

Software Projekt 2: Optical Character Recognition

Projektgruppe: *Corinne Zeugin*
 Priscilla Schneider
 Adrian Schmid

Version: *1.0*

Datum: *15.06.2012*

Inhaltsverzeichnis

1	Einleitung.....	4
1.1	Auftrag.....	4
1.2	Einführung zu diesem Dokument.....	4
1.3	Glossar.....	4
1.4	Verwendete Tools und Technologien.....	4
1.5	Installationsanleitung.....	5
1.5.1	Binaries.....	5
1.5.2	Source-Code (Beispiel Eclipse)	5
1.6	Themenwahl und Abgrenzung des Projektumfangs	5
1.6.1	Motivation.....	5
1.6.2	Projektumfang.....	5
2	Planung.....	6
2.1	Projektplanung.....	6
2.1.1	Aufteilung der Tasks.....	6
2.1.2	Besprechung im Team.....	6
2.1.3	Arbeitsrichtlinie	6
2.2	Tasks.....	6
2.2.1	Gesamtübersicht	6
2.2.2	Bild einlesen	7
2.2.3	Validierungen der gespeicherten Daten	7
2.2.4	Bild Analyse	7
2.2.5	Zeilen Analyse.....	7
2.2.6	Zeichen Analyse.....	7
2.2.7	Analyse von erkannten Zeichen	7
2.2.8	KNN erstellen.....	7
2.2.9	Total Aufwand	8
2.3	Einlesen des Bildes	8
2.4	Künstliches Neuronales Netzwerk.....	8
2.5	Nachbearbeitung.....	8
3	Umsetzung.....	9
3.1	Einlesen der Rastergrafik.....	9
3.1.1	Kontrast erhöhen / ContrastMatrix.....	9
3.1.2	Strategie / Verwendete Design Pattern	9
3.1.3	Trennzeichen / SpecialCharacter.....	10
3.1.4	Zeichentrennung	10
3.1.5	Wörtertrennung	11
3.1.6	Zeilentrennung	12
3.1.7	Underlines entfernen	12
3.2	Künstliches Neuronales Netzwerk.....	13
3.2.1	Topologie.....	13
3.2.2	Input Layer.....	13
3.2.3	Hidden Layer.....	14
3.2.4	Output Layer.....	15
3.2.5	Funktionsweise / Umsetzung	15
3.2.6	Training (Backpropagation)	16
3.2.7	Idee / Quellenangabe	18
3.2.8	Source-Code	18
3.3	Wörterbuch	19
3.3.1	Ansatz	19
3.3.2	Implementation.....	19
3.3.3	Erstellung des Wörterbuches	20

3.4	Benutzeroberfläche	20
3.4.1	Input-Lasche	20
3.4.2	KNN Console-Lasche.....	21
3.4.3	History-Lasche	22
3.4.4	Dictionary-Lasche	22
4	Tests	23
4.1	BitmapParser Tests.....	23
4.1.1	CharacterParserTest	23
4.1.2	WordParserTest.....	23
4.1.3	RowParserTest.....	23
4.1.4	UnderlineRemoverTest	23
4.1.5	ContrastMatrixTest.....	23
4.2	Dictionary Tests	24
4.3	MatrixHelper Tests	24
5	Verbesserungsvorschläge bei Weiterführung des Projektes	25
5.1	Verbesserung Wörterbuch	25
5.2	Unterstützte Zeichen.....	25
5.3	Anzahl der versteckten Neuronen.....	25
5.4	Trennung von Zeichen	25
5.5	Umgang mit grossen Ä, Ö und Ü	25
6	Auswertung	26
6.1	Burn Down Charts / Iterationsabschlüsse	26
6.1.1	1. Iteration, 14.03. – 04.04.2012	26
6.1.2	2. Iteration, 05.04. – 01.05.2012	26
6.1.3	3. Iteration, 02.05 – 23.05.2012	27
6.1.4	4. Iteration, 24.05. – 13.06.2012	28
6.2	Auswertung des Projektes.....	28
6.2.1	Funktionalitätsreview	28
6.2.2	Projektreview	28
Anhang A:	Abbildungsverzeichnis.....	29
Anhang B:	Tabellenverzeichnis	29
Anhang C:	Bibliographie.....	29
Anhang D:	Projektpläne	30
Anhang E:	Vorlesungsunterlagen KNN	30

1 Einleitung

1.1 Auftrag

Im Rahmen des Softwareprojekts 2 an der ZHAW im Frühjahrssemester 2012 der Klasse i10 wird das erworbene Wissen aus dem Bereich Physik, Numerik oder Algorithmen und Datenstrukturen, sowie Softwareentwicklung im Rahmen eines Softwareprojekts angewendet.

1.2 Einführung zu diesem Dokument

Wir haben uns in der Gruppe entschieden, eine Software zum Thema Algorithmen und Datenstrukturen zu entwickeln. Dieses Dokument begleitet unsere Projektarbeit. Das Dokument soll mit unserem Projekt mitwachsen und mitleben. Sowohl Probleme als auch Erfolgserlebnisse während des Projekts sollen hier festgehalten werden.

1.3 Glossar

Abkürzung	Erläuterung
OCR	Optical Character Recognition (Optische Zeichenerkennung)
KNN / NN	Künstliches Neuronales Netz / Neuronales Netz

1.4 Verwendete Tools und Technologien

Tool / Technologie	Verwendung
GitHub	Source-Repository (https://github.com/aschi/ocr)
Eclipse IDE	Entwicklungsumgebung / Debugging
Maven	Build Management Tool
Microsoft Word 2010	Dokumentation
Paint.Net	Grafiken
Whiteboard	Projektplanung, Iterationsplan, Tasksverwaltung
JUnit / JMock	Unit Testing
Skype	Kommunikation zwischen Teammitgliedern
hu.kazocsaba.math.matrix	Package zum Matrizenrechnen in Java
de.jungblut.math	Package zum minimieren unserer Kostenfunktion

1.5 Installationsanleitung

1.5.1 Binaries

Um eine lauffähige Version des abgegebenen Programms zu installieren, muss man einfach den gesamten Inhalt des „bin“ Verzeichnisses des Datenträger in das Zielverzeichnis kopieren.

Gestartet wird das OCR aus dem Zielverzeichnis mit `java -jar ocr.jar`. Die Trainings Demo lässt sich mit `java -jar TrainingDemo.jar` starten.

1.5.2 Source-Code (Beispiel Eclipse)

Um den Source-Code zu installieren, muss man den gesamten Inhalt des „src“ Verzeichnisses des Datenträgers in das Zielverzeichnis kopieren. Danach mit dem Befehl `mvn install` Installieren.

Um den Source-Code in einer Eclipse Umgebung zu betrachten kann man mit `mvn eclipse:eclipse` die benötigten Eclipse Projektfiles erstellen.

1.6 Themenwahl und Abgrenzung des Projektumfangs

1.6.1 Motivation

Wir alle drei interessieren uns für Künstlich Neuronale Netzwerke (zukünftig in diesem Dokument kurz KNN) und deren Logik und Umsetzung, so dass wir entschieden haben, ein solches zu implementieren. Um das Ganze spannend zu machen, kombinieren wir das mit der Herausforderung, Text in Form von Bildern einlesen zu können und diese in textueller Form wieder auszugeben.

1.6.2 Projektumfang

Wir wollen die Entwicklung unseres OCRs grob in folgende 3 Teile unterteilen:

1.6.2.1 Einlesen des Bildes

Es soll ein Bild, welches einen Text enthält eingelesen werden können. Wir begrenzen uns hierbei auf Bilder mit hellem Hintergrund und dunklem Text oder dunklem Hintergrund mit hellem Text. Als erstes wollen wir das Bild Pixel für Pixel einlesen und daraus eine Matrix erstellen, welche für helle beziehungsweise dunkle Pixel entsprechend 1 oder 0 enthält. In dieser Matrix wollen wir nun Schritt für Schritt Zeilen trennen, Underlines ermitteln und entfernen, zeilenweise Wörter auftrennen und schlussendlich die einzelnen Buchstaben isolieren.

1.6.2.2 KNN zur Erkennung von Zeichen

Das Analysieren und Erkennen von Zeichen soll mittels eines KNNs geschehen. Dieses KNN soll uns Zeichen aufgrund einer Repräsentation des zuvor isolierten Buchstabens zurückliefern.

1.6.2.3 Nachbearbeiten des eingelesenen Textes

Nach dem Einlesen des Textes mithilfe unseres KNNs möchten wir die Ausgabe nachbearbeiten um offensichtliche Fehler zu korrigieren.

2 Planung

2.1 Projektplanung

Die Projektplanung erfolgt manuell auf einem Whiteboard. Wir haben uns bewusst gegen ein elektronisches Projektmanagement Tool entschieden, da man so unserer Meinung nach flexibler ist und Erfolge besser nachvollziehen kann.

2.1.1 Aufteilung der Tasks

Die Tasks wurden entsprechend den Fähigkeiten der Teammitglieder aufgeteilt, um die Skills jedes Mitglieds optimal nutzen zu können und keine unnötige Zeit durch Know-How-Aufbau zu verlieren.

2.1.2 Besprechung im Team

Jeden Sonntag bespricht das ganze Team die aktuell laufende Iteration, Schnittstellenprobleme und Probleme oder Lösungsansätze. Lösungen zu potenziellen Problemen können so zusammen im Team erarbeitet werden. Vor jedem Iterationsende wird ein Rückblick im Team über die Iteration gemacht um aus eventuellen Fehlern während der Iteration zu lernen.

2.1.3 Arbeitsrichtlinie

Zu jeder Klasse, welche wichtige und komplexe Funktionen beinhaltet, soll zuerst eine separate Testklasse erstellt werden. Ebenfalls werden alle relevanten Methoden jeder Klasse mittels JavaDoc kommentiert und dokumentiert. Komplexe und aufwändige Methoden werden, sofern sinnvoll, noch weiter kommentiert, um die Wartbarkeit des Codes zu gewährleisten. Bei der Namensgebung unserer Klassen, Packages, Methoden und Variablen werden Java Konventionen eingehalten.

2.2 Tasks

Allgemeine Darstellungsdefinitionen; Wir haben einen Task jeweils in folgender Form aufgestellt:

Nummer des Tasks (beginnt pro User-Story neu)	Beschreibung des Tasks	Aufwandschätzung für den jeweiligen Task
---	------------------------	--

Da diese Darstellung einfach zu verstehen ist, werden nicht bei jeder neuen Task-Sammlung die Überschriften der Tabellen neu beschriftet.

2.2.1 Gesamtübersicht

Nr.1	Gesamtübersicht Design	1
Nr.2	Hochladen von Bilddateien	0.2
Nr.3	Speichern von eingelesenen Texten	0.2
Nr.4	History Lasche mit Übersicht der vergangenen, eingelesenen und analysierten Texte und deren Bilder	0.3
Nr.5	Übersicht der bekannten Wörter	0.5
Nr.6	KNN-Lasche, Übersicht der Arbeitsschritte vom KNN zum analysiertem Bild	0.2

2.2.2 Bild einlesen

Nr.1	Verschiedene Formate erkennen (jpg, bmp, gif, png)	0.2
Nr.2	Im System ablegen (für KNN zur Verfügung stellen)	1

2.2.3 Validierungen der gespeicherten Daten

Nr.1	Einlesen von gespeicherten Daten validieren	1
------	---	---

2.2.4 Bild Analyse

Nr.1	Kontrast anpassen	1
------	-------------------	---

2.2.5 Zeilen Analyse

Nr.1	Trennung von Zeilen	1
Nr.2	Entfernen von Underlines	1
Nr.3	Trennung von Wörtern	1
Nr.4	Trennung von einzelnen Zeichen (Buchstaben, Zahlen, Satzzeichen)	1

2.2.6 Zeichen Analyse

Nr.1	Pixel der Zeichen in Matrix digitalisieren	1
Nr.2	Erkennen von Zeichen mittels KNN	1
Nr.3	Lernen von neuen Fonts	1

2.2.7 Analyse von erkannten Zeichen

Nr.1	Erkennen von Wörter	2
Nr.2	Lernen von neuen Wörter	1

2.2.8 KNN erstellen

Nr.1	Neuronen festlegen (Für Zeichen, wie auch für Wörter)	1
Nr.2	Neuronen implementieren	0.5
Nr.3	Gewichtungsberechnung	1.5
Nr.4	Lernfähig machen	1
Nr.5	Trainingsphase	1.5
Nr.6	Testphase	1.5

2.2.9 Total Aufwand

Total Aufwand	22.6	Tage
=	158.2	Stunden
+Planung, Dokumentierung, Besprechungen, Präsentation	30	Stunden
	188.2	Stunden

2.3 Einlesen des Bildes

Wir wollen die Bilder erst wie beschrieben in eine Matrix umwandeln. Wir werden hierzu eine Klasse erstellen, welche diese Matrix inklusive einigen Bearbeitungsmethoden enthält.

Bei Zeilen, Wort und Zeichentrennung wollen wir den Ansatz verwenden, dass Zeichen, Wörter und Zeilen meist durch „weisse“ Zeilen bzw. Spalten voneinander getrennt sind.

2.4 Künstliches Neuronales Netzwerk

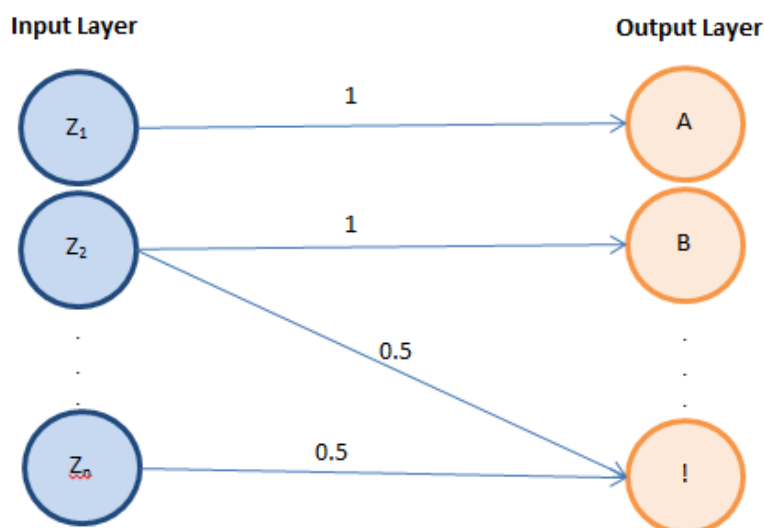


Abbildung 1: Einstufiges Neuronales Netzwerk

Geplant ist ein einfaches einstufiges Neuronales Netzwerk. Unsere Inputneuronen sollen jeweils, wie in Ke Sheng's¹ Artikel beschrieben, aus den zuvor isolierten Zeichenmatrizen erstellt werden. Diese Neuronen sollen dann direkt den Ausgabezeichen zugewiesen werden.

2.5 Nachbearbeitung

Zur Nachbearbeitung wollen wir entweder auch ein Neuronales Netzwerk oder aber eine Wörterbuch Implementation verwenden. Wir wollen uns diesbezüglich noch nicht in der Planungsphase des Projektes festlegen, um in der Umsetzung flexibler zu sein und so schliesslich die geeignetere Methode implementieren zu können.

¹ Ke Sheng: Optical Character Recognition Based on OCRchie, Stand 12.2002,
<http://pages.cs.wisc.edu/~dyer/cs766/hw/hw4/hw4-sheng/sheng.html> (05.04.2012)

3 Umsetzung

3.1 Einlesen der Rastergrafik

Unter „Einlesen der Rastergrafik“ verstehen wir alle Arbeitsschritte welche nötig sind um aus einer Rastergrafik gültige Eingabewerte für unser Neuronales Netzwerk zu erhalten.

3.1.1 Kontrast erhöhen / ContrastMatrix

Als erstes nehmen wir das Bild an sich, gehen Pixel für Pixel durch und entscheiden bei jedem Pixel ob es Hell oder Dunkel ist (0 oder 1). Diese Entscheidung fällen wir aufgrund des durchschnittlichen RGB Wertes.

$$\emptyset RGB \text{ Wert} < 150 \Rightarrow 1 \text{ sonst } 0$$

Die Resultate dieses Vorganges werden Pixel für Pixel in eine von uns eigens dafür entwickelte „ContrastMatrix“ abgefüllt. Eine *ContrastMatrix* enthält ein 2 Dimensionales Array von 0 und 1 Werten und einige hilfreiche Funktionen, welche bei der Weiterverarbeitung verwendet werden.

3.1.2 Strategie / Verwendete Design Pattern

Das Verarbeiten der eingelesenen Bilder, sprich das Trennen von Zeilen, Wörtern und Buchstaben wollten wir möglichst modular aufbauen. Um dies zu erreichen, haben wir das Parsen der Bilder nach dem Decorator Pattern implementiert.

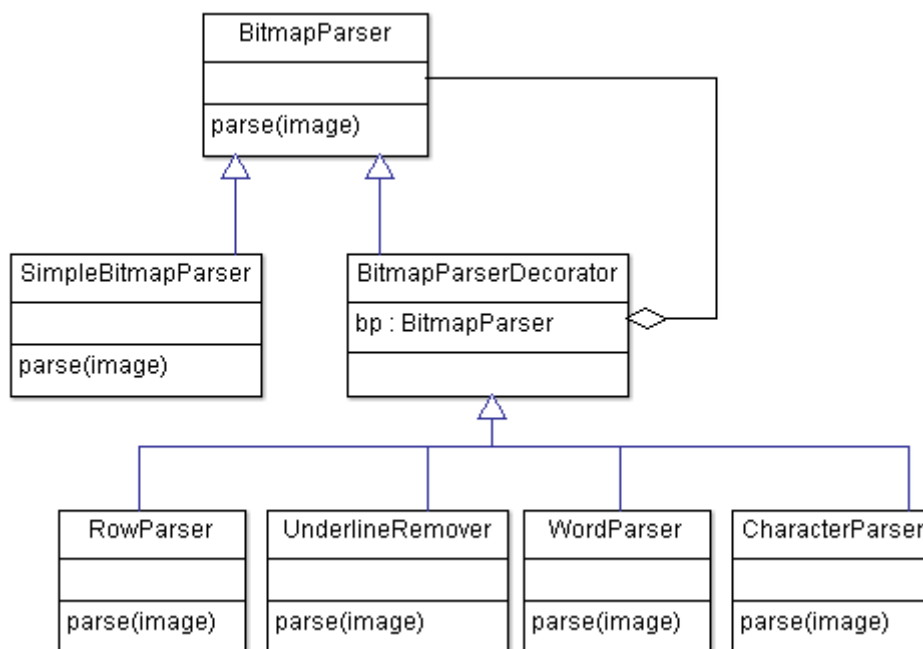


Abbildung 2: DecoratorPattern Pattern Bitmap Parser

Grundsätzlich läuft das einlesen eines Bildes wie folgt ab:

1. Der *SimpleBitmapParser* liest das Bild ein, und erstellt daraus eine *ContrastMatrix*. Diese wird einer Liste hinzugefügt und so zurückgegeben.
2. Der *RowParser* trennt die Zeilen und gibt eine Liste mit einer *ContrastMatrix* pro Zeile zurück
3. Der *UnderlineRemover* geht Zeile für Zeile durch und entfernt alle Underlines. Es wird wiederum eine Liste von Matrizen zurückgegeben. (Immer noch 1 *ContrastMatrix* pro Zeile)

4. Der *WordParser* geht Zeile für Zeile durch und isoliert Wörter. Es wird eine Liste mit einer *ContrastMatrix* pro Wort zurückgegeben
5. Der *CharacterParser* geht Wort für Wort durch und trennt die Buchstaben auf. Als Resultat erhalten wir eine Liste mit *ContrastMatrix* Repräsentationen der einzelnen Buchstaben

Falls nicht benötigt, können einzelne Decorators weggelassen werden. So wird zum Beispiel beim Training des Neuronalen Netzwerkes nur der *CharacterParser* verwendet (Alle Zeichen sind auf einer Zeile und Abstände interessieren uns in diesem Fall nicht).

3.1.3 Trennzeichen / SpecialCharacter

Nach der ersten Implementation unseres BitmapParsers und ersten Versuchen der Texterkennung mittels des entwickelten KNNs mussten wir feststellen, dass bei unserem Verfahren alle Leerzeichen und Zeilenumbrüche verloren gehen. Um dem entgegen zu wirken, haben wir einen Enum „*FunctionalCharacter*“ implementiert und der *ContrastMatrix* 2 Modi verliehen. Sie kann nun entweder wie gehabt einen Teil des eingelesenen Bildes repräsentieren oder aber für ein Trennzeichen stehen.

Beim späteren Verarbeiten der *ContrastMatrix List* wird dies jeweils geprüft und *FunctionalCharacters* werden einfach direkt ausgegeben.

Functional Character	String Äquivalent
FunctionalCharacter.space	" "
FunctionalCharacter.carriageReturn	"\n"

Tabelle 1: Trennzeichen

3.1.4 Zeichentrennung

Beim Entwickeln von Lösungsansätzen zur Trennung von einzelnen Zeichen in einer *ContrastMatrix* sind wir aufgrund optischer Studien auf den Ansatz gestossen, dass Buchstaben üblicherweise durch mindestens eine Spalte weisser Pixel voneinander getrennt sind.

Aus diesem Ansatz haben wir einen einfachen Algorithmus entwickelt, welcher die Bilder Spalte für Spalte durchsieht und sie bei jeder leeren Spalte unterteilt. Dies setzt voraus, dass Zeilen bereits im Voraus getrennt wurden.

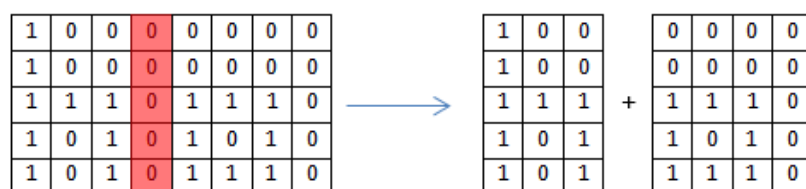


Abbildung 3: Buchstabentrennung

Problem: Je nach Bildquelle und Schriftart sind die Zeichen überlappend. Als Beispiel folgendes Bild. Das T überlappt sich mit dem nachfolgenden e.



Abbildung 4: Problem bei der Buchstabentrennung

Bezüglich dieses Problems haben wir folgende Lösungsansätze erarbeitet:

1. Wir unterstützen nur Texte in Bildformat welche sauber getrennt sind.
2. Wir suchen jeweils alle zusammenhängenden Pixel und erkennen unsere Buchstaben so.



Abbildung 5: Möglicher Lösungsansatz zur Lösung des Buchstaben-Trennproblems

Nach eingehender Analyse des 2. Ansatzes haben wir uns - um unsere Kapazitäten für weitere interessante Aspekte der Texterkennung offen zu halten - entschieden als Voraussetzung einer sauberen Analyse nur Texte in Bildformaten, mit sauber horizontal getrennten Buchstaben zu unterstützen.

Verbesserungsvorschlag bei Weiterführung des Projektes: Siehe [Verbesserungsvorschlag Trennung von Zeichen](#)

3.1.5 Wörtertrennung

Basierend auf der Idee zur Trennung von Zeichen suchen wir auch zum Trennen von Wörtern nach weissen Spalten im Bild. Jedoch müssen wir, zur Unterscheidung der Abstände zwischen einzelnen Zeichen und den Abständen zwischen zwei Wörtern, erst das ganze Bild analysieren.

Dazu gehen wir erst alle Spalten durch und messen jeweils die Breiten der Abstände zwischen Buchstaben. Wir merken uns den kleinsten und den grössten Abstand.

In einem zweiten Schritt gehen wir erneut alle Spalten durch und rechnen bei jeder gefundenen Lücke den Betrag der Differenz der Lückenbreite zur minimalen und zur maximalen Lückenbreite aus.

Definition:

\min = Minimale Abstandsbreite, \max = Maximale Abstandsbreite, cur = Abstandsbreite

Berechnung

$$|cur - \min| > |cur - \max| \Rightarrow \text{Abstand zwischen zwei Wörtern}$$

Die Eingangs *ContrastMatrix* wird folglich bei jedem Abstand zwischen zwei Wörtern aufgetrennt. Um das Leerzeichen nicht zu verlieren, wird nun jeweils zwischen zwei Wörtern ein *SpecialCharacter* (Space) in die Liste der Matrizen eingefügt.

3.1.6 Zeilentrennung

Um Zeilen zu trennen, suchen wir bei jeder der erhaltenen *ContrastMatrizen* nach komplett weissen Zeilen (alle Werte 0) und trennen so die *ContrastMatrix* jeweils bei den leeren Zeilen in neue Matrizen auf. Um die Zeilenumbrüche nicht zu verlieren, wird jeweils zwischen zwei Zeilen ein *SpecialCharacter* (carriageReturn) eingefügt.

Problem: Ein uns bekanntes Problem unserer Art der Zeichentrennung sind grosse Ä, Ö, und Ü. Da die Punkte oberhalb der Buchstaben sind, und somit zwischen den grossen Buchstaben und den Punkten von Ä, Ö und Ü eine leere Zeile vorhanden ist, werden diese abgeschnitten. Will man dieses OCR produktiv nutzen, sollte dies behoben werden (mehr dazu unter [Verbesserungsvorschlag Umlaute](#)).

3.1.7 Underlines entfernen

Wurden die Zeilen getrennt, werden die Underlines entfernt. Hierbei müssen folgende Konstellationen berücksichtigt werden.

Konstellation 1: Die ganze Zeile ist unterstrichen. Die Zeilentrennung trennt dieses Underline also als eigene Zeile ab, da zwischen dem Text und dem Underline eine vollständig leere Zeile vorhanden ist.

Text mit Underline

Abbildung 6: Underlineentfernung Konstellation 1

Konstellation 2: Im unterstrichenen Wort befindet sich ein Buchstabe, welcher sich mit dem Underline überkreuzt. Die Zeilentrennung trennt das Underline nicht vom Text als eigene Zeile, da sich zwischen dem Underline und dem Text keine vollständig leere Zeile befindet.


Text mit Underline von  gemeinen Wörter

Abbildung 7: Underlineentfernung Konstellation 2

Konstellation 3: Es sind nur einzelne Wörter des Textes unterstrichen.

Text mit Underline mit nur einzelnen unterstrichenen Wörtern

Abbildung 8: Underlineentfernung Konstellation 3

Alle drei Konstellationen müssen in der Underline-Entfernung berücksichtigt werden. Um dies zu gewährleisten wird zuerst geprüft, ob die ganze Zeile ein Underline ist (Konstellation 1). Dies ist der Fall, wenn die gesamte Zeile eine oder mehrere, aufeinanderfolgende Spalten enthält die dunkel sind. Ist dies nicht der Fall, wird geprüft, ob in einer oder mehreren aufeinanderfolgenden Spalten dunkle, waagrechte Stellen enthalten sind, die mindestens länger als die durchschnittliche Höhe der Zeichen sind. Ist dies der Fall, werden diese entfernt. Damit hierbei keine dunklen Stellen eines Zeichens mit entfernt werden wie bei Konstellation 2 vom Buchstaben g, wird jeweils geprüft, ob sich unmittelbar oberhalb oder unterhalb des ermittelten Underlines weitere dunkle Stellen befinden. Falls ja, werden während dieser Länge keine dunklen Stellen entfernt. Folgendes Bild soll dies

genauer erläutern. Bei der roten Stelle ist jeweils oberhalb und unterhalb des Underlines dunkel, diese Stellen werden darum erkannt und nicht entfernt.



Abbildung 9: Underlineentfernung bei Überlappung mit einem Zeichen

3.2 Künstliches Neuronales Netzwerk

Bei der Umsetzung dieses Projekt wurde uns klar, dass der primitive einstufige Ansatz, den wir ursprünglich geplant hatten nicht praktikabel ist. Man müsste, damit dieser Ansatz funktioniert, jedes Zeichen in jeder Schriftart / Schriftgrösse trainieren um danach Texte einlesen zu können.

Wir haben dieses Problem mit Jens-Christian Fischer besprochen. Er hat uns vorgeschlagen, ein mehrstufiges Künstliches Neuronales Netz mit mindestens einer versteckten Schicht zu implementieren.

3.2.1 Topologie

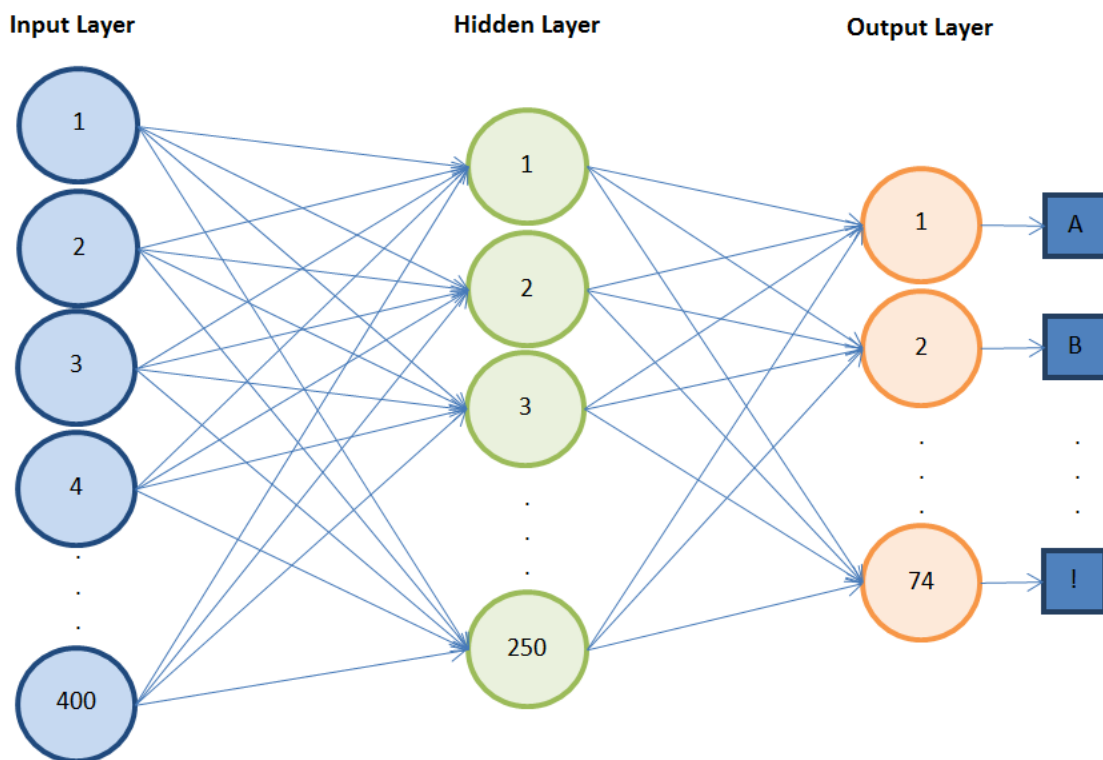


Abbildung 10: Topologie des verwendeten Neuronales Netzwerkes

3.2.2 Input Layer

Der Input Layer eines künstlichen neuronalen Netzwerkes definiert sich durch die Eingangsdaten. In unserem Falle bedeutet dies, dass wir Rastergrafiken von einzelnen Buchstaben als Eingabe für unser neuronales Netz verwenden wollen. Um dies zu bewerkstelligen, legen wir ein 20x20 Raster über jeden Buchstaben und messen in jedem der 400 Rechtecke die Helligkeit. Diese Helligkeitswerte (zwischen 0 = ganz weiss und 1 = ganz schwarz) füllen wir zur Weiterverarbeitung in einen 400-dimensionalen Vektor.

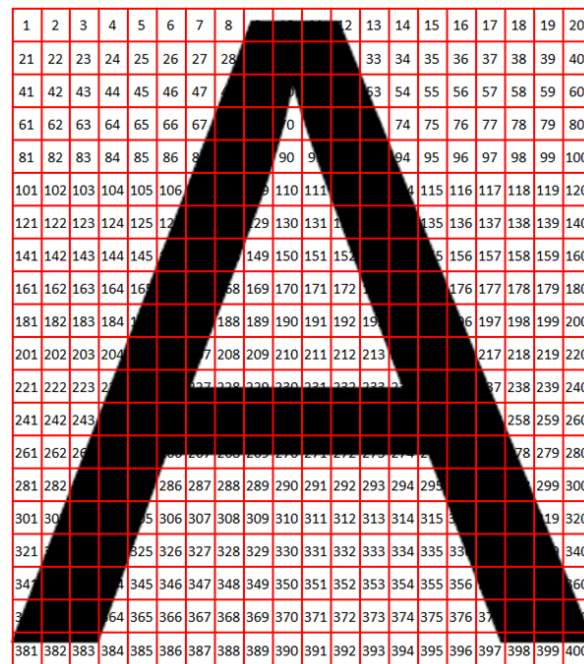


Abbildung 11: Input Layer

3.2.3 Hidden Layer

„There are many rule-of-thumb methods for determining the correct number of neurons to use in the hidden layers, such as the following:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.”²

Aufgrund dieser Ansätze haben wir verschiedene Netzwerk Topologien (mit mehr oder weniger Versteckten Neuronen) ausprobiert und jeweils sowohl das Konvergenzverhalten beim Training der Gewichtungsmatrizen (θ_1 und θ_2) als auch die dafür benötigte Rechenzeit betrachtet.

Anzahl Versteckte Neuronen	Kosten nach 100 Iterationsschritten	Aufwand
100	0.8562266999236073	105s
150	0.5041715500526324	203s
200	0.09241045938241019	417s
250	0.029615303291139018	712s
315 (2/3)	0.05209200008415559	1262s
350	0.010099091296756587	1735s
400	0.017634926417761723	2357s

Tabelle 2: Anzahl versteckter Neuronen / Trainingsverhalten

² Heaton Jeff: The Number of Hidden Layers. Stand: 14.09.2008 <http://www.heatonresearch.com/node/707> (06.06.2012)

Mit 250 versteckten Neuronen haben wir dabei ein gutes Ergebnis bei einem vertretbaren Zeitaufwand beobachten können. Deshalb haben wir uns in diesem Projekt für 250 versteckte Neuronen entschieden.

Verbesserungsvorschlag bei Weiterführung des Projektes: Siehe [Verbesserungsvorschlag Anzahl der versteckten Neuronen](#)

3.2.4 Output Layer

Die Ausgabe Schicht unseres neuronalen Netzwerkes definiert sich durch die Zeichen, welche von dem Netzwerk erkannt werden sollen. Im Rahmen dieses Projektes haben wir uns auf unser Alphabet, Umlaute und einige gängige Satzzeichen beschränkt und sind so auf 74 Outputneuronen gekommen.

Neuron #	Zeichen	Neuron #	Zeichen	Neuron #	Zeichen	Neuron #	Zeichen
1	A	20	T	39	j	58	ü
2	B	21	U	40	k	59	0
3	C	22	V	41	l	60	1
4	D	23	W	42	m	61	2
5	E	24	X	43	n	62	3
6	F	25	Y	44	o	63	4
7	G	26	Z	45	p	64	5
8	H	27	Ä	46	q	65	6
9	I	28	Ö	47	r	66	7
10	J	29	Ü	48	s	67	8
11	K	30	a	49	t	68	9
12	L	31	b	50	u	69	.
13	M	32	c	51	v	70	,
14	N	33	d	52	w	71	:
15	O	34	e	53	x	72	;
16	P	35	f	54	y	73	?
17	Q	36	g	55	z	74	!
18	R	37	h	56	ä		
19	S	38	i	57	ö		

Tabelle 3: Output-Neuronen.

Verbesserungsvorschlag bei Weiterführung des Projektes: Siehe [Verbesserungsvorschlag Unterstützte Zeichen](#)

3.2.5 Funktionsweise / Umsetzung

Die einzelnen Schichten sind in unserer Umsetzung durch Vektoren abgebildet, welche zusammen mit errechneten Gewichtungsmatrizen (θ_1 und θ_2) das neuronale Netzwerk bilden.

Um nun einen Buchstaben zu erkennen, wandeln wir die Rastergrafik-Repräsentation wie im Kapitel „Input-Layer“ beschrieben in einen Input Vektor um. Dieser wird mit der entsprechend optimierten Gewichtungsmatrix θ_1 multipliziert um die Inputwerte der versteckten Schicht zu erhalten. Diese Werte werden durch unsere Sigmoid-Aktivierungsfunktion verarbeitet, um so die Werte der versteckten Neuronen zu erhalten. Diese werden wiederum mit θ_2 multipliziert was - nach erneuter Anwendung der Sigmoid-Funktion – in einem 74-dimensionalen Ausgabevektor resultiert. Bei diesem Vektor ist optimalerweise ein Wert sehr nahe bei 1 und alle anderen Werte sehr nahe bei 0. Den

Index dieses höchsten Wertes suchen wir nun in einer einfachen Zuweisungstabelle um das entsprechende Zeichen zu erhalten.

Definitionen:

$$\text{sig}(t) = \frac{1}{1 + e^{-t}}$$

$$i = \begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ \vdots \\ i_{400} \end{pmatrix}, \theta_1 = \begin{pmatrix} (\theta_1)_{1,1} & \cdots & (\theta_1)_{1,401} \\ \vdots & \ddots & \vdots \\ (\theta_1)_{250,1} & \cdots & (\theta_1)_{250,401} \end{pmatrix}, \theta_2 = \begin{pmatrix} (\theta_2)_{1,1} & \cdots & (\theta_2)_{1,251} \\ \vdots & \ddots & \vdots \\ (\theta_2)_{74,1} & \cdots & (\theta_2)_{74,251} \end{pmatrix}$$

Ablauf:

$$h = \text{sig}\left(\begin{pmatrix} 1 \\ i \end{pmatrix} \cdot \theta_1\right); o = \text{sig}\left(\begin{pmatrix} 1 \\ h \end{pmatrix} \cdot \theta_2\right); \dim(o) = 74$$

$$\text{Index des gesuchten Buchstabens} = \text{index}(\max(o))$$

Damit dies funktioniert müssen die beiden Gewichtungsmatrizen Werte enthalten, welche aus unterstützten Eingaben möglichst optimale Ausgabewerte erzeugen.

3.2.6 Training (Backpropagation)

Um brauchbare Thetamatrizen für ein bestimmtes Problem zu erhalten, ist die Trainingsphase essentiell. Wir trainieren unser Neuronales Netzwerk mit einem Backpropagation Algorithmus welcher auf einer Kostenfunktion beruht.

Grundsätzlich funktioniert Backpropagation wie folgt:

1. Thetamatrizen werden zufällig initialisiert; Eine Liste von Inputvektoren und eine Liste von jeweils dazugehörenden soll-output Vektoren werden vorbereitet (Trainingsset)
2. Inputvektor #1 wird aus der Liste ausgelesen
3. Aus dem Inputvektor wird mithilfe des Neuronalen Netzwerkes ein Outputvektor berechnet (Siehe Abschnitt „Funktionsweise“)
4. Der errechnete Outputvektor wird mit dem entsprechenden soll-Output verglichen und ein Delta wird berechnet
5. Dieses Delta wird durch das Neuronale Netz zurückgerechnet um die nötigen Änderungen der Gewichtungsmatrizen zu erhalten
6. Schritte 3-5 werden nun mit allen Einträgen des Trainingssets durchgeführt. Die Deltas der Theta Matrizen werden aufsummiert und schlussendlich durch die Anzahl Einträge des Trainingssets geteilt
7. Dieser Vorgang wird so oft wiederholt, bis die Deltas minimal sind

Implementiert haben wir das Ganze mithilfe einer Kostenfunktion $[J, \theta_{\text{grad}}] = J(\theta, x, y)$ welche uns sowohl die Theta Gradienten als auch die Kosten (als Zahl) zurückgibt. Berechnet wird das Ganze wie folgt:

Definitionen:

$$x = \text{InputList}, y = \text{OutputList}, m = \text{size}(x), K = \dim(y^{(0)}) = 74$$

$$h_{\theta}(x) = \text{sig} \left(\left[\text{sig} \left(\begin{bmatrix} 1 \\ x \end{bmatrix} \cdot \theta_1 \right) \cdot \theta_2 \right] \right) \text{ (Berechnung des Outputs des neuronalen Netzwerkes)}$$

$$i_1 = \text{Input der ersten Schicht}, i_2 = \text{Input der zweiten Schicht}$$

Berechnungen:

$$J(\theta, x, y) = \left(\frac{1}{m} \right) \cdot \sum_{i=1}^m \sum_{k=1}^K \left[-y_k^{(i)} \cdot \log((h_{\theta}(x^{(i)}))_k) - (1 - y_k^{(i)}) \cdot \log(1 - (h_{\theta}(x^{(i)}))_k) \right]$$

$$\Delta_3(x, y) = h_{\theta}(x) - y; \Delta_2 = \theta_2^T \cdot \Delta_3(x)$$

$$\theta_{1_{grad}}(\theta, x, y) = \left(\frac{1}{m} \right) \cdot \sum_{i=1}^m (\Delta_2(x^{(i)}) \cdot i_1)$$

$$\theta_{2_{grad}}(\theta, x, y) = \left(\frac{1}{m} \right) \cdot \sum_{i=1}^m (\Delta_3(x^{(i)}) \cdot i_2^T)$$

Implementierung:

Als erstes werden alle Bilder aus dem Ordner für Trainingsmaterial (Default: *res/knnMaterial*) eingelesen. Die Buchstaben dieser Files werden getrennt und zu entsprechenden Inputvektoren verarbeitet. Wichtig ist, dass alle Files dieselben Zeichen in derselben Reihenfolge beinhalten. Diese Inputvektoren werden nun unserer InputList hinzugefügt.

Eine eigentliche „OutputList“ brauchen wir nicht. Wir nehmen hierzu einfach eine Einheitsmatrix, lesen die Zeile mit Index des gerade verarbeiteten Inputs aus und transponieren diese um den Soll-Outputvektor zu erhalten.

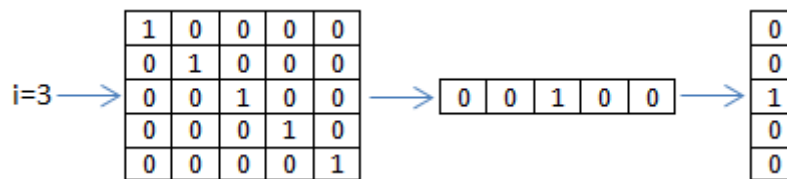


Abbildung 12: Generierung eines Soll-Outputvektors

Als nächstes generieren wir zufällige Gewichtungsmatrizen und bringen diese für die Weiterverarbeitung wie folgt in eine Vektorform:

$$\theta_1 = \begin{pmatrix} (\theta_1)_{1,1} & \cdots & (\theta_1)_{1,401} \\ \vdots & \ddots & \vdots \\ (\theta_1)_{250,1} & \cdots & (\theta_1)_{250,401} \end{pmatrix}, \theta_2 = \begin{pmatrix} (\theta_2)_{1,1} & \cdots & (\theta_2)_{1,251} \\ \vdots & \ddots & \vdots \\ (\theta_2)_{74,1} & \cdots & (\theta_2)_{74,251} \end{pmatrix}$$

wird zu $\theta = \begin{pmatrix} (\theta_1)_{1,1} \\ \vdots \\ (\theta_1)_{250,1} \\ (\theta_1)_{2,1} \\ \vdots \\ (\theta_1)_{250,401} \\ (\theta_2)_{1,1} \\ \vdots \\ (\theta_2)_{74,251} \end{pmatrix}$

Zuerst wird θ_1 Spalte für Spalte abgefüllt, danach machen wir dasselbe mit θ_2 . Wir verwenden diese Form sowohl als Eingabe- als auch als Rückgabeparameter unserer Kostenfunktion.

Als letzter und finaler Schritt minimieren wir - mithilfe der Java Fmincg Implementation von Thomas Jungblut³ - iterativ unsere Kostenfunktion. Fmincg sucht, in Funktionen welche Gradienten von sich selbst und einen Wert zurückgeben, lokale Minima.

3.2.7 Idee / Quellenangabe

Unsere Implementation des künstlichen Neuronalen Netzes basiert auf Vorlesungsunterlagen der Stanford University welche wir von Jens-Christian Fischer erhielten. Insbesondere haben wir die Mathematischen Komponenten (Kostenfunktion, Aktivierungsfunktion, Berechnung der Gradienten) praktisch 1:1 diesen Unterlagen entnommen.

Sie finden diese Unterlagen im Anhang E.

3.2.8 Source-Code

Konkret im Code abgebildet haben wir das ganze wie folgt:

Klasse	Inhalt
ch.zhaw.ocr.nn.NeuralNetwork.java	Zeichen erkennen, Thetas aus File laden, in File schreiben
ch.zhaw.ocr.nn.BackPropagation.java	Kostenfunktion, Zufälliges generieren von Theta Matrizen
ch.zhaw.ocr.nn.NeuralNetworkTraining.java	Einlesen der Bilder aus dem Ressourcen-Ordner, Generieren der InputList, Zusammenstellen der Parameter der Fmincg Funktion, Aufruf der Fmincg Funktion zurückschreiben der Thetas in das zu trainierende „NeuralNetwork“

³ Thomas Jungblut: Fmincg.java, Stand 06.06.2012 <https://github.com/thomasjungblut/thomasjungblut-common/blob/master/src/de/jungblut/math/minimize/Fmincg.java> (08.06.2012)

ch.zhaw.ocr.nn.helper.MatrixHelper.java	Diverse Matrizenfunktionen (u.a Sigmoid, Log, schreiben einer Matrix in ein Textfile, lesen einer Matrix aus einem Textfile, Thetas in einen Vektor umwandeln & Rücktransformation, ...)
ch.zhaw.ocr.nn.CharacterRepresentation.java	Rastern eines Bildes zur Erstellung eines Input-Vektors
ch.zhaw.ocr.Properties.java	Topologieeinstellungen, Zuweisungstabelle für die Output-Schicht, Speicherungsart der Theta Matrizen, Pfad zum Ressourcen-Ordner

Tabelle 4: KNN im Quellcode

3.3 Wörterbuch

3.3.1 Ansatz

Bei der Besprechung der zweiten Iteration hat uns Jens-Christian Fischer auf eine statistische Analyse zur Fehlerkorrektur von Wörtern aufmerksam gemacht. Folgende Ansätze hat er uns vorgeschlagen:

- Wörterbuch anlegen
- Levenstein Distanz, Wörter mit kleinster Abweichung suchen
- Trigramm Analyse (Korrekten Satz erkennen)

Nach eingehender Recherche haben wir uns entschieden, ein Wörterbuch - ähnlich dem von Peter Norvig vorgestellten⁴ - zu implementieren. Ein Wort wird mit den Einträgen des Wörterbuches verglichen. Kennt das Wörterbuch dieses Wort nicht, wird geprüft, welche ähnlichen Wörter bekannt sind um das analysierte Wort je nach dem entsprechend anzupassen.

3.3.2 Implementation

Nach eingehender Analyse haben wir ein Wörterbuch implementiert das Wörter zusammen mit einer Gewichtung anhand der Häufigkeit des Auftretens des Wortes enthält. Bei Erkennung eines Wortes durch das KNN wird dieses nun wie folgt mit dem Wörterbuch abgeglichen und wo nötig verbessert:

1. Es wird geprüft, ob das Wort im Wörterbuch enthalten ist.
→ Wenn ja, wird das Wort so beibehalten
→ Wenn nicht, weiter bei 2.
2. Gibt es Wörter welche nur einen Buchstaben Abweichung haben, wird das Wort mit der grössten Häufigkeit zurückgegeben
→ Gibt es kein Wort mit nur einem Buchstaben Abweichung, weiter bei 3.
3. Gibt es Wörter, welche zwei Buchstaben Abweichung haben, wird das mit der grössten Häufigkeit zurückgegeben
→ Gibt es kein Wort mit zwei Buchstaben Abweichung, wird das Originalwort, welches nicht im Wörterbuch enthalten ist zurückgegeben.

Verbesserungsvorschlag bei Weiterführung des Projektes: Siehe [Verbesserungsvorschlag Wörterbuch](#)

⁴ Peter Norvig: How to Write a Spelling Corrector, <http://norvig.com/spell-correct.html> (09.06.2012)

3.3.3 Erstellung des Wörterbuches

Um das Wörterbuch mit den gewichteten Wörtern zu erstellen, haben wir einige Textfiles zusammengestellt welche - mehr oder weniger repräsentativ - die deutsche Sprache abbilden sollen. Beim Erstellen des Wörterbuches werden diese Textfiles eingelesen, nach Wörtern aufgetrennt und jedes der enthaltenen Wörter wird wie folgt dem Wörterbuch hinzugefügt:

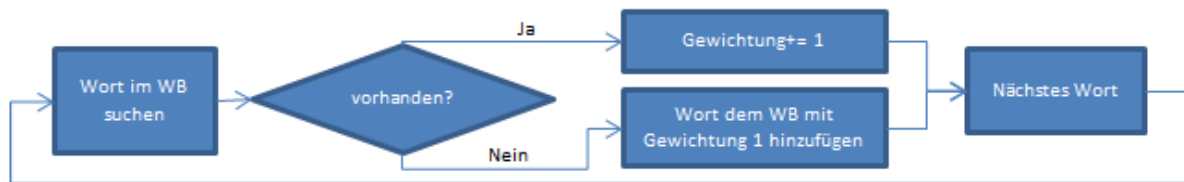


Abbildung 13: Wörter zum Wörterbuch hinzufügen

Quellenangabe der verwendeten Texte:

Quelle	Beschreibung
http://sourceforge.net/projects/germandict/	Umfangreiches, Deutsches Wörterbuch
http://wortschatz.uni-leipzig.de/Papers/top10000de.txt	Die 10'000 meist gebrauchten, deutschen Wörter
http://www.gutenberg.org/ebooks/19163	Märchen für Kinder von H. C. Andersen
http://www.gutenberg.org/ebooks/27891	Die moderne Ehe von Maud Churton Braby
http://www.gutenberg.org/ebooks/14075	Die Frauenfrage von Lily Braun
http://www.gutenberg.org/ebooks/23228	Arabische Nächte von Hans Bethge
http://www.gutenberg.org/ebooks/4505	Die drei Nüsse von Clemens Brentano
http://www.gutenberg.org/ebooks/27220	Himmelsvolk von Waldemar Bonsels
http://www.gutenberg.org/ebooks/35636	Narzissmus als Doppelrichtung von Lou Andreas-Salomé
http://www.gutenberg.org/ebooks/12921	Gesammelte Werke in fünf Bänden — 1. Band von Bjørnstjerne Bjørnson
http://www.gutenberg.org/ebooks/16264	Deutsches Leben der Gegenwart von Bekker, Briefs, Scheler, Sommerfeld, und Witkop
http://www.gutenberg.org/ebooks/26261	Die Frau von dreißig Jahren von Honoré de Balzac

Tabelle 5: Input des Wörterbuchs

3.4 Benutzeroberfläche

Die Idee unserer Benutzeroberfläche ist nicht nur die Bedienung der Applikation, sondern auch ein Schaufenster des Hintergrundes. Aus diesem Grund haben wir uns für vier Laschen entschieden.

3.4.1 Input-Lasche

Die Input-Lasche dient zur Bedienung unserer Applikation. Mittels dem Browse-Button kann im Filesystem nach einer Bilddatei gesucht werden, dieses wird nach bestätigen direkt analysiert. Der ermittelte Text aus dem Bild wird in der Textarea angezeigt. Dieser Text kann nach Belieben mittels „Save as text“-Button gespeichert werden.

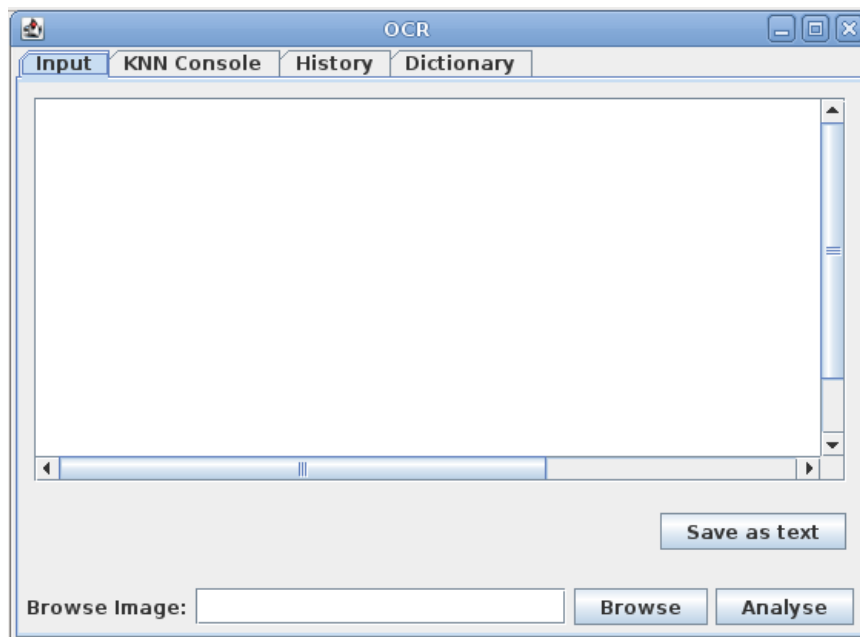


Abbildung 14: Input-Lasche

3.4.2 KNN Console-Lasche

In der KNN-Console-Lasche wird die Verarbeitung des analysierten Bildes angezeigt. Sie wird erst bei der Analyse eines Bildes mit Informationen gefüllt, und ist deshalb bei Start der Applikation noch leer. Sie enthält für jedes Zeichen die Position im Output Vektor des Neuronalen Netzwerkes, die dazugehörige Emphasis und das ermittelte Zeichen. Für jedes Wort wird die Wörterbuchverarbeitung angezeigt (Input und Output).

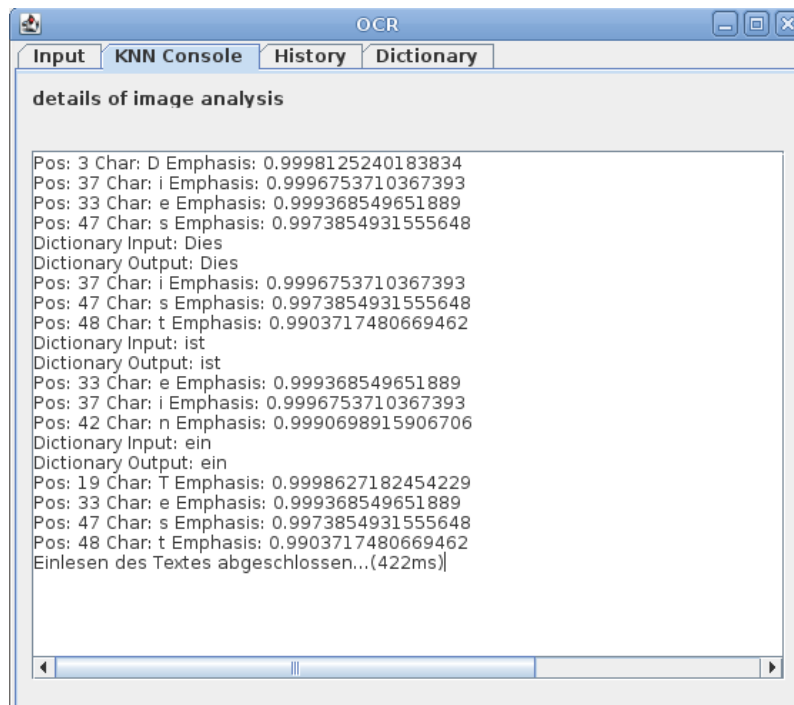


Abbildung 15: KNN Console - Lasche

3.4.3 History-Lasche

Die History-Lasche dient dazu, eine Übersicht der vorgängig analysierten Bilddateien zu haben. Sie wird abgesichert und ist auch beim nächsten Ausführen der Applikation noch ersichtlich. Links befindet sich eine Liste der verarbeiteten Bildnamen. Durch doppelklicken auf einen Bildnamen erscheint rechts oben das verarbeitete Bild. Der dazu analysierte Text erscheint in der Textarea rechts unten.

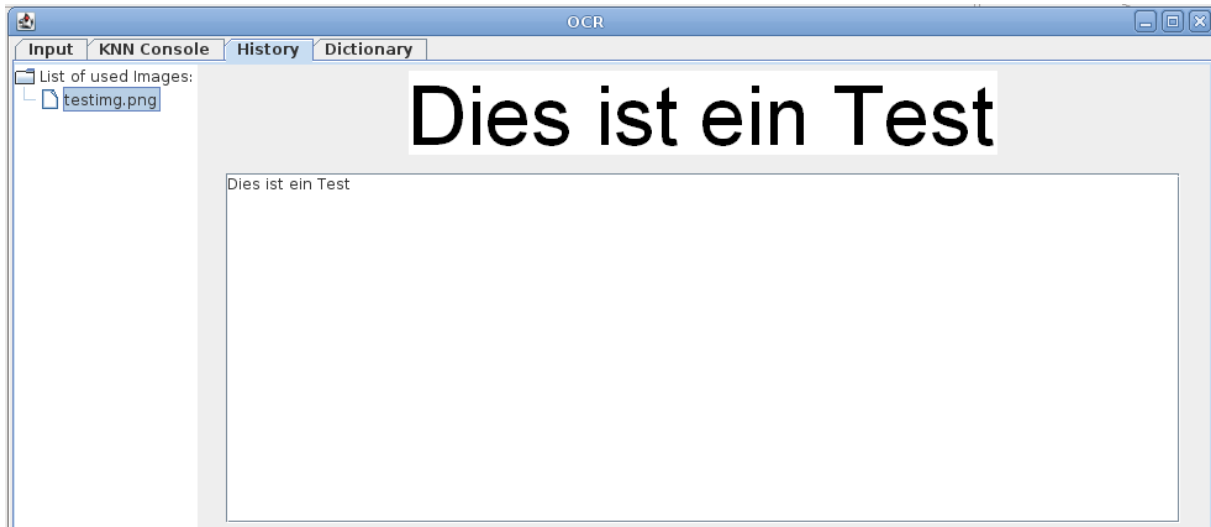


Abbildung 16: History-Lasche

3.4.4 Dictionary-Lasche

Eine Übersicht aller Wörter findet man in der Dictionary-Lasche. Durch Anklicken des gewünschten Buchstabens werden die ersten 500 Einträge mit dem gewählten Anfangsbuchstaben angezeigt. Durch klicken des „Show next entries“-Buttons werden jeweils weitere 500 Einträge des gewählten Buchstabens angezeigt. Sind alle Wörter des Buchstabens geladen, verschwindet der „Show next entries“-Button. Wählt man ein Wort an wie in Abbildung 17, hat man die Möglichkeit, dieses Wort mittels Textfeld unten links zu korrigieren/ändern.

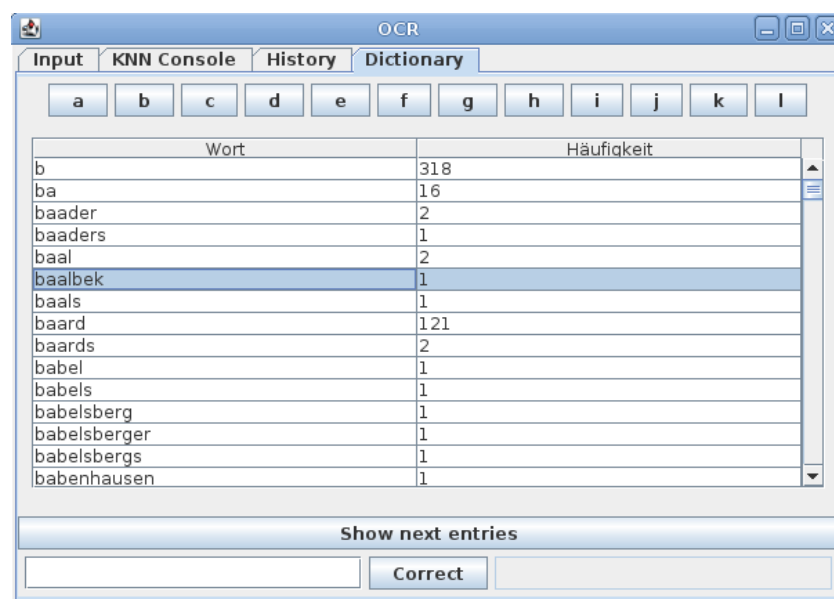


Abbildung 17: Dictionary-Lasche

4 Tests

Wir haben alle wichtigen und eindeutig testbaren Funktionen mit JUnit Tests abgedeckt. Aufgrund des undurchsichtigen Verhaltens Folgend eine kurze Erläuterung zu den programmierten Tests.

4.1 BitmapParser Tests

Bei allen BitmapParser Tests haben wir das Mock Pattern verwendet um jeweils eine Quelle für unsere Input ContrastMatrix zu haben.

4.1.1 CharacterParserTest

Es wird geprüft, ob leere Spalten in einer *ContrastMatrix* erkannt, und entsprechende Buchstaben-Matrizen korrekt daraus gebildet werden.

4.1.2 WordParserTest

Es wird geprüft, ob leere Spalten in einer *ContrastMatrix* erkannt, ob Wort-Abstände von Buchstaben Abständen unterschieden und ob Trennzeichen korrekt eingefügt werden.

4.1.3 RowParserTest

Es wird geprüft, ob leere Zeilen korrekt erkannt und entsprechende Trennzeichen korrekt eingefügt werden.

4.1.4 UnderlineRemoverTest

Es wird geprüft ob, Underlines erkannt und korrekt entfernt werden.

4.1.5 ContrastMatrixTest

Die Funktionalität folgender *ContrastMatrix*-internen Methoden wird geprüft:

- **testContrastMatrixIntInt()** – Wird die ContrastMatrix richtig initialisiert?
- **testContrastMatrixFunctionalCharacter()** – Wird die ContrastMatrix mit FunctionalCharacter richtig initialisiert?
- **testToString()** – Wird die ContrastMatrix richtig in einen String umgewandelt?
- **testGetValue()** – Werden gezielte Werte richtig ausgegeben?
- **testGetWidth()** – Wird die Breite richtig ausgegeben?
- **testGetHeight()** – Wird die Höhe richtig ausgegeben?
- **testGetFunctionalChar()** – Wird das Steuerzeichen richtig ausgegeben (falls vorhanden)?
- **testSetValue()** – Können Werte korrekt geschrieben werden?
- **testRemoveCol()** – Können Spalten entfernt werden?
- **testRemoveRow()** – Können Zeilen entfernt werden?
- **testInvertMatrix()** – Wird die Matrix korrekt invertiert?
- **testSetSubMatrix()** – Werden Teil-Matrizen korrekt zurückgegeben?
- **testIsFullRow()** – Werden volle Zeilen korrekt erkannt?
- **testIsEmptyRow()** – Werden leere Zeilen korrekt erkannt?
- **testIsFullCol()** – Werden volle Spalten korrekt erkannt?
- **testIsEmptyCol()** – Werden leere Spalten korrekt erkannt?
- **testIsFull()** – Werden volle Matrizen korrekt erkannt?
- **testTrim()** – Werden leere Spalten / Zeilen an den Rändern korrekt entfernt?

4.2 Dictionary Tests

Folgende Funktionalitäten des Dictionarys werden geprüft:

- **correctWordTest()** – Werden Wörter richtig korrigiert?
- **dontCorrectWordTest()** – Werden Wörter, welche im Wörterbuch enthalten sind unverändert zurückgegeben?
- **unknownCharTest()** – Wird das unknownChar Zeichen bevorzugt behandelt?
- **capitalLetterTest()** – Werden Wörter mit Grossbuchstaben auch nach der Korrektur mit Grossbuchstaben zurückgegeben?
- **firstSignNonAlphanumericTest()** – Werden Satz und Sonderzeichen vor dem Wort auch nach dem korrigieren korrekt zurückgegeben?
- **lastSignNonAlphanumericTest()** – Werden Satz und Sonderzeichen nach dem Wort auch nach dem korrigieren korrekt zurückgegeben?

4.3 MatrixHelper Tests

Folgende Methoden des MatrixHelpers werden geprüft:

- **convertToMatrixTest()** – Wird ein DoubleVektor korrekt in eine Matrix umgewandelt?
- **convertToDoubleVectorTest()** – Wird eine Matrix korrekt in einen DoubleVector umgewandelt?
- **unmergeThetasTest()** – Werden Thetas korrekt aus einem Vektor wieder hergestellt?
- **mergeThetasTest()** – Werden Thetas korrekt in einen Vektor „gemerged“?
- **maxTest()** – Wird das maximale Element einer Matrix korrekt ausgegeben?
- **sumTest()** – Werden die Elemente einer Matrix korrekt aufsummiert?
- **addScalarTest()** – Wird ein Skalar korrekt zu allen Matricelementen dazu addiert?
- **logTest()** – Werden alle Matricelemente korrekt logarithmiert?
- **add1ToVectorVerticalTest()** – Wird ein Vektor korrekt um eine 1 erweitert? (Vertikal)
- **add1ToVectorHorizontalTest()** – Wird ein Vektor korrekt um eine 1 erweitert? (Horizontal)
- **elementMultiplicationTest()** – Funktioniert die Elementweisemultiplikation von zwei Matrizen korrekt?
- **sigmoidTest()** – Wird die Sigmoidfunktion korrekt auf alle Matricelemente angewandt?
- **sigmoidGradientTest()** – Wird der Sigmoidgradient von allen Matricelementen korrekt berechnet?

5 Verbesserungsvorschläge bei Weiterführung des Projektes

5.1 Verbesserung Wörterbuch

Bei unserer Lösung ist es so, dass das Wörterbuch die Wörter aufgrund von statistischen Werten korrigiert ohne die Struktur des Satzes in die Entscheidung einfließen zu lassen. Zum Beispiel folgender Satz (wobei das Fragezeichen ein nicht erkanntes Zeichen symbolisiert): “Das ist ein gute? Test.” Beim Wort “gute?” kann das Programm das letzte Zeichen (ein r) nicht erkennen und gibt das Wort deshalb ans Wörterbuch. Dieses trifft, aufgrund statistischer Werten die Entscheidung, das Wort in “guten” zu ändern, da der Akkusativ öfter gebraucht wird als der Nominativ. Würde man das OCR produktiv nutzen wollen, müsste man die Satzstellung in die Entscheidung mit einfließen lassen. in diesem Beispiel prüfen, ob ein Akkusativ Sinn macht.

5.2 Unterstützte Zeichen

Um das im Rahmen dieses Projektes erstellte OCR produktiv verwenden zu können, müsste man den Unterstützten Zeichensatz erweitern.

5.3 Anzahl der versteckten Neuronen

Die Anzahl der versteckten Neuronen würde sich durch statistische Verfahren nach weiteren Faktoren optimieren lassen. Da unser KNN mit 250 versteckten Neuronen gut funktioniert und man in eine solche Analyse nahezu beliebig viel Zeit investieren könnte, haben wir im Rahmen des Softwareprojekts 2 darauf verzichtet.

5.4 Trennung von Zeichen

Bevor man ein solches Produkt produktiv einsetzen könnte, müsste man eine saubere Trennung der Buchstaben implementieren. Eine, die mit dem von uns aufgezeigten Problem (siehe [Zeichentrennung](#)) umgehen kann. Ein Ansatz hierfür wäre, dass man die Trennung der Zeichen nicht aufgrund von leeren Spalten macht sondern lückenlose, dunkle Stellen sucht und die so von hellen Stellen abgrenzt.

5.5 Umgang mit grossen Ä, Ö und Ü

Wie in Kapitel [Zeilentrennung](#) beschrieben besteht das Problem, dass die Punkte von grossen Ä, Ö und Ü abgeschnitten werden. Bevor ein solches Produkt produktiv eingesetzt werden könnte, müsste man hierfür eine Lösung finden. Zwei Lösungsansätze:

1. Durch das Abschneiden der Punkte hat man eine Zeile, die nur die Punkte solcher Buchstaben enthalten (siehe Abbildung 14). Sobald diese Zeile analysiert wird, wird dies erkannt und die nächste Zeile, zu denen die Umlautpunkte gehören wieder angefügt.
2. Man prüft bei der Zeilentrennung bereits, ob sich in einer Zeile nur Punkte befinden, und würde diese dann nicht abtrennen.

Zeile mit Ä und Ö zum Beispiel

Abbildung 18: Zeilentrennung bei grossen Umlauten

6 Auswertung

6.1 Burn Down Charts / Iterationsabschlüsse

6.1.1 1. Iteration, 14.03. – 04.04.2012

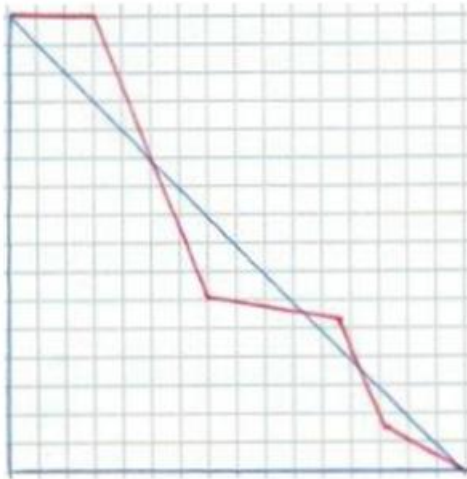


Abbildung 19: Iteration 1; Burndownchart und Planung⁵

Analyse:

In der ersten Iteration haben wir versucht alle Tasks zeitnah zu erledigen. Bis auf ein paar kleine Verspätungen ist uns dies auch gut gelungen. Die Verspätungen lassen sich dadurch erklären, dass wir uns bei der Zeilenanalyse anfangs nicht einig waren, wie wir diese umsetzen sollen. Ansonsten traten keine Komplikationen auf, wir konnten die Iteration erfolgreich abschliessen.

6.1.2 2. Iteration, 05.04. – 01.05.2012



Abbildung 20: Iteration 2; Burndownchart und Planung

⁵ Hochauflösende Bilder finden Sie im Anhang D

Analyse:

Die zweite Iteration begann anfangs Ostern, was man auch am Burndown-Chart erkennen kann. Über das Osterwochenende haben wir einen grossen Fortschritt gemacht um danach ein bisschen verschlafen zu können. Am Wochenende darauf hatten wir uns dann wieder getroffen um das Gui, die Zeilenanalyse und das Einlesen der Bilder in Angriff zu nehmen. An diesem Wochenende machten wir grosse Fortschritte. Nichtsdestotrotz mussten wir den Task „Gesamtübersicht Design“ in die nächste Iteration übernehmen, da wir entschieden hatten, das Gui mit mehr Funktionalität zu erweitern.

6.1.3 3. Iteration, 02.05 – 23.05.2012

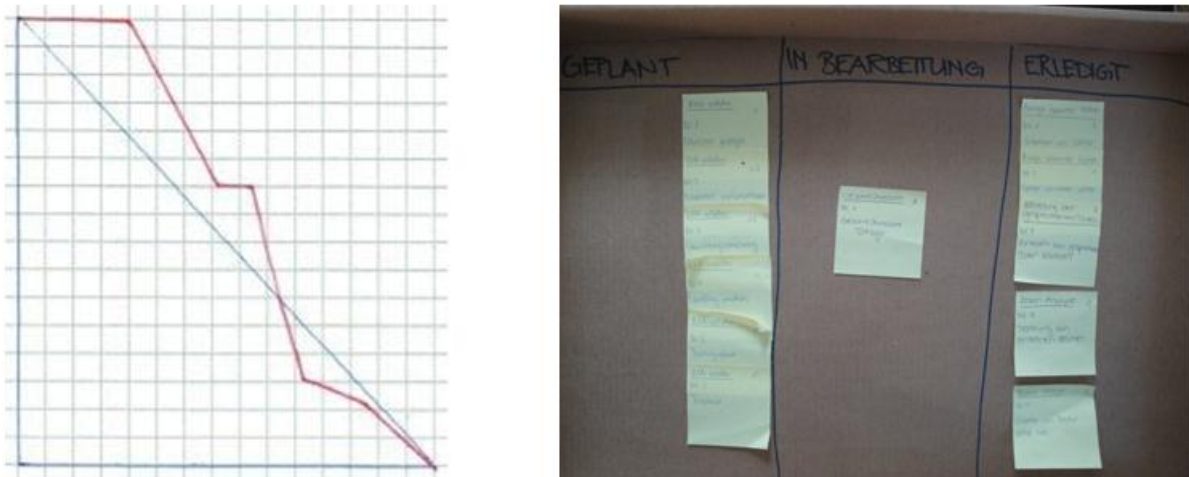


Abbildung 21: Iteration 3; Burndownchart und Planung

Analyse:

In der ersten Woche mussten wir uns überlegen wie wir mit dem Gui und der „Analyse der erkannten Zeichen“ weiterfahren. Dies sieht man auch im Burndown-Chart, da das Programmieren auf Eis gelegt war. Nach dem Entscheid, sich vor allem auf die Analyse der Zeichen und die Trennung von einzelnen Zeilen zu konzentrieren, nahm das Projekt wieder seinen gewohnten Lauf. Leider konnten wir auch in dieser Iteration den Task „Gesamtübersicht Design“ nicht abschliessen, da wir entschieden haben, in unserem Gui eine History anzuzeigen. Bis anhin hatten wir lediglich vor, ein Inputform sowie die Anzeige der Neuronen im Gui zu implementieren. Alle anderen Tasks konnten wie geplant abgeschlossen werden.

6.1.4 4. Iteration, 24.05. – 13.06.2012

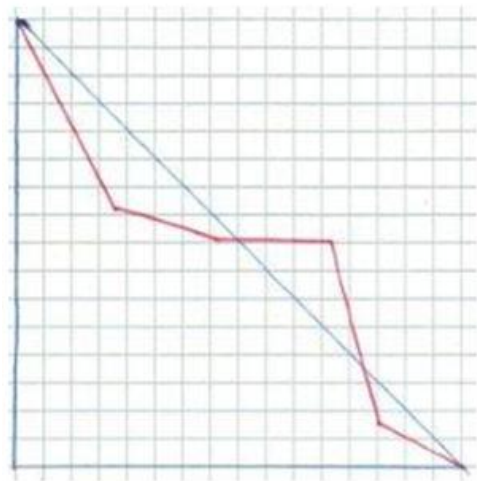


Abbildung 22: Iteration 4; Burndownchart und Planung

Analyse:

Endspurt: in der letzten Iteration haben wir nochmals richtig Gas gegeben. Wir haben das KNN implementiert sowie das Gui fertiggestellt. Die Implementation des KNN dauerte länger als erwartet, da auch noch die Trainingsphase dazu kam, jedoch konnten wir dies pünktlich abschliessen. Die verbleibende Zeit nutzten wir mit Schreiben der Dokumentation. Diesen Task hatten wir anfangs unterschätzt, durch gute Teamarbeit konnten wir diese Lücke in der Planung aber ausgleichen.

6.2 Auswertung des Projektes

6.2.1 Funktionalitätsreview

Wir konnten alle Ziele und Anforderungen an ein lauffähiges OCR erfüllen.

Um ein solches Produkt produktiv einsetzen zu können, müsste man jedoch einige Funktionen umschreiben und das Ganze um weitere Funktionalität erweitern. Im Kapitel [Verbesserungsvorschläge bei Weiterführung des Projektes](#) finden Sie einige Verbesserungsvorschläge welche zwar von uns als potentielle Probleme identifiziert wurden, jedoch aus zeitlichen Gründen nicht im Rahmen dieses Projektes implementiert werden konnten.

6.2.2 Projektreview

Wir blicken auf ein sowohl intensives als auch sehr spannendes Projekt zurück. Wir haben viel dazu gelernt und konnten unser, doch etwas simpel gewählter Ansatz, zu einer guten Lösung entwickeln. Einige Aspekte unseres OCRs waren zu Beginn des Projektes nicht vollständig klar und wir waren um die Unterstützung von Jens-Christian Fischer froh, welcher uns in den Iterationsmeetings stets mit guten Ideen und Denkanstössen beiseite stand. Deshalb wollen wir an dieser Stelle ein grosses Dankeschön an ihn richten.

Im Verlauf des Projektes konnten wir unser, zu Beginn eingeschränktes Know-How in Bezug auf künstliche Neuronale Netze, erweitern und uns solide, theoretische Grundlagen in diesem Gebiet aneignen. Wir konnten trotz der Tatsache, dass unser ursprünglich geplanter Ansatz nicht praktikabel war, auf unserem Grundlagewissen aufbauen und so ein gutes Endresultat erzielen.

Anhang A: Abbildungsverzeichnis

Abbildung 1: Einstufiges Neuronales Netzwerk	8
Abbildung 2: DecoratorPattern Pattern Bitmap Parser	9
Abbildung 3: Buchstabentrennung	10
Abbildung 4: Problem bei der Buchstabentrennung.....	11
Abbildung 5: Möglicher Lösungsansatz zur Lösung des Buchstaben-Trennproblems	11
Abbildung 6: Underlineentfernung Konstellation 1	12
Abbildung 7: Underlineentfernung Konstellation 2	12
Abbildung 8: Underlineentfernung Konstellation 3	12
Abbildung 9: Underlineentfernung bei Überlappung mit einem Zeichen	13
Abbildung 10: Topologie des verwendeten Neuronales Netzwerkes	13
Abbildung 11: Input Layer	14
Abbildung 12: Generierung eines Soll-Outputvektors	17
Abbildung 13: Wörter zum Wörterbuch hinzufügen	20
Abbildung 14: Input-Lasche.....	21
Abbildung 15: KNN Console - Lasche.....	21
Abbildung 16: History-Lasche.....	22
Abbildung 17: Dictionary-Lasche.....	22
Abbildung 18: Zeilentrennung bei grossen Umlauten	25
Abbildung 19: Iteration 1; Burndownchart und Planung	26
Abbildung 20: Iteration 2; Burndownchart und Planung	26
Abbildung 21: Iteration 3; Burndownchart und Planung	27
Abbildung 22: Iteration 4; Burndownchart und Planung	28

Anhang B: Tabellenverzeichnis

Tabelle 1: Trennzeichen	10
Tabelle 2: Anzahl versteckter Neuronen / Trainingsverhalten	14
Tabelle 3: Output-Neuronen.	15
Tabelle 4: KNN im Quellcode.....	19
Tabelle 5: Input des Wörterbuchs	20

Anhang C: Bibliographie

Jeff, H. (14. September 2008). *The Number of Hidden Layers*. Abgerufen am 06. Juni 2012 von <http://www.heatonresearch.com/node/707>

Jungblut, T. (06. Juni 2012). *Fmincg.java*. Abgerufen am 08. Juni 2012 von <https://github.com/thomasjungblut/thomasjungblut-common/blob/master/src/de/jungblut/math/minimize/Fmincg.java>

Norvig, P. (kein Datum). *How to Write a Spelling Corrector*. Abgerufen am 09. Juni 2012 von <http://norvig.com/spell-correct.html>

Sheng, K. (Dezember 2002). *Optical Character Recognition Based on OCRchie*. Abgerufen am 05. April 2012 von <http://pages.cs.wisc.edu/~dyer/cs766/hw/hw4/hw4-sheng/sheng.html>

Anhang D: Projektpläne

Hochauflösende Projektpläne finden Sie im Ordner „Anhang D“ im Anhänge Verzeichnis des beigelegten Datenträgers.

Anhang E: Vorlesungsunterlagen KNN

Die von Jens-Christian Fischer zu Verfügung gestellten Vorlesungsunterlagen zum Thema Künstliche Neuronale Netze finden Sie im Ordner „Anhang E“ im Anhänge Verzeichnis des beigelegten Datenträgers.