Das OCR-Modul

Das erste Kapitel beschäftigt sich mit Aufbau und Funktionsweise des OCR-(Optical Character Recognition) Moduls. Zunächst werden der allgemeine Ablauf einer Texterkennung, wie sie in diesem Fall zum Einsatz kommt, und einige Grundbegriffe erklärt. Im Anschluss werden die Kernprobleme bei der Erkennung von Devanagari dargestellt, die die Entwicklung eines eigenständigen Texterkennungsprogramms nötig machten. Den letzten Teil nimmt die Darstellung der aktuellen Implementierung ein.

Übersicht

OCR kann als Unterbereich der Mustererkennung per Computer gedeutet werden. Ein solcher Prozess kann in folgende Stadien unterteilt werden (das folgende nach Trier (1996), 2ff.):

Objektextraktion

Aus einem bildlichen Umfeld müssen auf Basis von Vorwissen über Form und Verteilung ein oder mehrere Objekte - in diesem Fall Zeichen - extrahiert werden. Das Programm muss also für den Erkennungsvorgang relevante Bereiche von bedeutungslosen trennen. Dieser erste Programmschritt, der für den menschlichen Leser keinerlei Problem darstellt - auch bei unbekannten Schriften können die meisten sofort Zeilenbereiche und Einzelzeichen markieren - ist für ein Computerprogramm eine erhebliche Hürde, weil es nicht über das menschliche Vorwissen über Schrift und ihre Anordnung auf einer Seite verfügt.

Viele Extraktionsalgorithmen führen als eine Vorstufe die Umwandlung der graustufigen Vorlage (meist 256 verschiedene Graustufen) in ein Schwarz-Weiss-Bild auf Basis von Schwellenwertoperationen durch, wobei resultierende schwarze Pixel mögliche Objekte (Objektpixel), weisse den Hintergrund (Hintergrundpixel) markieren. Der hier eingesetzte Algorithmus sollte gegen bildimmanentes Rauschen und schlechte Beleuchtung der Vorlage resistent sein, weil diese Faktoren zu Fehlklassifikationen von Objekt- als Hintergrundpixeln (und umgekehrt) führen können. In vielen Fällen empfiehlt sich die Anwendung lokaler Bildoperationen. In diesem Fall werden nicht sämtliche Bildpunkte nach denselben Vorgaben behandelt (“Für alle Bildpunkte: wenn ein Punkt dunkler als 111 ist, ist er ein Objektpixel ...”), sondern das Programm berücksichtigt bei der Zuweisung der Pixelkategorie eine eingeschränkte Nachbarschaft des aktuellen Punktes. Wenn diese z.B. sehr dunkel ist, kann es dazu tendieren, einen gegebenen Punkt als Hintergrund zu markieren, obwohl seine Helligkeit deutlich unter dem global gültigen Schwellenwert liegt.

Die Objektextraktion kann auf verschiedene Weisen erfolgen, von denen zwei kurz erwähnt seien. Viele Programme wenden grafische Schablonen an, die typische Objektformen darstellen. Ein Beispiel für dieses Vorgehen ist der häufig beschriebene Roboter, der mit Hilfe von rechteckigen Schablonen Kästchen auf einem Tisch finden und bewegen soll. Eine weitere Möglichkeit ist die Verwendung mathematisch kodierter Objekt-Beschreibungen. Im Rahmen eines OCR-Programms kann man hierunter z.B. die Beschreibung regelmässig wiederkehrender horizontaler Schwarzwerte verstehen, die die Zeilen innerhalb eines Textblocks markieren, oder das Auftreten vertikaler Weissbereiche, die einzelne Zeichen innerhalb einer Zeile

voneinander trennen. Die Objekterkennung kann verbessert werden, wenn dem Programm Bereiche innerhalb eines Bildes bekannt sind, in denen Objekte bevorzugt auftreten. In OCR-Programmen wird dieser Schritt durch die Festlegung von Textblöcken durchgeführt, die entweder auf Basis von Algorithmen gefunden oder vom Benutzer festgelegt werden.

Ein generelles Problem bei der Objektextraktion ist die Verwechslung bedeutungsloser Strukturen mit Objekten. Bei OCR-Programmen zeigt sich diese Tendenz, wenn z.B. hohe waagerechte Striche mit Zeilen kleingeschriebener Buchstaben verwechselt werden. Das Programm muss deshalb entscheiden können, ob ein Bereich, der der äusseren Beschreibung bzw. der bildlichen Vorlage für Objekte entspricht, auch wirklich ein Objekt ist. In diesem Stadium zeigt sich der Übergang zur nächsten Stufe der Mustererkennung, die stärker vorgegebenes Wissen berücksichtigt.

Zeichenerkennung

Die Zeichenerkennung, d.h. die Zuweisung eines Lautwerts zu einer grafischen Vorlage, lässt sich in zwei Stufen unterteilen. Der erste Schritt ist die **Merkmalsextraktion**. Dabei wird die grafische Vorlage, das Bild des Zeichens, mit einem oder mehreren numerischen Werten (Merkmalsvektor) assoziiert.

Ein einfaches Beispiel für solche Merkmale ist das Seitenverhältnis des Rechtecks (bounding box), das das Zeichen umschliesst. Auf Grundlage dieser Information liesse sich in einem OCR-Programm, das auf die Erkennung von Zahlen spezialisiert ist, die Ziffer "1" relativ gut von anderen Zahlen unterscheiden. In den meisten Fällen wird ein Objekt nicht nur durch einen Wert, sondern eine Zahl n von numerischen Werten x umschrieben, die zu einem Merkmalsvektor $v = \{x_0, x_1, ... x_{n-1}\}$ zusammengefasst sind. Eine wichtige Entscheidung besteht darin, ob der Merkmalsvektor für alle Objekte dieselbe Grösse haben muss oder ob seine Dimension variieren darf. Ein Beispiel für die erste Art von Merkmalsvektor ist ein aus vier Zahlen bestehender Vektor, der das Schwarz-Weiss-Verhältnis in den vier Unterrechtecken eines Zeichens beschreibt; für die zweite Art ein Vektor, in dem z.B. Länge und Winkel der Randgeraden eines Zeichens aufgezählt sind - je nach Anzahl dieser Randgeraden würde die Länge des Vektors variieren. Die Auswahl der passenden Kriterien für einen Merkmalsvektor bildet eines der Hauptprobleme jedes OCR-Systems und orientiert sich an der zugrundeliegenden Schrift. So kann z.B. ein Vektor, der die wichtigsten Linien in einem Zeichen beschreibt, für lateinische Schriften sehr geeignet sein, da ihre Zeichen grundsätzlich aus relativ wenigen, gut separierbaren Linien bestehen. Derselbe Vektor kann sich aber für die Extraktion von Merkmalen aus Devanagari-Zeichen als völlig ungeeignet erweisen.

In vielen Fällen besteht der Vektor aus einer Kombination verschiedener Merkmale, so dass z.B. zuerst das Seitenverhältnis, dann Helligkeitsverteilungen usw. eingetragen werden. Aus Sicht der Ausführungszeit sollte der Merkmalsvektor zudem zwei Bedingungen erfüllen. Er sollte einerseits während der Laufzeit schnell zu berechnen sein - ein OCR-Programm für Devanagari muss pro Seite 500 bis mehrere tausend Vektoren dieser Art berechnen. Andererseits sollte er nicht zu umfangreich sein, d.h. seine Dimension n sollte bestimmte Grenzen nicht überschreiten. Mit der aktuellen Computergeneration scheint hier für altsprachliche Anwendungen ein Vektor von mehreren hundert Einträgen noch vertretbar.

Nachdem der jeweiligen Schrift angemessene Merkmale aus dem Bild eines Zeichens extrahiert worden sind, folgt die Aufgabe der **Klassifikation**. Dabei versucht das System zunächst, den Merkmalsvektor einer zugelassenen Objektklassen, in diesem Fall einer Transkription, zuzuordnen. Für die Objektklassifizierung, eines der aufwendigsten und schwierigsten Verfahren innerhalb der Mustererkennung, ist eine unübersehbare Zahl von Ansätzen entwickelt worden, von denen hier nur drei angedeutet werden sollen.

- **Beschreibungen**: Objekte könnten auf Grundlage vorgefertigter Beschreibungen klassifiziert werden. Für ein OCR-System bedeutet das z.B. eine Aufzählung von Merkmalen, die bestimmten Objektklassen zukommen und sie von anderen unterscheiden. Also z.B.: "Wenn ein Objekt in der oberen Bildhälfte einen langen waagerechten Strich, in der rechten Bildhälfte einen langen senkrechten Strich und links eine mittelgrosse Ellipse besitzt und ausser diesen drei Elementen keine anderen, dann ist es ein *va*". Der Ansatz klingt bestechend einfach, stellt das Programm aber vor eine Reihe gravierender Probleme. So werden die Formen eines Zeichens durch bildimmanentes Rauschen oder Unsauberkeiten des Drucks oft so verändert, dass Linien in mehrere Teile aufgespalten werden, Kreise durch Lücken unterbrochen werden usw. Die gesuchten Merkmale können also nicht mehr ohne Vorbehandlung ("Wenn drei kurze Linien mit ungefähr demselben Winkel aufeinander folgen, fasse sie zu einer zusammen" ...) gefunden werden. Diese Vorbehandlung wiederum kann Basis für neue Fehler sein. Ausserdem ist die unscharfe Beschreibung von Positionen ("obere Bildhälfte", "links") und Dimensionen ("lang", "mittelgross"), die für den Menschen leicht zu verstehen ist, für den Computer in dieser Form ungeeignet. Werden Formbeschreibungen, Positionen und Dimensionen numerisch exakt definiert, findet das Programm sie in vielen Fällen nicht wieder, werden sie zu weit gefasst, besteht die Gefahr der Fehlklassifikation. Eine umfangreiche Beschreibung von Varianten in der Regelbasis, die die Grundlage zu einem solchen Ansatz darstellt, ist ausserdem nur mühsam zu implementieren. Einen möglichen Ausweg aus diesem Dilemma können z.B. Fuzzy-Inferenz-Ansätze bieten, die die Unsicherheit sprachlicher Beschreibungen zu einem gewissen Grad mathematisch modellieren können. Auch hier bleibt aber die Aufgabe, eine erschöpfende Regelbasis zu erstellen, die möglichst alle Fälle erfasst und geeignete Fuzzifizierungs-/Defuzzifizierungs-Funktionen zu finden (vgl. Tizhoosh (1998), passim).
- **Vergleich der Merkmalsvektoren**: Eine weitere Möglichkeit der Objektklassifizierung beruht auf dem Einsatz der schon erwähnten numerischen Merkmalsvektoren, die z.B. die Helligkeits- und Linienverteilung in einem Objekt beschreiben. Ein gefundenes Objekt wird dabei auf Basis seines Merkmalvektors mit einer Menge gespeicherter und schon klassifizierter Objekte verglichen. Dieser Vergleich bildet die Grundlage für die Zuordnung des Objekts zu einer der schon bekannten Objektklassen. Eine wichtige Forderung bei der Ausbildung von Klassen besteht darin, dass sich die Objekte einer Klasse möglichst ähnlich und gleichzeitig von allen Objekten der anderen Klassen möglichst verschieden sind, um eine optimale Aufteilung des Klassenraums zu erhalten. Ist das System dynamisch erweiterbar, was für ein Texterkennungssystem (neue Zeichenklassen!) angemessen ist, muss es einen Schwellenwert für noch akzeptable Zuordnungen festsetzen. Liegt die Distanz eines Objektvektors zu allen schon vorhandenen Klassen unter diesem Schwellenwert, muss die Möglichkeit bestehen, eine neue Klasse zu eröffnen. Als Distanzmaß werden verschiedene Methoden verwendet. Eine der wichtigsten ist die euklidische Distanz zweier n-dimensionaler Vektoren.
  Die Vorteile dieses Ansatzes gegenüber dem ersten liegen zunächst in seiner

dynamisch erweiterbaren Struktur. Dem Programm müssen zur Klassifikation nicht die Beschreibungen aller möglichen Objektklassen vorliegen, denn es kann während der Laufzeit auf Basis der Akzeptanz für ein neues Objekt darüber entscheiden, ob für dieses Objekt eine weitere Klasse eröffnet werden soll. Sind die Eigenschaften, die in den Merkmalsvektor einfliessen, zudem geschickt gewählt, zeigt sich ein solches System resistent gegenüber Rauschen und anderen Bildverschlechterungen. Die Nachteile dieses Ansatzes liegen vielmehr in seiner mathematischen Modellierung. So kann ein ungeeignetes Distanzmaß die Ergebnisse der Erkennung drastisch verschlechtern. Gerade bei hochdimensionalen Vektoren (n>>20) verliert zudem die euklidische Distanz mehr und mehr an Aussagekraft über die wirkliche Distanz zweier Merkmalsvektoren ("curse of dimensionality"). Auch die Aufteilung des Musterraums und damit die Modellierung der Einzelklassen ist ein erhebliches Problem, das hier nicht weiter vertieft werden soll (vgl. Paulus (1998), 315ff.). Schliesslich kann dieser Ansatz zu einer deutlich erhöhten Rechenzeit führen, wenn eine umfangreiche Menge von Mustern zum Vergleich durchsucht werden muss. Ein sinnvoller Schritt ist daher die Partitionierung des Musterraums auf Basis einfacher Kriterien, die dem eigentlichen Suchvorgang vorausgeht bzw. eine vor der Zeichenerkennung durchgeführte Reduktion möglicher Klassenvektoren durch Clusterung im Merkmalsraum.

- **Künstliche neuronale Netze (KNN)**: Eine Möglichkeit, die mathematische Strukturierung des Musterraums zu umgehen, ist der Einsatz künstlicher neuronaler Netze (das folgende nach Scherer (1997), 3ff.). Sehr grob gesprochen können KNN einen Eingangsvektor aus einem Datenraum mit einem Ausgangsvektor, der aus einem anderen Datenraum stammen kann, assoziieren. Am Beispiel der Texterkennung: Ein KNN ist in der Lage, den Vektor der Helligkeitsverteilung eines gegebenen Zeichens mit einem Vektor aus Nullen und genau einer Eins zu assoziieren, in dem die Position der Eins die zugeordnete Musterklasse repräsentiert. KNN sind aus Schichten einfacher Schaltelemente, den Neuronen, aufgebaut. Dabei unterscheidet man in mehrschichtigen Backpropagation-Netzen, einer der am häufigsten gebrauchten Klasse künstlicher neuronaler Netze,
  1. **Eingangsneuronen**, die dazu dienen den Merkmalsvektor eines Objekts aufzunehmen;
  2. **verdeckte Neuronen**, die die Ausgabe der Eingangsneuronen aufnehmen und weiterleiten;
  3. **Ausgangsneuronen**, die die Entscheidung des KNN nach aussen sichtbar machen.

  Diese Schaltelemente sind miteinander auf verschiedene Arten verknüpft, die Relevanz dieser Verknüpfungen wird über Gewichte bewertet. Jedes Neuron besitzt einen Eingang und einen Ausgang. Über den Eingang erhält ein Neuron (gewichtete) Eingaben eines vorgeschalteten Neurons oder den Eingangswert des Netzes, die es dann intern nach einer festgelegten Rechenvorschrift bearbeitet. Das Ergebnis dieses Rechenvorgangs wird über den Ausgang des Neurons an nachgeschaltete Neuronen weitergeleitet oder, wenn das Neuron selbst ein Ausgangsneuron ist, nach aussen präsentiert. Der Vorteil neuronaler Netze besteht v.a. in ihrer Trainierbarkeit auf unbekannte und verrauschte Daten, die sich in den Gewichten niederschlägt, und der daraus resultierenden Fähigkeit zur Abstraktion von den Daten der Lernphase. Der Lernprozess wird dabei als fortschreitende Gewichtsmodifizierung zwischen den Neuronen implementiert und solange wiederholt, bis das KNN die zu trainierenden Eingangsvektoren korrekt mit den gewünschten Ausgangsvektoren assoziieren kann, bis also der Gesamtfehler des Netzes minimal ist. Ein trainiertes KNN ist zudem in der Lage, bis zu einem gewissen Grad von den ihm während des Trainings vorgelegten

Daten zu abstrahieren und damit neue Ausprägungen der gelernten Musterklassen zu erkennen, ohne seine Struktur ändern zu müssen. Schliesslich sind KNN, v.a. wenn ihre Struktur nicht erweitert werden kann, während der Ausführung meist bedeutend schneller als herkömmliche Klassifikatoren. Der bedeutendste Nachteil von KNN - sieht man von der passenden Wahl des Merkmalsvektors ab, die das Trainingsergebnis natürlich beeinflusst - ist die teilweise erhebliche Dauer der Trainingsphase und der grosse Umfang der dazu erforderlichen Muster. Schliesslich kann ein KNN auch als Möglichkeit gedeutet werden, das Minimum einer Funktion, der Fehlerfunktion, zu finden. Es besteht daher immer die Gefahr, dass das KNN während des Trainingsprozesses in einem lokalen Minimum "hängenbleibt", deshalb das Training abbricht und nicht weiter optimiert werden kann. Auf den Klassifikationsprozess hat ein solcher Trainingsabbruch ungewünschte Auswirkungen, da das Netz in der Trainingsphase seine klassifikatorischen Fähigkeiten nicht ausbilden konnte und damit unbekannte neue Muster nicht korrekt behandelt.

Darstellung der wichtigsten OCR-Algorithmen

Umwandlung des Graustufen- in ein Schwarz-Weiss-Bild

In einem vorbereitenden Arbeitsgang wird die z.B. über den Scanner importierte Graustufenvorlage in ein SW-Bild umgewandelt. Das Programm muss im Laufe dieses Vorgangs die vom Benutzer markierten Textblöcke so ausrichten, dass der Deckstrich der Devanagari möglichst parallel zur waagerechten Bildkante steht. Ausserdem sollte die Binarisierung mögliche Beleuchtungsfehler der Vorlage auszugleichen versuchen, ohne z.B. die dünnen Linien der übergeschriebenen Zeichen zu unterbrechen.

Eine automatische Textblockerkennung ist in der aktuellen Programmversion nicht vorgesehen. Sie könnte z.B. mithilfe rekursiver Helligkeitsscans des Graustufenbildes auf Basis einer ersten Schwellenwert-Schätzung erstellt werden. Allerdings (man vergleiche marktübliche OCR-Programme) müssen in fast allen Fällen auch die Ergebnisse einer solchen Vorbehandlung manuell verbessert werden, so dass die manuelle Markierung der Textblöcke keinen ernsthaften Zeitverlust darstellt. Der erste Algorithmus besteht aus folgenden Schritten:

Kontrastverbesserung in Bildbereichen mit Schrift

Der Algorithmus zur Kontrastverbesserung greift die Idee der Histogramm-Hyperbolisierung auf (vgl. Tizhoosh (1998), 189ff.). Dabei werden die Graustufen eines Bildes – analog zu Vorstellungen über die Kontrastwahrnehmung des menschlichen Auges – nicht linear, sondern logarithmisch transformiert, um den Kontrast der Vorlage zu verstärken.
In der hier implementierten Variante werden zunächst für eine Reihe von rechteckigen Bildausschnitten die beiden Maxima der lokalen Histogramme, d.h. die Werte für sichere Objekt- und Hintergrundpixel ermittelt. Jeder Bildpunkt wird im nächsten Schritt den zwei nächsten Bildausschnitten zugeordnet. Dabei wird die Zugehörigkeit des Punktes zu den zwei Ausschnitten auf Basis der euklidischen Distanz ermittelt. Die Maxima der beiden zugeordneten Bildausschnitte werden unter Einbezug der Zugehörigkeiten des Bildpunkts gemittelt. Liegt der Helligkeitswert des

untersuchten Bildpunkts unter/über einem dem gemittelten Maxima, wird er als Objekt- bzw. Hintergrundpixel markiert. Liegt die Helligkeit p_{xy} des Punktes dagegen zwischen den beiden Maxima (m_{OBJ}: gemittelter Wert für Objekt-, m_H: gemittelter Wert für Hintergrundpixel), wird der neue Helligkeitswert p_{xy}‘ unter Anwendung der Standardformel errechnet. Das Resultat ist in den meisten Fällen ein „Flickenmuster“, da nur die Seitenbereiche mit Schrift auf den Algorithmus ansprechen. Hier wird der Kontrast der Objektpixel vor dem Hintergrund deutlich und zuverlässig erhöht.

Approximation des Schwellenwerts

Das Programm sucht mit der in (Parker (1997), 35ff.) beschriebenen Methode den globalen Two-Peaks-Schwellenwert (s_{tp}) des Bildes (d.h. optimal den Wert, der in einem globalen Graustufen-Histogramm das Minimum zwischen den Mengen der sicheren Objektpixel und der Hintergrundpixel bildet). s_{tp} stellt einen ersten Schätzwert für die Binarisierung des Bildes dar.

Drehen der Textblöcke

Mithilfe des vorläufigen Schwellenwerts s_{tp} vollzieht das Programm innerhalb des Graustufenbilds eine Hough-Transformation für jeden benutzermarkierten Textblock. Der maximal besetzte Winkel wird als Drehwinkel auf den jeweiligen Textblock angewendet, wodurch der Textblock parallel zur unteren Seitenkante ausgerichtet wird.

Umwandlung in ein SW-Bild (Binarisierung)

Die Binarisierung ist ein wichtiger Schritt zur korrekten Erkennung gedruckten Textes. Werden hier z.B. viele Objektpixel falsch als Hintergrundelemente klassifiziert, wirkt sich dieser Fehler direkt auf die nachfolgenden Algorithmen, v.a. die Segmentierung der Zeilen und die Extraktion von Merkmalsvektoren, aus. In der aktuellen Programmversion sind zwei Binarisierungs-Algorithmen eingebaut, die im folgenden beschrieben werden.

Binarisierung mit Fuzzy-Komponenten

- Ausserhalb der markierten Textblöcke wird einfach s_{tp} zur Segmentierung des Bildes angewendet. Innerhalb der Textblöcke werden dagegen vor Anwendung von s_{tp} drei Fälle unterschieden:
- Liegt der Grauwert eines Bildpunkts maximal 40 Stufen über s_{tp}, wird über eine lineare Zugehörigkeitsfunktion (Stützpunkte: s_{tp} -> 1.0; s_{tp}+40 -> 0.0) die Zugehörigkeit des Punktes z_{lin} zur Menge der Objektpixel ermittelt. Danach werden in einer 3x3-Umgebung die Pixel ausgezählt, deren Helligkeit unter s_{tp} liegt, die Bildpunkte also, die als Kandidaten für Objektpixel fungieren; ihre Menge wird mit einem diskreten Faktor z_{menge} (Stützpunkte: Menge 0 -> 0.65; Menge 8 -> 1.0) bewertet. Aus der Zugehörigkeit z_{lin} und dem Faktor z_{menge} wird ein Abdunklungsfaktor f nach
  $$f = z_{menge} + (1.0 - z_{lin})*(1.0 - z_{menge})$$
  errechnet, mit dem der Grauwert des aktuellen Bildpunkts multipliziert wird.
- Nach Abdunklung der helleren Werte nimmt das Programm eine erste vertikale Schätzung der Zeilen vor. Die gefundenen Zeilenbereiche werden einem einfachen vertikalen Helligkeitsscan unterzogen. Sofern ein dunkler Bereich (Anzahl der Pixel unter stp mindestens 0.6 mal Fläche des Bereichs) von genügender Breite gefunden wird, führt das Programm in diesem Bereich einen

Aufhellungsalgorithmus durch, der dem beschriebenen Abdunklungsalgorithmus ähnelt.
- Nach Aufhellung und Abdunklung wird der Schwellenwert s_{tp} angewendet. In einem letzten Schritt wird eine leichte Rauschunterdrückung durchgeführt. Dabei werden Pixel, die vollständig von andersfarbigen Pixeln umgeben sind, invertiert.

Der Fuzzy-Binarisierungsalgorithmus liefert für Schriftbereiche einer Seite akzeptable Resultate. Allerdings hat er Probleme, Hintergrundbereiche mit graduell zunehmender Helligkeit (z.B. dunkle Bereiche in der Nähe der mittleren Buchfalz) korrekt zu klassifizieren. Diese Bereiche bilden die Motivation für den zweiten Algorithmus, der auf statistischen Informationen über die Helligkeitsverteilung auf einer Seite basieren.

Binarisierung auf Basis der Standardabweichung in Bildspalten

- Der zentrale Unterschied zum ersten Binarisierungsalgorithmus liegt in einem vorgeschalteten Schritt zur Kontrastverstärkung. Dazu wird das Bild zunächst spaltenweise gescannt und für jede Spalte x der Mittelwert der auftretenden Graustufen und die Standardabweichung sa_x gespeichert. Dabei ist zu erwarten, dass Hintergrundbereiche – auch diejenigen mit einem graduell zu-/abnehmenden Helligkeitswert – eine relativ geringe Standardabweichung hervorbringen, während in Spalten, in denen Textzeichen vorhanden sind, eine deutlich höhere Standardabweichung auftreten wird. Seien sa_{MIN} und sa_{MAX} die global gefundenen minimalen bzw. maximalen Standardabweichungen, wird in einem zweiten Schritt jeder Bildpunkt p_{xy} mit dem Faktor
  $$f = 2 - sqrt((sa_x - sa_{MIN})/(sa_{MAX} - sa_{MIN}))$$
  multipliziert. Der Faktor f nimmt für Spalten mit geringer Standardabweichung, in denen eher Hintergrundpixel zu erwarten sind, offenbar höhere Werte an als für Spalten mit Schriftelementen (die Differenz $(sa_x - sa_{MIN})$ wird kleiner, womit sich f eher in Richtung des Wertes 2 bewegt), wodurch der gewünschte Effekt, eine Aufhellung des Hintergrundbereichs, erzielt wird.
- Nach der vorgeschalteten Kontrastverstärkung wird das Bild mit dem Two-Peaks-Verfahren wie in der Fuzzy-Implementierung beschrieben weiterverarbeitet.

Rauschunterdrückung

Im Anschluss an die Binarisierung führen beide Verfahren eine einfache Prozedur zur Entfernung isoliert auftretender weisser und schwarzer Pixel durch. Bei manchen Vorlagen reicht diese Methode aber nicht aus, da sich die Störungen in Gruppen von 3 bis 10 Pixeln gruppiert haben. Für diesen Fall steht ein Perzeptron zur erweiterten Rauschunterdrückung zur Verfügung. Dieses KNN wurde manuell an einer Menge von fünf binarisierten Bildvorlagen trainiert. Dabei tritt jede Vorlage in zwei Varianten auf: Die erste stellt das binarisierte Bild dar, in der zweiten wurden störende Pixel manuell entfernt. Das Netz wird nun darauf trainiert, die erste verrauschte Vorlage in die zweite korrigierte zu überführen.
Bei Anwendung auf ein verrauschtes Bild werden mithilfe des KNN die meisten Störungen korrekt entfernt. Zudem werden einige Bruchstellen innerhalb der Zeichen, die durch falsche lokale Schwellenwerte entstanden sind, wieder geschlossen.

Segmentierung der Zeilen

Der zweite Arbeitsgang im OCR-Modul umfasst die Erkennung der Zeilenbereiche innerhalb der Textblöcke und die Isolierung der Einzelzeichen, d.h. die Objektextraktion aus dem binarisierten Bild der Textvorlage.

Probleme bei der Segmentierung von Devanagari

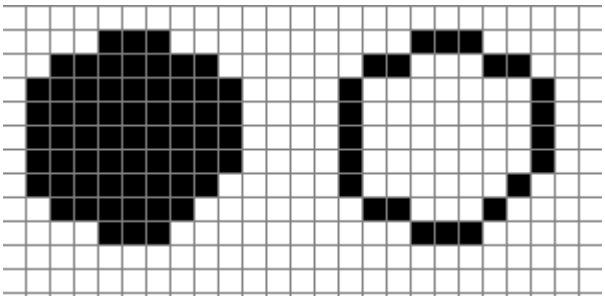
Die meisten OCR-Programme sind auf Ableger der lateinischen Schrift ausgerichtet, in der - grob gesagt - jedes Phonem durch ein eigenständiges Zeichen in der Hauptzeile repräsentiert wird. Die Zeichen bilden eine Sequenz, in der die Position des Zeichens der Position des Phonems im zugrundliegenden Wort entspricht.
Die Devanagari ist nach anderen grafischen Prinzipien aufgebaut, die den Prozess der Texterkennung erheblich schwieriger machen. Hier treten, wie in der unten abgebildeten Darstellung erkennbar, eigenständige Zeichen in drei Positionen innerhalb einer Zeile auf.

- Im **Hauptteil** der Zeile stehen v.a. Konsonantenzeichen, denen ein a inhäriert. Daneben finden sich eigenständige Vokalzeichen am Wortbeginn. Eine problematische Klasse bilden Konsonantenzeichen, die durch ein untergeschriebenes Zeichen vokalisiert werden, das sich aber grafisch nicht vom Hauptzeichen trennen lässt (ein Beispiel ist das erste Zeichen im Bild auf der rechten Seite). Diese Zeichen werden in der aktuellen Programmversion als vollständig zur Hauptzeile gehörig und damit als eigenständige Zeichenklasse behandelt.
- **Über** dem Hauptteil der Zeile findet sich eine Vielzahl von Vokalisierungen und die häufigen Laute r und Anusvara.
- **Unter** der Zeile treten mehrere Vokalisierungen und - v.a. in älteren Devanagari-Drucken - auch Konsonantenzeichen (z.B. *ra*, *va*) auf, die eigentlich zu Ligaturen der Hauptzeile gehören, aus drucktechnischen Gründen aber tiefer gesetzt wurden.

Nach der vergleichsweise einfachen Vorbehandlung des Graustufenbilds ist die Segmentierung der Zeilen die erste Aufgabe, für die es in bisherigen OCR-Programmen keine Vorbilder gibt.
Der naheliegendste Ansatz, die über- und untergeschriebenen Zeichen in den Erkennungsprozess einzuschliessen, hätte darin bestanden, die Vokalisierung in einer vertikalen Projektion in das Hauptzeichen einzubeziehen. Aus rechentechnischen Gründen scheint dieser Weg allerdings nicht angemessen: Geht man konservativ von einer Zahl von ca. 200 wichtigen Zeichen in der Hauptzeile und ca. 25 über- und untergeschriebenen Vokalisierungen aus, kommt man auf eine Zahl von 5000 zu trainierenden Zeichenkombinationen. Es muss also ein analytischer Weg mithilfe einer Zeilensegmentierung beschritten werden.
Ein weiteres Problem bei der Segmentierung ist ein horizontaler Deckstrich, der in den meisten Devanagari-Druckvorlagen einzelne oder alle Zeichen eines Wortes verbindet. Stellt schon bei lateinischer Schrift die Separierung benachbarter Zeichen ein grosses Problem dar, versagen Algorithmen, die allein auf das Auffinden eines vertikalen Helligkeitsminimums ausgerichtet sind, meist vollständig bei der Auftrennung zusammengeschriebener Wörter in der Devanagari.

Auffinden zusammengehöriger Regionen

Das aus dem ersten Arbeitsschritt resultierende SW-Bild wird basierend auf einer Vierer-Umgebung in ein Randbild umgewandelt. Der Vorgang ist im nebenstehenden Bild angedeutet: Das links abgebildete massive Objekt wird auf



die Pixel reduziert, in deren Vierer-Umgebung sich mindestens ein Hintergrundpixel (weiss) befindet. – Die resultierenden schwarzen Pixel im Randbild (rechtes Objekt) werden auf Basis des Freeman-Chaincodes zu Regionen zusammengefasst. In einer Region vorhandene Löcher werden in der aktuellen Programmversion nicht untersucht und deshalb aus dem Chaincode gelöscht.

Zusammenfassen der Regionen zu Zeilen

Die vom Benutzer markierten Textblöcke werden horizontal gescannt und anhand der relativen Helligkeitswerte zu Zeilenbereichen zusammengefasst. Bei diesem Arbeitsschritt ist der schon erwähnte horizontale Deckstrich der Devanagari eine gute Orientierung, da periodisch auftretende Schwarzmaxima den Abschluss der jeweiligen Hauptzeilen kennzeichnen. Für jede der im ersten Schritt gefundenen Regionen wird die maximale Schnittfläche mit einem der gefundenen Zeilenbereiche ermittelt. Die Region wird dann dem entsprechenden Zeilenbereich zugeordnet.

Segmentierung der einzelnen Zeilen

Bei der Segmentierung der Devanagari besteht das Problem, dass über- und untergeschriebene Zeichen grafisch mit der Hauptzeile verknüpft sein können. Das Programm versucht, über einen horizontalen Scan ein im oberen Bereich der jeweiligen Region befindliches Schwarzmaximum (= Deckstrich) zu finden und so übergeschriebene Zeichen abzutrennen. Ein ähnliches Verfahren (gemittelter unterer Abschluss jeder Region in einer Zeile) wird verwendet, um untergeschriebene Zeichen zu entdecken.

Eine weitere Schwierigkeit besteht in der Trennung einzelner Zeichen, die möglicherweise zu einer Region zusammengefasst wurden. Hier kommen zwei Methoden zum Einsatz. Zunächst werden mögliche Trennungspunkte mit einem vertikalen Helligkeitsscan gesucht. Die gefundenen Punkte werden danach selektiert, dass vertikal nur ein Wechsel von Schwarz nach Weiss auftritt, der im oberen Bereich der Zeile, also nahe am Deckstrich liegen muss. Allerdings reichen viele Zeichen der Devanagari - z.B. wenn sie mit einem nicht abtrennbaren untergeschriebenen Zeichen verbunden sind - deutlich über ihren eigentlichen rechten Rand hinaus. Hier liegt also ein senkrechter Trennungsbereich vor, der aber nicht lotrecht nach unten führt und so für den Algorithmus nicht zu erkennen ist (vgl. das Zeichen *bhū* in der nebenstehenden Abbildung, dessen Vokalisierung deutlich über den rechten Rand des *bha* hinausragt). In diesem Fall wird innerhalb bestimmter horizontaler Grenzen ein Pfad vom unteren Rand der Region bis hin zum Deckstrich gesucht, der nur über weisse Pixel führt (“toleranter Trennungspfad”). Ist solch ein Pfad vorhanden, wird die Region entlang dieses Pfades in zwei Hauptzeichen aufgeteilt.

toleranter Trennungspfad

Fehlerkorrektur

Im nächsten Schritt sucht das Programm ungewöhnlich breite Regionen, die vermutlich aus nicht-segmentierten Einzelzeichen bestehen. Diese Regionen werden nochmals mit strengeren Parametern segmentiert.

Zusammenfassung der Regionen zu Wörtern

In einem letzten Schritt werden die Regionssegmente zu Wörtern zusammengefasst. Dabei definiert das Programm weisse Bereiche innerhalb einer Zeile, deren Seitenverhältnis einen benutzerdefinierten Grenzwert überschreitet, als Freizeichen.

Die Fehlerrate dieses Algorithmus für Devanagari liegt bei unter 1%, wenn es sich um eine gute Druckqualität mit gerade ausgerichteten Zeilen handelt. Das Ergebnis kann durch eine höhere optische Bildauflösung oder mögliche Helligkeitskorrekturen im Scanprogramm verbessert werden.

Probleme treten in der aktuellen Version v.a. in der Nähe der Mittelfalz eines eingescannten Buches auf. An dieser Stelle werden die Zeilen oft so stark gekrümmt, dass der horizontal scannende Algorithmus den Deckstrich der Nagari nicht erkennt und damit den Zeilenbereich falsch segmentiert. Auch die oft schlechte Druckqualität der Textvorlagen wirkt sich negativ auf die Segmentierungsergebnisse aus. Die Verbesserung der Segmentierung ist damit eine der wichtigsten Aufgaben für eine Folgeversion des Programms. Erste experimentelle Resultate unter Einsatz von perzeptualer Gruppierung zeigen besonders bei gekrümmten Zeilen sehr erfolgversprechende Ergebnisse.

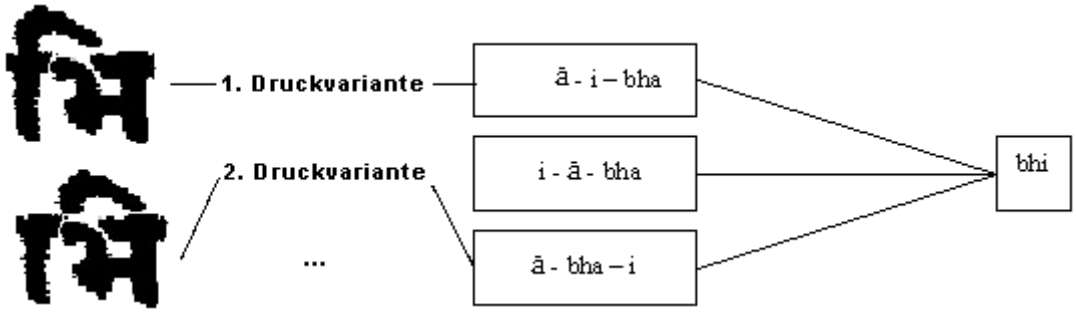
Merkmalsextraktion und Zeichenerkennung

Probleme
Bei der Zeichenerkennung von Devanagari ergibt sich eine Reihe von Problemen, die grösstenteils in Systemen für die lateinische Schrift nicht auftreten:

- **Anzahl der Zeichenklassen**
  Die Devanagari besitzt allein in der Hauptzeile über 500 verschiedene Zeichenklassen (ein vorläufiger Wert, der nach oben hin noch offen zu sein scheint!). Hinzu kommen ca. 30 Klassen von über- und ca. 10 Klassen von untergeschriebenen Zeichen. Für die Anwendung ergeben sich folgende Konsequenzen:
  1. ein grösserer Umfang der Zeichen-Datenbank, der v.a. bei Anwendung linearer Suchmethoden in einer mit fortschreitendem Training längeren Erkennungszeit resultiert.
  2. eine deutlich erhöhte Trainingszeit, bis akzeptable Ergebnisse erzielt werden. Diese Aussage gilt natürlich auch für die Klassenbildung mithilfe neuronaler Netze.
  3. ein erhöhter Speicherbedarf sowohl im Arbeitsspeicher als auch bei Speicherung auf einem Datenträger.
- **Grafische Komplexität der Zeichen**
  Der grafische Aufbau von Devanagari-Zeichen ist komplexer als der von Zeichen

der lateinischen Schrift. Während viele Zeichen des lateinischen Alphabets in einer durchgehenden Linie nachgezeichnet werden können, ist fast jedes Zeichen der Devanagari - jedenfalls in Druckversionen - aus mehreren unabhängigen Teillinien aufgebaut. Viele Ligaturen unterscheiden sich nur durch geringfügige und leicht zu übersehende Zusatzlinien (vgl. die Schwierigkeiten von Systemen für lateinische Schrift bei der Unterscheidung von *e* und *c*). Schliesslich nehmen Devanagari-Zeichen generell einen grösseren Anteil der sie umschliessenden bounding box ein. Grafische Differenzierungen auf Basis lokaler Schwarzverteilungen sind so schwieriger zu implementieren.

- **Statistische Information über die Verteilung der Zeichen**
  Als zentrales Element vieler OCR-Systeme werden statistische Informationen über das Auftreten von Zeichen und v.a. von bestimmten Zeichenkombinationen eingesetzt. Diese Informationen werden zur Entscheidung über die richtige Klassifizierung eines nicht sicher erkannten Zeichens benutzt.
  Ein einfaches Beispiel aus dem Deutschen: Wenn ein System hier an zwei aufeinander folgenden Positionen die jeweils leicht verwechselbaren Lautpaare e/c und y/g erkennt, entscheidet es sich mit grosser Sicherheit für die Endkombination *eg*, da *ey*,*cy* und *cg* in deutschsprachigen Texten selten / überhaupt nicht auftreten. Diese Art statistischer Information würde auch bei der Erkennung von Devanagari beste Dienste leisten. Für eine reine Lautschrift wie z.B. das Deutsche sind diese Informationen über mögliche Zeichenkombinationen leicht aus digitalisierten Texten extrahierbar, da die Anordnung der in der Textvorlage gedruckten Laute derjenigen der Laute im digitalisierten String vollständig entspricht und man damit zur Schätzung der nötigen Parameter auf umfangreiche Sammlungen schon digitalisierter deutscher Texte zurückgreifen kann (aus denen dann z.B. erschlossen wird, dass die Kombinationen cg, cy, ... nicht auftreten). Diese Voraussetzung trifft, wie die folgende Grafik illustriert, für das Verhältnis von gedruckter Devanagari und ihrer digitalisierten Umschrift nicht zu.



Das Bild zeigt die Zeichenfolge, die in der Umschrift den Lautwert *bhi* besitzt. Durch geringe Verschiebungen im Druck der Zeichen kann das übergeschriebene *i*, das die Vokalisierung des *bha* in der Hauptzeile darstellt, nach rechts rutschen, so dass schon für diese einfache Silbe drei verschiedene Folgen von grafischen Elementen existieren, aus denen sie zusammengesetzt sein kann (vgl. Kästen in der Mitte der Grafik). Wird jetzt noch berücksichtigt, dass eine Ligatur in den verschiedensten Zusammensetzungen aus Einzelzeichen auftreten kann, ist leicht erkennbar, dass aus einem digitalisierten Sanskrittext weder die Reihenfolge noch der eigentliche Lautwert der zugrunde liegenden Zeichen problemlos abgeleitet werden kann. (Zum Lautwert: Das Wortfragment -brahma- tritt in älteren Drucken z.B. als -bra-hma- oder -ba-ra (untergeschrieben)-hma- auf). Aufgrund der geschilderten Probleme arbeitet das Programm momentan ohne statistische Information, was in manchen Passagen die Erkennungsqualität herabsetzt. Der Möglichkeiten, statistische Merkmale zu extrahieren, werden

intensiv untersucht; sie werden (hoffentlich) eine deutliche Erhöhung der Erkennungsraten in künftigen Programmversionen bescheren.

Implementierung

Merkmalsextraktion
Wie beschrieben wird in diesem Programmschritt jede der gefundenen Regionen, die im Idealfall den einzelnen Zeichen der Textvorlage entsprechen, in eine Folge numerischer Werte umgewandelt. Die folgende Aufzählung umfasst nur die wichtigsten der getesteten Algorithmen. Alle dargestellten Algorithmen geben einen float-Vektor fester Grösse zurück, der per Absolutwert der einfachen minimalen Distanz mit gespeicherten Zeichen verglichen oder als Eingabe an ein KNN angelegt wurde.

- **Zeichenprofile**
  Jedes Zeichen wird auf eine festgesetzte Dimension skaliert. In diesem skalierten Bild werden zunächst zeilen- und spaltenweise die Profile aller vier Seiten, d.h. das Auftreten des ersten schwarzen Pixels, abgetastet (0.0 -> Schwarz beginnt direkt an der fraglichen Seite; 1.0 -> weisse Zeile/Spalte). Dann werden die relativen Schwarzhäufigkeiten für alle Zeilen bzw. Spalten eingetragen. Anschliessend wird das Bild auf seiner senkrechten und vertikalen Achse aufgeschnitten und für jede Achse nach beiden Seiten der jeweils erste Schwarzwert gesucht. Dieser Teil des Algorithmus hat besonders die Erkennungsrate von Devanagari-Zeichen verbessert, deren Profile identisch sind (z.B. va und ba; ba unterscheidet sich nur durch einen Querstrich innerhalb der beiden Zeichen eigenen Ellipse). In einer letzten Stufe wurden die Profile und relativen Schwarzwerte für ein zentriertes Fenster von halber Breite und Höhe des Hauptbildes eingetragen. Die Erkennungsraten bei einem Selbsttest (jedes Zeichen wird anhand des eigenen Zeichensatzes klassifiziert) von vier umfangreichen Devanagarialphabeten liegen über 95%.
- **Randvektoren**
  Das Zeichen wird auf seine Randpixel reduziert (vgl. S. 8). Diese Randpixel werden zu Vektoren zusammengefasst. Anschliessend wird das Zeichen in eine feste Anzahl von rechteckigen Regionen aufgeteilt. Die gefundenen Vektoren werden über die Lage ihres Mittelpunkts einer der Regionen zugeordnet. Im Zielfeld werden Länge und Steigungswinkel des Vektors eingetragen. Die Einträge werden anschliessend normalisiert. Bei alleiniger Anwendung dieser Methode erhält man Resultate, die deutlich unter denen aus Prozedur 1. liegen. Eine analoge Prozedur kann auch für das Skelett des Zeichens durchgeführt werden, allerdings liegen die Erkennungsresultate dann deutlich unter denen für die Randpixel. Eine alternative Möglichkeit, Randvektoren zu verarbeiten, liegt in der Bewertung ihrer geometrischen Ähnlichkeit. Dazu wurde einerseits die minimale Schnittfläche zwischen den Linienzügen zweier Zeichen berechnet, d.h. für jeden Randvektor v1 eines Zeichens wird der Randvektor v2 des Vergleichszeichens gefunden, mit dem v1 eine minimale Fläche einschliesst. Diese Schnittflächen werden für das gesamte Zeichen summiert. Die Prozedur ist langsam, und die Ergebnisse rechtfertigen die Rechenzeit nicht.
  Eine ähnliche Prozedur trägt die Längen und den Drehwinkel aller Vektoren, die das Randpolygon eines Zeichens bilden, in ein Diagramm ein (x-Achse: Länge des jeweiligen Vektors; Skalierung x-Achse: [0,1] entspricht der Gesamtlänge des Polygons; y-Achse: [0,1] entspricht Winkelintervall [0,pi]). Für dieses Diagramm wird ein Deckungsvergleich mit allen Diagrammen des Vergleichszeichens

durchgeführt; Sieger ist das Zeichen, dessen Randdiagramme mit denen des Ausgangszeichens die grösste Fläche gemeinsam haben. Die Prozedur hat den Vorteil, resistent gegen verschiedene Drehrichtungen und Startpunkte der Randpolygone zu sein, da man die Linienzüge per Verschieben der Histogrammwerte schnell drehen kann. Leider entsprach die ansprechende Form der Grundidee nicht den Erwartungen, die sie in Hinsicht auf die Ergebnisse erweckte.

- **Statistische Verteilungen**
  Ein Vektor umfasst in diesem Fall eine Reihe von statistischen Beschreibungen der Schwarzverteilung innerhalb eines Zeichens. Zunächst werden das Seitenverhältnis der bounding box und das globale Schwarz-Weiss-Verhältnis eingetragen. Das Zeichen wird dann in eine feste Anzahl von Regionen aufgeteilt. Für jede der Regionen werden verschiedene statistische Informationen gewonnen, so das lokale Schwarz-Weiss-Verhältnis, die mittlere Lage der x- und y-Werte (Modus, Median, arithmetisches Mittel), ein Moment dritten Grades der x- und y-Werte und / oder die Varianz der Lagewerte von ihrem arithmetischen Mittel. Diese Angaben werden für jedes Fenster auf das Intervall [0,1] skaliert. - Bei alleiniger Anwendung liefert dieser Vektor keine zufriedenstellenden Ergebnisse, er kann aber der Untersuchung umfangreicherer Merkmalssätze vorgeschaltet werden und so die Zahl der möglichen Zeichenklassen reduzieren.
- **Typische Vektoren**
  In einer vorgeschalteten Prozedur werden alle Zeichen des Trainings-Zeichensatzes skelettiert. Aus den Skeletten werden alle Vektoren, deren Länge einen bestimmten Grenzwert überschreitet, extrahiert und auf das Intervall [0,1] skaliert. Die Start- und Endpunkte dieser Vektoren werden auf eine Stelle nach dem Komma gerundet und mit einer ID versehen in eine Tabelle eingetragen. Nun wird in jedem Zeichen des Zeichensatzes das Auftreten der extrahierten Vektoren überprüft, mit einer Masszahl zwischen 0.0 (= "tritt nicht auf") und 1.0 (= "tritt in voller Länge auf") bewertet und am entsprechenden Index des Ergebnisvektors eingetragen. Als Ergebnis erhält man einen Vektor, dessen Länge der Zahl von Vektoren-Mustern entspricht. Als Eingabe an einem KNN liefert dieser Vektor zufriedenstellende Ergebnisse. - Ein Nachteil des Verfahrens ist die mangelnde Vorhersagefähigkeit, wenn in Zeichen, die während der Trainingsphase nicht vorlagen, neue Vektorenausprägungen auftreten. In diesem Fall müssen die zugrunde liegenden Tabellen erneuert und die Erkennungsvektoren neu erstellt werden. Ausserdem lassen sich gute Ergebnisse nur für einen relativ kleinen Längen-Schwellenwert erzielen, so dass der Umfang der Tabelle und damit des Ergebnisvektors schnell eine beachtliche Grösse erreichen kann.
- **Pixeldistanz**
  Bei diesem Ansatz wird die Ähnlichkeit zweier Zeichen nach der Summe der minimalen Distanzen zwischen den Punkten zweier Zeichen bewertet. Diese Methode wurde für das unbearbeitete Zeichen, das Skelett oder die Randpixel getestet. Dabei kamen entweder absolute Pixelkoordinaten in zwei vorher auf dieselbe Grösse gezoomten Zeichen (Vergleich über Dilation mit einem morphologischen Element) oder relative Koordinaten im Bereich [0,1] (euklidische Distanz zum nächsten Wert des Vergleichszeichens) zum Einsatz. Es wurden sowohl Ansätze getestet, die eine eindeutige Korrelation zwischen nächsten Pixeln verlangen (jeder Pixel kann nur genau einem Pixel des Vergleichsbildes zugeordnet sein), als auch Methoden, die es erlauben, einen Punkt mehreren Punkten des Vergleichsbildes zuzuordnen. Obwohl die Ergebnisse für Randpixel relativ gut waren, scheidet dieser Algorithmus durch seine lange Ausführungsdauer aus, die sich auch bei Vorordnung der Pixel nur geringfügig verbesserte. Wahrscheinlich liegt die Lösung in einer eleganteren

Vorstrukturierung des Datenraums (Anmerkung: eine Distanz-Map bewirkt Wunder!!). Eine alternative Prozedur kann sich auf die Start- und Endpunkte der Vektoren in einem vektorisierten Bild beschränken. Obwohl durch die Vektorisierung die Zahl der Vergleichspunkte dramatisch sinkt, überzeugen die Ergebnisse nicht, was unter anderem an den durch Rauschen verursachten unterschiedlichen Ergebnissen der Vektorisierung liegt.

Als bester Merkmalsvektor hat sich eine Codierung der **Konvexität der Zeichen** herausgestellt. Dieser Ansatz beruht auf Überlegungen aus Parker (1997), 310 ff. Wieder wird das Zeichen zuerst auf seine Randpixel reduziert und dann in eine feste Anzahl von Regionen aufgeteilt. Im folgenden wird für jeden Weisspunkt des Zeichens ermittelt, ob er - innerhalb eines Schwellenwerts für die maximale Entfernung - rechts, links, oben und unten einen schwarzen Nachbarpixel besitzt. Je nach Kombination der Trefferwerte wird das entsprechende Feld des Ergebnisvektors inkrementiert, d.h. der Vektor speichert Aussagen über rudimentäre Formen, die im Bild des Zeichens auftreten. Schliesslich wird jeder Wert durch die Summe aller Werte des Vektors geteilt. Der Vorteil dieser Methode liegt in der relativ hohen Ausführungsgeschwindigkeit und vor allem den deutlich höheren Erkennungsraten. - Wird der Algorithmus beim nicht auf die Randpixel reduzierten Zeichen oder seinem Skelett eingesetzt, liefert er deutlich schlechtere Ergebnisse.

Klassifikation

Die Klassifikation der extrahierten Merkmalsvektoren stellt den Kern der Texterkennung dar. Nachdem die die Struktur der wichtigsten hier eingesetzten Klassifikatoren, der Backpropagation-Netze, und die Auswahl möglicher Merkmalsvektoren schon oben skizziert wurden, liegt an dieser Stelle das Hauptaugenmerk auf der Implementierung des Algorithmus.

Für die Klassifikation von Zeichen stehen zwei Strukturen zur Verfügung:

- Ein sogenannter Omnifont, der verschiedene neuronal basierte Klassifikatoren vereinigt. Diese Gruppe von KNN ist fest in den Quellcode integriert und lässt sich nicht auf neue Zeichenklassen oder ungewöhnliche Ausprägungen bekannter Zeichenklassen trainieren. Der Omnifont wurde auf Basis von über 40000 manuell klassifizierten Zeichen aus 16 verschiedenen Drucktypen trainiert (Details auf Seite **Fehler! Textmarke nicht definiert.**).
- Ein trainierbarer Klassifikator, der sich am Prinzip lokaler gausscher Dichtekurven orientiert.

Der Omnifont

Der Omnifont besteht zur Zeit aus vier eigenständigen Klassifikatoren, die sich nach ihrer internen Architektur in drei Klassen unterteilen lassen.

- Klasse 1: Dieser Klassifikator besteht aus einer Gruppe von Perzeptronen, die auf jeweils eine Zeichenklasse trainiert wurden. Er umfasst soviele autonome Netze, wie Zeichenklassen in der Trainingsmenge vorhanden sind. Als Ausgabe stehen pro Klassifikator nur zwei Neuronen zur Verfügung: Wird das erst aktiviert, gehört das Zeichen zu der Klasse, die der Klassifikator repräsentiert, und der Suchprozess wird abgebrochen. Wird das zweite Ausgabeneuron aktiviert, gehört das Zeichen nicht zu dieser Klasse. Die Erkennungsleistung einzelner Klassifikatoren lässt gelegentlich etwas zu wünschen übrig. Allerdings ist die Verlässlichkeit der einzelnen Netze sehr hoch, d.h., wenn ein KNN auf ein zur Klassifizierung vorgelegtes Zeichen reagiert, gehört dieses Zeichen mit grosser

Sicherheit auch zu der vom KNN repräsentierten Klasse. Als Eingabe erwartet dieser Klassifikator den Konvexitäts-Vektor.
- Klasse 2: In dieser Klasse finden sich zwei Perzeptronen, die mit dem Konvexitäts-Vektor und einem auf 12x12 Pixel skalierten Template der Zeichen trainiert wurden. Zur Erhöhung der Klassifikationsleistung wird während der Testphase, die sich an das Training anschliesst, die Ausgabe des jeweiligen Gewinner-Neurons aufgezeichnet und anhand dieser Daten ein Schwellenwert für korrekte Klassifikationen errechnet. Wird während der Texterkennung dieser Wert im Gewinner-Neuron nicht zumindest erreicht, lehnt der Klassifikator das vorgelegte Zeichen als unbekannt ab.
- Klasse 3: Diesen Klassifikator, der ebenfalls mit dem Konvexitäts-Vektor trainiert wird, kann man als Rudiment eines Radialen-Basisfunktions-Netzes (RBF) ansehen, das mit einem einfachen Cluster-Algorithmus trainiert wurde. Von einem RBF sind nur die Basisfunktionen, nicht aber das nachgeschaltete einstufige Netz übernommen. Zur Ausführungszeit bestimmt der Klassifikator anhand der Eingabe die jeweils nächste Zeichenklasse mit und ohne Einbeziehen der Standardabweichung für jede radiale Basisfunktion. Stimmen beide Zeichenklassen überein, gibt der Klassifikator diese Klasse als Erkennungsresultat zurück.

Nachdem während der Erkennungsphase ein Zeichen nacheinander an die vier Klassifikatoren des Omnifonts angelegt worden ist, können je nach Aktivierung der einzelnen Klassifikatoren unterschiedliche Resultate ausgegeben werden. Einige Extremfälle als Beispiele:
- Kein Klassifikator wurde durch den Zeichenvektor aktiviert. Das System hat auf Basis des Omnifonts keine Vorstellung über die Transkription des Zeichens.
- Alle Klassifikatoren wurden aktiviert, und alle geben dasselbe Resultat aus. – Der Idealzustand, eine einstimmige Klassifizierung eines Zeichens, die allerdings nicht der korrekten Klassifizierung entsprechen muss. Fehlklassifikationen auf Basis eines einstimmigen Votums der Omnifont-Klassifikatoren sind übrigens besonders schwierig im Postprocessing zu erkennen und zu beseitigen.
- Alle Klassifikatoren wurden aktiviert, und jeder gibt ein anderes Resultat aus.

Allgemein ausgedrückt muss das Programm das Votum von vier Klassifikatoren zu einem Endresultat verschmelzen und dann die Korrektheit dieser Klassifizierung bewerten. Für diese Aufgabe existiert eine Reihe von Algorithmen (vgl. Parker (1997), 330ff.), die im vorgestellten System getestet wurden, deren Resultate aber nicht den Erwartungen entsprachen. Stattdessen wird das Resultat mit folgender einfachen Regel generiert: Wenn mindestens drei der vier Klassifikatoren für dieselbe Identifizierung plädieren, wird diese als Ausgabe angesetzt; andernfalls wird der Benutzer zur Korrektur aufgefordert.

Der trainierbare Klassifikator

Wenn der Omnifont ein vorgelegtes Zeichen nicht klassifizieren kann, tritt ein vom Benutzer trainierbarer dreistufiger Klassifikator in Aktion, der sich am Prinzip der mixture densities orientiert (Paulus (1998), 340ff.). Dieser Klassifikator unterteilt den Datenraum mittels eines dreischichtigen Entscheidungsprozesses, der die Merkmale "Seitenverhältnis", "Profilvektoren" und "Konvexitätsvektoren" vereinigt. Die Komplexität des untersuchten Merkmalsvektors steigt in jeder Schicht an. In jeder Schicht befinden sich soviele Vergleichsvektoren, wie Zeichenklassen im Zeichensatz vorhanden sind.

Es sei n die Dimension des Eingabevektors für eine Schicht, v_i der Wert eines Eingabevektors der untersuchten Klasse an der Position i und k die Anzahl der Eingabevektoren, die zur Erstellung des Vergleichsvektors herangezogen werden

können. Der Vergleichsvektor speichert dann an der Position i den Mittelwert

$$m_i = \frac{\sum_{0}^{k-1} v_{ki}}{k}$$

und die Standardabweichung

$$Si = \frac{1}{k} \sum_{0}^{k-1} (v_{ki} - m_i)^2,$$

d.h. die Elemente zur Erstellung einer lokalen Dichtekurve. Zur Ausführungszeit wird der Zugehörigkeitsgrad z eines Eingabevektors mithilfe der Dichtekurve als

$$Z = \sum_{0}^{n-1} e^{(-0.5(\frac{v_i - M_i}{S_i})^2)}$$

ermittelt. Die Ausgaben werden mithilfe eines Quicksorts absteigend geordnet. Ein festgesetzter, von Schicht zu Schicht abnehmender Prozentsatz der besten Ergebnisse darf den entsprechenden Vergleichsvektor der nächsten Schicht aktivieren, bis in der letzten Schicht ein sicheres Ergebnis oder eine kleine Anzahl von Kandidaten als mögliche Gewinner feststeht. Ist das Ergebnis der letzten Schicht unsicher, wird auf eine Reihe von Originalvektoren zurückgegriffen, deren beste Entsprechung als Klassifizierung angesetzt wird.

Anordnung und Sortierung der erkannten Zeichen

Eine Aufgabe, die bei der Erkennung lateinischer Schriftsätze unbekannt ist, ist die korrekte Anordnung der Zeichen nach der Zeichenerkennung. So wird in der Devanagari die Silbe ”*si*” durch die Zeichenfolge ”*ā*” (Hauptzeile) - ”*sa*” (Hauptzeile) - ”*ī*” (Oberlänge) repräsentiert; es können aber auch die Kombinationen “*ā*” - “*ī*” - ”*sa*”, ”*ī*” - “*ā*” - ”*sa*” vorkommen, oder das übergeschriebene *i* wird auf Basis seines Flächenschwerpunkts sogar einem folgenden oder vorhergehenden Zeichen zugeordnet (vgl. S. 11).

Das Programm nutzt zur korrekten Konstruktion des transkribierten Textes reguläre Ausdrücke. Dazu wird der erkannte Text zunächst in eine geeignete Umschrift umgewandelt. Im folgenden wird er mit einer Reihe von fest eingebauten Regeln überarbeitet. Ein Beispiel für die korrekte Plazierung des übergeschriebenen Lautes „*ī*“:

"(.)ai(.*)"* -> *"\\1ī\\2".*

Die Regel besagt, dass in allen Fällen, in denen zwischen beliebigen Folgen von Lauten - repräsentiert durch den Ausdruck „(.*)“ - die Laute „*a*“ und „*ī*“ auftreten, das konsonanten-inhärente „*a*“ gelöscht und das „*ī*“ an seiner Position belassen und damit als eigentliche Vokalisierung des zugehörigen Konsonanten aufgefasst werden soll.

Diese für die Nagari notwendige Nachbearbeitung wird ausserdem zur Korrektur häufiger Fehlsegmentierungen benutzt. So werden z.B. die Zeichen ”*ga*” und ”*śa*” bei der Zeilensegmentierung in den meisten Fällen getrennt, da sie zeichenintern nur über den Deckstrich zusammenhängen. Es ergeben sich daher Segmentierungen in ”*g*” + ”*ā*” (für ”*ga*”) bzw. ”*ś*” und ”*ā*” für ”*śa*”, die im hier beschriebenen Postprocessing mit einer Regel der Form

„(.)gā(.*)“ -> „\\1ga\\2“*

zu ihrer korrekten Form zusammengeführt werden.

Speicherung und Training der Zeichensätze

Grundsätzlich sollte der Benutzer für jeden neuen Schrifttyp, den er einlesen will, einen neuen Zeichensatz anlegen, da die schrifttyp-übergreifende Erkennungsmethode, der „Omnifont“, in der aktuellen Programmversion noch nicht mit genügender Genauigkeit arbeitet. Ein solcher Zeichensatz wird als separate Datei in einem festgelegten Verzeichnis gespeichert, wodurch die bisher erstellten Zeichensätze aus Listen in der Benutzeroberfläche durch einfache Auswahl aufgerufen werden können.

Im Rahmen des **Trainings** werden dem Benutzer Zeichen vorgelegt,

- deren Erkennungswahrscheinlichkeit unter einem bestimmten Schwellenwert liegt. Dieser Schwellenwert wird für den trainierbaren Klassifikator jedes Zeichensatzes individuell berechnet, und zwar als Modus der Ergebniswerte für richtige Klassifizierungen.
- die im aktuellen Zeichensatz nicht mit genügend Beispielen vertreten sind. Der Schwellenwert liegt hier bei 30 Beispielen.

Eine Korrektur der bounding box eines Zeichens während der Laufzeit, d.h. das Training auf eine korrekte Zeilensegmentierung, ist in der aktuellen Programmversion nicht möglich.

Ein Problem stellt der Umfang der Devanagari-Zeichensätze dar, die erst sicher klassifizieren, wenn für jedes häufig auftretende Zeichen mindestens 15 bis 20 Beispiele vorliegen. Dabei werden mit einem Zeichen u.a. ein BYTE-Feld für seine SW-Stufen und der Erkennungsvektor auf Basis der Konvexität gespeichert. Zur Speicherplatzreduktion werden die float-Werte des Erkennungsvektors, die sich zwischen 0 und 1 bewegen, mit einem Byte im Bereich von 0 bis 250 kodiert, also ca. bei der dritten Nachkommastelle gerundet. Ein Genauigkeitsverlust bei der Texterkennung ist mit diesem Verfahren nicht erkennbar, im Gegenteilt liefert die diskrete Form bei einigen Zeichensätzen sogar bessere Ergebnisse.

Rechtschreibungs- und Grammatikprüfung

Zur Korrektur der Erkennungsergebnisse sollte auf die grafische Zeichenerkennung eine grammatikalische und lexikalische Prüfung der gefundenen Wörter folgen. An dieser Stelle befindet sich der Übergang von der grafischen Behandlung der Devanagari mit all ihren Problemen zur sprachlichen Analyse des zugrundeliegenden

Sanskrit-Textes. Wie bei der Generierung von statistischen Informationen zur Kookkurrenz von Zeichen aus digitalisierten Sanskrit-Texten können auch hier keine endgültigen Resultate vorgelegt, sondern nur Probleme und mögliche Lösungsansätze aufgezeigt werden. Da die Prozeduren zur sprachlichen Analyse von Sanskrit-Texten im nächsten Kapitel behandelt werden, werden an dieser Stelle die Fragestellungen nur kurz angerissen.

Die sprachliche Nachbehandlung erkannter Texte steht vor folgenden Herausforderungen:

a) Das Programm muss ohne eine zeitaufwendige Grammatikprüfung erkennen können, ob eine Lautfolge im Sanskrit überhaupt auftreten kann oder ob diese Lautfolge von vornherein mit möglicherweise erkannten Alternativzeichen korrigiert werden muss.
b) Es muss eine Prozedur implementiert werden, die schnell die grammatikalische bzw. lexikalische Korrektheit einer vorliegenden Zeichenfolge bewertet.
c) Wegen der sprachlichen Vieldeutigkeit des Sanskrits sollte schliesslich auch ein Kriterium dafür gefunden werden, ob eine lautlich und grammatikalisch-lexikalisch korrekte Lautfolge inhaltlich in den erkannten Text passt.

Zu a): Eine schnelle Vorentscheidung über die Richtigkeit einer Lautfolge kann auf Basis schon digitalisierter Texte durchgeführt werden. Hier werden z.B. alle Zweierkombinationen von Lauten gesammelt, die in diesen Texten vorkommen, und in eine Tabelle eingetragen. Treten in einem erkannten Wort Lautfolgen auf, die nicht in der Tabelle enthalten sind, wird es als lautlich falsch markiert und könnte in einer Grammatikprüfung verändert werden. – Schwierigkeiten treten an dieser Stelle v.a. wegen der wenig normierten Orthografie der Sanskrit-Drucke auf. Ein Beispiel ist der häufig auftretende Ersatz des Anusvara durch die entsprechenden Klassennasale. Auch der Sandhi am Wortende kann so ungewöhnliche Lautkombinationen hervorbringen, dass eine korrekt erkannte Lautfolge als falsch markiert wird. Eine grössere Zahl von digitalisierten Texten in normierter Orthografie bietet v.a. im ersten Fall keine Lösung für das Problem, da seltene Schreibungen in diesen Texten gerade nicht auftreten bzw. korrigiert wurden.
Zu b): Grundsätzlich kann die grammatikalische und lexikalische Richtigkeit einer Lautfolge mit dem eingebauten Sanskrit-Analyse-Algorithmus geprüft werden, der im nächsten Kapitel beschrieben wird. Allerdings ist dieser Algorithmus auf orthografisch richtige Texte hin ausgelegt. Ein Beispiel aus einem älteren Druck eines medizinischen Textes: Die Lautfolge „sīsakantu“, vom Programm grafisch völlig korrekt erkannt, wird in der orthografisch normalisierten Form in „sīsakaṃ tu“, d.h. zwei Einzelwörter, aufgelöst. In dieser Form kann der Sanskrit-Analyse-Algorithmus die Wörter korrekt klassifizieren. Die erste Form mit Ersatz des Anusvaras durch den Klassennasal würde dagegen als falsch markiert und möglicherweise modifiziert werden, was offenbar nicht erwünscht ist. Eine Möglichkeit, mit solch abweichenden Schreibweisen umzugehen, ist eine Reihe von orthografischen Ersatz-Regeln, die z.B. probeweise die Lautfolgen „nt“ in „ṃ t“ umwandeln und dann eine neue Grammatikanalyse durchführen. Führte diese neue Analyse zum Erfolg, würde die Lautfolge „sīsakantu“ in zwei Wörter aufgeteilt und diese als korrekt markiert werden. Hier - wie auch bei anderen Ersetzungsoperationen zur Korrektur von OCR-Text - erhebt sich allerdings das Problem der Verarbeitungszeit. Selbst in orthografisch standardisierten Texten tritt eine hohe Zahl von langen (Wortlänge >> 10 Zeichen) Wörtern auf, von denen die meisten auf verschiedene Arten analysiert werden können. Dabei kann schon die Untersuchung eines korrekten Wortes von 15 Zeichen auf Basis des gesamten Wortschatzes mehrere Sekunden in Anspruch nehmen.

Nimmt man die Fehlersuche hinzu, für die im ungünstigsten Fall jedes Zeichen einmal verändert werden muss, wächst die Zahl der von der Grammatikprüfung zu untersuchenden Wortvarianten sehr schnell an. Eine manuelle Fehlerkorrektur ist beim momentanen Stand der grammatikalischen Nachbehandlung in den meisten Fällen schneller und zuverlässiger. Aus diesem Grund ist die Möglichkeit zur Grammatikprüfung in der beigelegten Programmversion deaktiviert.

Einen möglichen Ausweg aus den geschilderten Problemen bieten Algorithmen, die das Durchlaufen eines Baums über Zeichen mit einer möglichst hohen a posteriori-Wahrscheinlichkeit modellieren. Dadurch könnte die bedingte Wahrscheinlichkeit bestimmter Lautkombinationen über ein Hidden-Markov-Modell (HMM; vgl. S. **Fehler! Textmarke nicht definiert.**) in die Nachbehandlung des Textes einbezogen werden.
Bei einem solchen Ansatz könnten in jeder Position einer von Freizeichen eingeschlossenen Zeichenfolge null bis vier Vorschläge für das korrekte Erkennungsergebnis auftreten. Zusätzlich zu den von den Klassifikatoren vorgeschlagenen Zeichen könnten an jeder Position z.B. typische „Verwechslungskandidaten" vorgeschlagen werden, die beim Training des Omnifonts automatisch auf Basis der häufigsten Fehler gespeichert werden. Wenn die Klassifikatoren also für ein „*va*" plädieren, könnte – mit einer geringeren Bewertung – ein „*ba*" als mögliche Korrektur vorgeschlagen werden. Mithilfe eines geeigneten Algorithmus würde dann diese Struktur durchlaufen und die wahrscheinlichste Folge von Zeichen ausgegeben werden, aus denen dann der Ausgabetext generiert wird. Positionen, für die die Klassifikatoren keinen oder nur einen sehr unsicheren Vorschlag geliefert haben, könnten anhand des Basismaterials für das HMM ebenfalls mit Vorschlägen besetzt werden. Ein einfaches Beispiel: Gegeben sei die Folge „*ta*" – „?" – „*ā*". Das Programm sucht in der Datenbank für das HMM nach ähnlichen Sequenzen und findet für die mit dem Fragezeichen markierte Position als häufigste Lösung den Laut „*tha*". Als Ergebnis würde trotz völligen Scheiterns des Erkennungsprozesses an dieser Position die Lautfolge „*tathā*" ausgegeben werden. Verfeinern liesse sich eine solche Einsetzungsmethode noch über eine Rückkopplung zur Zeichenerkennung. Wie dem Sanskrit-Kundigen auffallen wird, tritt mit wohl fast ähnlich hoher Wahrscheinlichkeit an der mit dem Fragezeichen markierten Position der Laut „*ya*" auf (Ergebnis: „*tayā*"). Das Programm könnte hier also aus dem sprachlichen Kontext heraus die Zielmenge der Zeichensuche auf die Zeichen „*tha*" und „*ya*" einschränken. Der Algorithmus zur Zeichenerkennung wird mit diesem nachgelieferten sprachlichen Wissen wiederholt, als Ausgabe diejenige Zeichenklasse gewählt, der das Zeichen in seiner grafischen Gestalt am nächsten liegt.
Diese Art der Rückkopplung, die nachträgliche Eingrenzung des Zielraums der Mustersuche auf Basis von statistisch abgesicherten Vermutungen über den Kontext eines Zeichens, wäre ein erster Schritt hin zu einer Erkennungsmethode, die sich eher am menschlichen Lesen orientiert als die bisher implementierte lineare Erkennung ohne Rücksicht auf das Umfeld eines Zeichens.
Die Implementierung eines solchen Algorithmus scheitert aber am bisher fehlenden Datenmaterial über die Apriori-Wahrscheinlichkeit von Einzelzeichen und Lautkombinationen, da die Wahrscheinlichkeit bestimmter grafischer Elemente und damit die Wahrscheinlichkeit ihrer Kookkurrenz nicht (oder nur mit grossen Schwierigkeiten und sehr ungenau) aus digitalisierten Texten rekonstruiert werden kann (vgl. S. 11). Die Grammatikprüfung bleibt damit ein Teil des Programms, der in künftigen Versionen vollständig überarbeitet werden muss.