of printed english alphabets for document image understanding. Modeling Decisions for Artificial Intelligence Volume 3558/2005 Springer Berlin / Heidelberg 0302-9743 (Print) 1611-3349 (Online) ISBN978-3-540-27871-9:354–364, 2005.

[13] H. Ma and D. Doermann. Gabor filter based multi-class classifier for scanned document images. *Document Analysis and Recognition, International Conference on*, 2:968, 2003.

[14] H. Ma and D. Doermann. Font identification using the grating cell texture operator. DRR:148–156, 2005.

[15] J. MacQueen. Some methodes for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California, 1967.

[16] A. Rosenfeld and J. L. Pfaltz. Sequential operations in digital picture processing. *J. ACM*, 13(4):471–494, 1966.

[17] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proceedings of the International Conference on Computer Vision*, volume 2, pages 1470–1477, Oct. 2003.

[18] M. Solli and R. Lenz. *FyFont: Find-your-Font in Large Font Databases*, volume B.K. Ersbll and K.S. Pedersen (Eds.): SCIA 2007, LNCS 4522, pages 432 – 441. Springer-Verlag Berlin Heidelberg, 2007.

[19] Y. W. Song-Chun Zhu, Cheng-en Guo and Y. Wang. What are textons? In *A. Heyden et al. (Eds.): ECCV 2002, LNCS 2353*, pages 793 – 807. Springer-Verlag Berlin Heidelberg, 2002.

[20] H.-M. Sun. Multi-linguistic optical font recognition using stroke templates. *International Conference on Pattern Recognition*, 2:889–892, 2006.

[21] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30:1958–1970, 2008.

[22] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[23] L. Weixiang, Z. Nanning, and Y. Qubo. Nonnegative matrix factorization and its applications in pattern recognition. *Chinese Science Bulletin*, 51(1), January 2006.

[24] Y. Zhu, T. Tan, and Y. Wang. Font recognition based on global texture analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(10):1192–1200, 2001.

[25] A. Zramdini and R. Ingold. Optical font recognition using typographical features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8):877–882, 1998.

List of Figures

List of Tables