



**School of
Engineering**

InIT Institut für angewandte
Informationstechnologie

Bachelorarbeit Informatik

Digitalisierung von Schach-Formularen mittels Character Recognition Ensembling und Zug-Korrektur

Autoren

Volkan Caglayan
Bernt Nielsen

Hauptbetreuung

Prof. Dr. Mark Cieliebak

Datum

11.06.2021

Erklärung betreffend das selbstständige Verfassen einer Bachelorarbeit an der School of Engineering

Mit der Abgabe dieser Bachelorarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Bachelorarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmaßnahmen der Hochschulordnung in Kraft.

Ort, Datum:

Winterthur, 11.06.2021

Winterthur, 11.06.2021

Name Studierende:

Volkan Caglayan

Bernt Nielsen

Zusammenfassung

Um die Schachspiele bei Turnieren und Schachvereinen festzuhalten, werden diese auf Schach-Formulare in Papierform notiert. Die Vorlagen für die Schach-Formulare werden dabei von den jeweiligen Schachinstitutionen individuell gestaltet. Zur Digitalisierung dieser Schach-Formulare werden sie üblicherweise von Hand auf einem virtuellen Schachbrett nachgespielt, indem die Züge direkt abgelesen werden. Diese Vorgehensweise ist aber mühevoll und zeitaufwändig. Auf dem Markt gibt es momentan keine zufriedenstellende Lösung zu diesem Problem. Die Produkte, welche mittels Zeichenerkennung versuchen, die Züge zu erkennen, sind entweder nicht zuverlässig oder setzen voraus, dass eine Vorlage verwendet wird, die eigen für die Vereinfachung der Erkennung entworfen wurde. Alternative Produkte, wie digitale Schach-Formulare auf einem Tablet oder elektronische Schachbretter, werden auch nicht weitreichend genutzt. In dieser Arbeit wird nun eine bestehende Webapplikation weiterentwickelt, welche dem Benutzer hilft, die Digitalisierung der Schach-Formulare zuverlässig und intuitiv fertigzustellen. Dafür lädt der Benutzer als erstes das Bild des Schach-Formulars hoch. Danach werden zuerst die Boxen gesucht, in welchen sich die handgeschriebenen Züge befinden. Anschliessend werden diese vom Benutzer kontrolliert. Als nächstes werden die Zeichen innerhalb der Boxen erkannt und dem Benutzer präsentiert. Bei Bedarf korrigiert der Benutzer das Spiel nachträglich auf der Weboberfläche. Schlussendlich wird das Spiel als «Portable Game Notation»-Datei (PGN) zur Verfügung gestellt. Als Grundlage zur Box- und Zug-Erkennung dienen «Intelligent Character Recognition»-Services (ICR), die darauf spezialisiert sind, handschriftliche Zeichen zu erkennen. Der Kernpunkt der Arbeit ist die Idee eines Ensembles mit mehreren Anbietern von ICRs, was die Erkennung in hohem Grade zuverlässiger und genauer macht. Zur Überprüfung der Resultate wird ein eigenerstelltes Datenset verwendet, das von mehreren Personen geschrieben wird. Zusätzlich werden verschiedene Algorithmen zur Zug-Korrektur angewandt, so werden die typischen Fehler nach der Erkennung abgefangen. Die wenigen Fehler, die bestehen bleiben, werden vom Benutzer manuell korrigiert. Die Implementation der Applikation wird als Prototyp angesehen, Aspekte wie Deployment und Skalierbarkeit werden vorerst vernachlässigt. Die Arbeit bildet dafür aber eine solide Basis zur Weiterentwicklung.

Abstract

In order to record the chess games at tournaments and chess clubs, they are noted down on chess scoresheets in paper form. The templates for the chess forms are designed individually by the respective chess institutions. To digitize these chess scoresheets, they are usually replayed by hand on a virtual chess board by reading the moves. However, this procedure is tedious and time-consuming. There is currently no satisfactory solution to this problem on the market. The products that attempt to recognize the moves using character recognition are either not reliable or require the use of a template specifically designed to facilitate recognition. Alternative products, such as digital chess scoresheets on a tablet or electronic chess boards, are also not widely used. In this work, an existing web application is now further developed, which helps the user to complete the digitization of the chess scoresheets in a reliable and intuitive way. For this, the user first uploads the image of the chess scoresheet. Then the boxes containing the handwritten moves are searched. Afterwards the user validates them. Next, the characters inside the boxes are recognized and presented to the user. If necessary, the user corrects the game on the web interface. Finally, the game is made available as a «Portable Game Notation» (PGN) file. As a basis for box and move recognition, «intelligent Character Recognition» services (ICR) are used, which are specialized to recognize handwritten characters. The key point of the work is the idea of an ensemble with multiple providers of ICRs, which makes the recognition highly reliable and accurate. A self-created dataset written by multiple people is used to verify the results. In addition, various algorithms are applied for move correction, thus catching the typical errors after recognition. The few errors that remain are manually corrected by the user. The implementation of the application is considered as a prototype, aspects like deployment and scalability are neglected for the time being. However, the work forms a solid basis for further development.

Vorwort

Als wir die Ausschreibung für die Arbeit gesehen haben, waren wir, Bernt Nielsen und Volkan Caglayan, sehr daran interessiert. Wir wollten das erlernte Wissen während des Studiums über AI und Bildbearbeitung auf die Probe stellen, und diese Arbeit ermöglichte uns das. Es war nicht immer einfach, aber es hat uns gefreut, das Projekt Very Chess weiterzuentwickeln.

Wir möchten uns an dieser Stelle herzlich bei Prof. Dr. Mark Cieliebak bedanken. In den wöchentlichen Sitzungen hat er uns zuverlässig durch die Arbeit hinweg begleitet, mit vielen hilfreichen Diskussionen und Vorschlägen. Wir danken auch den Personen, die uns geholfen haben, das Wissen über Schach und die gängigen Vorgänge in Schachevents zu erlangen, dabei danken wir speziell Oliver Marti vom Schweizer Schachbund für die Zeit, der er sich für uns genommen hat. Als letztes möchten wir unsere Dankbarkeit für die Personen ausdrücken, die mit uns zusammen die Schach-Formulare geschrieben haben, um das Datenset fertigzustellen.

Winterthur, 11. Juni, 2021

Inhaltsverzeichnis

Zusammenfassung.....	I
Abstract.....	II
Vorwort.....	III
1 Einleitung	1
1.1 Aufgabenstellung und Zielsetzung.....	1
1.2 Aufbau der Arbeit	2
1.3 Übersicht zur vorherigen Applikation Very Chess.....	2
1.4 Funktionsumfang.....	5
1.5 Limitationen.....	6
2 Marktanalyse.....	7
2.1 Reine – Chess.....	7
2.2 CheSScan.....	9
2.3 Chess Score Pad	11
2.4 DGT-Schachbretter	12
2.5 Erkenntnisse	13
3 Theoretische Grundlagen	14
3.1 Zeichenerkennung	14
3.1.1 Optische Zeichenerkennung (OCR)	14
3.1.2 Intelligente Zeichenerkennung (ICR).....	14
3.2 Schachnotation	15
4 Vorgehen	18
4.1 Arbeitsweise.....	18
4.2 Tooling	19
4.2.1 ICR Preview Tools	19
4.2.2 CSV-Generator	20
4.2.3 Zug-Korrektur Analyse Tool.....	20
5 Implementation.....	22

5.1	Datenset.....	22
5.1.1	Vorheriges Datenset	22
5.1.2	Neues Datenset	23
5.1.3	Testing-Verfahren	24
5.2	Überblick.....	25
5.3	Preprocessing	26
5.3.1	Ausrichtung.....	26
5.3.2	Entfernte Preprocessing-Schritte.....	26
5.4	ICR-Anbieter	28
5.4.1	Tesseract	28
5.4.2	ABBYY Cloud.....	28
5.4.3	Google Vision API.....	29
5.4.4	Azure Cognitive Services	29
5.4.5	Amazon Rekognition	30
5.4.6	Anwendung des ICR-Ensembles.....	30
5.5	Box-Erkennung	30
5.5.1	Vorherige Box-Erkennung	31
5.5.2	Box-Erkennung mit ICR.....	31
5.5.3	Merkmale eines Schach-Formulars.....	32
5.5.4	Erkennung der Nummerierung	32
5.5.5	Erstellung der Boxen.....	36
5.5.6	Weitere Verbesserungen.....	42
5.5.7	Verwendung der Boxen.....	42
5.6	Zug-Korrektur	43
5.6.1	Confusion-Modul.....	43
5.6.2	Confidence-Modul.....	46
5.6.3	Ensembling der ICR-Ausgaben	48
5.6.4	Corrector-Algorithmus	50

5.6.5	Skipping-Algorithmus	52
6	Zusätzliche Änderungen.....	54
6.1	Benutzeroberfläche	54
6.2	Softwarearchitektur	59
7	Resultate.....	60
7.1	ICR-Anbieter	60
7.2	Box-Erkennung	61
7.3	Zug-Korrektur.....	62
7.4	Zeitersparnis	65
8	Diskussion.....	66
9	Ausblick.....	70
10	Verzeichnisse	72
10.1	Literaturverzeichnis	72
10.2	Abbildungsverzeichnis.....	75
10.3	Tabellenverzeichnis.....	77
11	Anhang.....	78
11.1	Projektmanagement	78
11.1.1	Offizielle Aufgabenstellung.....	78
11.1.2	Zeitplan	78
11.2	Weiteres.....	79
11.2.1	Anleitungen	79
11.2.2	Deployment.....	83
11.2.3	Dateistruktur.....	84
11.2.4	Confusion-Matrizen	86
11.2.5	Datenset.....	90

1 Einleitung

Das Schachspiel ist etwas, wovon jeder schon etwas gehört hat. Weniger bekannt ist, dass man bei offiziellen Schachturnieren und auch in den meisten Schachvereine die Spiele aufzeichnen muss. Dies wird üblicherweise mit einem Schach-Formular bewerkstelligt. Ein Schach-Formular ist ein Papier-Formular, auf welchem beide Spieler alle Züge des Spiels von Hand aufschreiben. Es dient dazu, das Spiel zu archivieren und fungiert auch als Rechtsmittel, da jeder Spieler alle Züge aufschreibt und das Formular als Bestätigung anschliessend unterschreibt. Die allgemein meist genutzte und anerkannte Notation für das Aufschreiben der Züge ist die Standard Algebraic Notation, abgekürzt SAN (siehe 3.2 Schachnotation).

Will nun ein Spieler oder Turnierveranstalter die Schachspiele digitalisieren, muss er die Schach-Formulare nehmen, jeden einzelnen Zug vom Formular ablesen und auf einem digitalen Brett nachspielen. Plattformen zum Nachspielen eines Schachspiels auf einem digitalen Brett bieten beispielsweise lichess.org [1] oder ChessBase [2]. Dieses Vorgehen wird auch beim Schweizer Schachverband [3] wie beschrieben angewendet.

Die Idee des bestehenden Projektes Very Chess ist nun, diesen Vorgang für den Benutzer zu vereinfachen. Dabei wird das Schach-Formular fotografiert und auf eine Weboberfläche hochgeladen. Anschliessend wird das Bild des Schach-Formulars mithilfe von AI analysiert und es wird versucht, das Spiel so weit wie möglich automatisch nachzuspielen. Dabei werden dem Benutzer beim Vorgang Optionen für die Korrektur zur Verfügung gestellt. Im Rahmen dieser Arbeit wird das Projekt Very Chess nun weiterentwickelt.

1.1 Aufgabenstellung und Zielsetzung

Das allgemeine Ziel ist es, eine Applikation zu entwickeln, welche ein Bild eines Schach-Formulars einliest. Die handgeschriebenen Schachzüge sollen mithilfe von optischer und intelligenter Zeichenerkennung (OCR, ICR) erkannt werden (siehe 3.1 Zeichenerkennung). Die Erkennungen werden geprüft und falls nötig korrigiert. Der Benutzer hat auch die Möglichkeit, selbst Korrekturen vorzunehmen. Das Ergebnis wird in der SAN-Notation als maschinenlesbare PGN-Datei ausgegeben (siehe 3.2 Schachnotation).

Es besteht bereits eine Applikation, die im Rahmen einer Bachelorarbeit fertiggestellt wurde [4]. Diese Applikation namens Very Chess soll nun erweitert und optimiert werden, so dass sich eine zufriedenstellende Lösung für den Benutzer entwickelt. Dies beinhaltet folgende Schritte:

- Analysiere, welche algorithmischen Schritte während der Bildbearbeitung für die Fehler bei den erkannten Zügen verantwortlich sind.
- Optimierte die diesbezüglichen Algorithmen.

- Erweitere und verbessere die vorhandene Webapplikation, da die Benutzeroberfläche zurzeit sehr rudimentär und manchmal nicht intuitiv ist.

1.2 Aufbau der Arbeit

Zunächst wird die bestehende Applikation beschrieben. Danach wird eine Marktanalyse durchgeführt, welche die bereits existierenden Produkte der Konkurrenz aufzeigt. Nach der Marktanalyse werden für das Verständnis der folgenden Kapitel die theoretischen Grundlagen erläutert. Weiter wird das Vorgehen bei der Entwicklung und die verwendeten Tools erklärt. Das darauffolgende Kapitel «Implementation» beinhaltet den Werdegang der Applikation, mit der Analyse der bestehenden Lösung und den darauf aufbauenden Designentscheidungen und Herausforderungen. Das Kapitel «Implementation» ist in die einzelnen Bestandteile der Applikation unterteilt. Im nächsten Kapitel «Resultate» werden die Ergebnisse dieser Implementierungen präsentiert. Schlussendlich werden die Resultate diskutiert und Verbesserungsvorschläge festgehalten. Weitere Informationen, wie die Anleitungen für die selbstentwickelten Tools, sowie für das Deployment sind nicht zentraler Bestandteil der Arbeit und somit im Anhang zu finden.

1.3 Übersicht zur vorherigen Applikation Very Chess

Die vorherige Version von Very Chess verwendet ein System, bei welchem die Schach-Formulare eine klare tabellarische Struktur haben müssen, und wo die Tabellenränder klar sichtbar und durchgehend sind. Um die Züge zu Orten, wendet die bisherige Version von Very Chess einen Algorithmus zur Tabellenerkennung an. Solch ein Schach-Formular kann auf der Webapplikation hochgeladen werden.

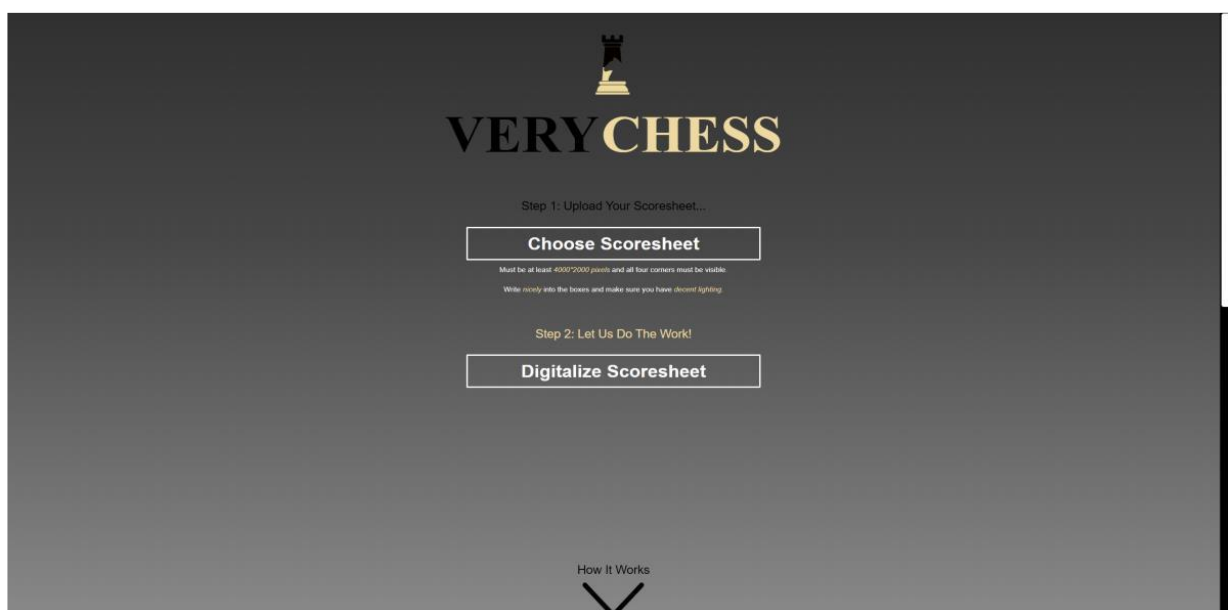


Abbildung 1: Seite zum Hochladen des Schach-Formulars für vorherige Applikation [4]

Als nächstes wird versucht, das Schach-Formular durch Bildtransformationen auszurichten.

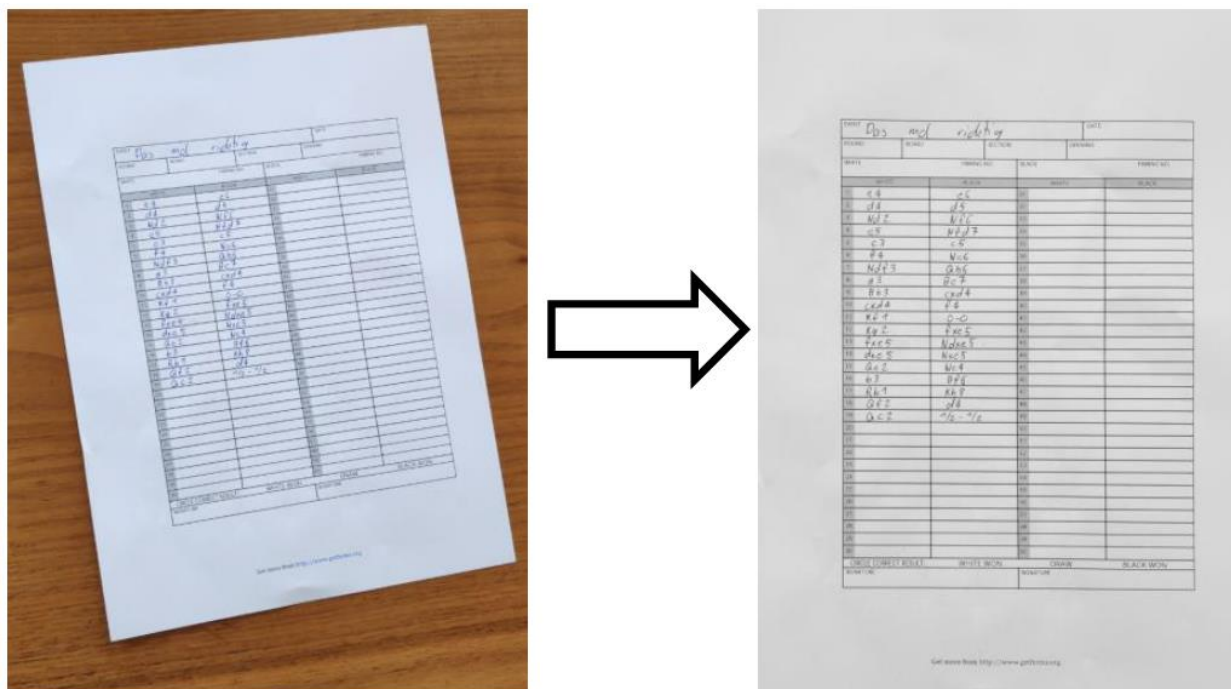


Abbildung 2: Ausrichtung eines Schach-Formulars [4]

Gelingt die Ausrichtung, wendet die Webapplikation den Algorithmus zur Tabellenerkennung an, um die Zellen zu finden, in welchen die Züge stehen. Der Benutzer legt dann noch fest, wo das Ende des Spiels im Schach-Formular ist, indem er auf die letzte Zelle verweist.



Abbildung 3: Bestätigung der letzten Zelle, in welcher der Zug steht [4]

Für den vorletzten Schritt kann der Benutzer bei Bedarf einzelne Zellen entfernen und dies wieder Bestätigen, um fehlerhafte Züge zu streichen, oder eventuelle Fehler bei der Erkennung zu korrigieren.

#	White	Black
1	e 4	e 5
2	N f 3	N c 6
3	B b 5	a 6
4	B a 4	N f 6
5	O-O	B e 7
6	B x c 6	d x c 6
7	R e 1	N d 7
8	d 3	O-O
9	N b d 2	f 6
10	N c 4	N c 5
11	b 3	N e 6
12	N e 3	N d 4

Abbildung 4: Entfernung der Kopfzeilen durch Benutzer [4]

Schlussendlich gehen die Zellen, die der Benutzer bestätigt hat, wieder an den Webserver. Dieser verwendet nun die Zellenkoordinaten auf dem Schach-Formular, um eine Abfrage an ABBYY Cloud zu erstellen. Die ICR-Ausgabe von ABBYY Cloud wird dann gefiltert. Diese gefilterte Zeichenerkennung wird dann als Eingabe für einen Algorithmus gebraucht, der mit Baumsuche versucht, die Eingabe in ein valides Schachspiel umzuwandeln. Das vorgeschlagene Spiel des Baumsuche-Algorithmus wird wieder an den Benutzer gesendet. Der Benutzer hat nun eine Benutzeroberfläche, wo er diese Vorschläge Zug für Zug validieren und korrigieren kann.

	White	Black	
1.	a4	b5	21.
2.	axb5	Nc6	
3.	b6	a6	
4.	b7	Ra7	
5.	b8-Q	Bb7	
6.	f4	e5	
7.	fxe5	d6	
8.	Qc8	Nxe5	
9.	c4	d5	
10.	Qh3	h6	
11.	Qh4	dx c4	
12.	Nc3	Ng6	
13.	d3	Bd6	
14.	Nf3	Nf6	
15.	Error		
16.			
17.			
18.			
19.			
20.			

The definition of colors:

- Move was validated by the user.
- Move could not be recognized.
- Move must be validated by the user.
- Move is not yet processed.

Info

Move needs to be validated by the user.

e4

We entered:

c4

Our suggestion:

-- select an option --

-- select an option --

e4

c4

Reset Table

Abbildung 5: Benutzeroberfläche zur Bearbeitung des vorgeschlagenen Spiels [4]

Wird eine Korrektur vom Benutzer durchgeführt, geht diese an den Server und ein neues Spiel wird vorgeschlagen. Sobald der Benutzer das ganze Spiel so durchgearbeitet hat, kann er es im standardisierten PGN-Format mit den gewünschten Turnierdaten herunterladen.

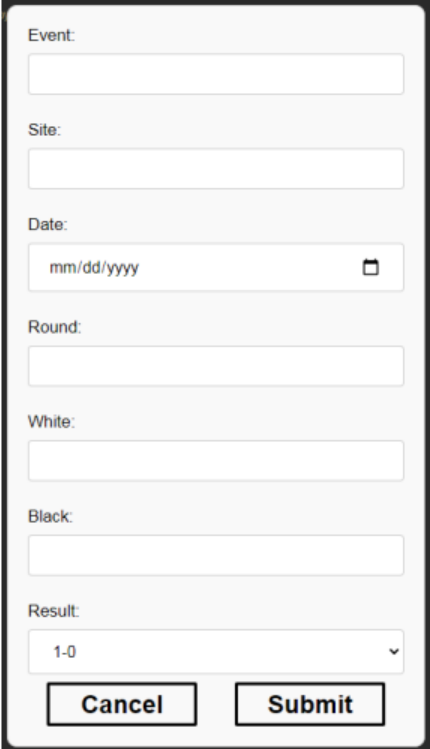
The image shows a web form for downloading a chess game. It contains several input fields: 'Event', 'Site', 'Date' (with a calendar icon), 'Round', 'White', and 'Black'. At the bottom, there is a 'Result' dropdown menu showing '1-0' and two buttons: 'Cancel' and 'Submit'.

Abbildung 6: Formular für das Herunterladen des Spiels [4]

1.4 Funktionsumfang

Durch die Analyse der Ausgangslage und dem Ausblick der vorherigen Bachelorarbeit [4] werden folgende Bereiche erkannt, die verbesserungsdürftig sind:

- Deployment der Applikation (Multiuser-Fähigkeit)
- Geschwindigkeit/Performance der Applikation
- Benutzeroberfläche
- Zeichenerkennung
- Box-Erkennung
- Zug-Korrektur

Da es sich bei Very Chess noch um einen Prototyp handelt, liegt der Fokus bei der Zeichenerkennung, Box-Erkennung sowie Zug-Korrektur. Verbesserungen in diesen Bereichen dienen am besten dazu, ein Proof-of-Concept für die Digitalisierung von Schach-Formularen zu belegen. Zusätzlich werden als zweite Priorität die Multiuser-Fähigkeit und Teile der Benutzeroberfläche verbessert.

1.5 Limitationen

Die Webapplikation wird weiterhin basierend auf dem Webframework Flask [5] mit Python entwickelt. Sie wird im Laufe der Arbeit so erweitert, dass mehrere Benutzer in der Lage sind, gleichzeitig mit der Applikation zu arbeiten. Die Anzahl der Benutzer ist jedoch weiterhin limitiert und nicht geeignet für eine produktive Umgebung. Die Applikation dient lediglich als Prototyp.

Als Input werden Bilder von Schach-Formularen mit einer Auflösung von mindestens Full-HD vorausgesetzt. Die empfohlene Auflösung ist jedoch 4K.

Die in der letzten Bachelorarbeit bestimmte Einschränkung auf den Umfang der unterstützten Schach-Formulare wird in dieser Arbeit aufgehoben [4]. Schach-Formulare ohne durchgezogene Tabellen werden neu auch unterstützt. Die Schach-Formulare müssen nun lediglich eine tabellarische Struktur aufweisen. Falls mehrere Tabellen vorhanden sind, müssen diese parallel angeordnet sein (siehe 5.1 Datenset).

2 Marktanalyse

Folgend werden bereits existierende Dienste zur Digitalisierung von Schach-Formularen für Endnutzer in einer Marktanalyse untersucht. Bei der Analyse der individuellen Anbieter liegt der Fokus bei den angebotenen Diensten und die dabei verbundenen Einschränkungen der Lösung. Es hat sich nach den Untersuchungen nämlich ergeben, dass es noch keine AI gibt, die besser Handschriften lesen kann als Menschen. Somit muss jeder Dienst irgendeine Form von Beihilfe schaffen. Diese Beihilfe kann die Überprüfung der Erkennung durch den Benutzer sein, Einschränkungen bei der Wahl des Schach-Formulars oder man setzt voraus, dass das Schach-Formular schöngeschrieben wird.

Zusätzlich werden auch Alternativen untersucht, die versuchen, das Schachspiel in einer anderen Weise zu digitalisieren als mit Papier-Formularen.

2.1 Reine – Chess

Eine bestehende Lösung zur Digitalisierung von Schach-Formularen ist eine Software namens Reine – Chess [6]. Um die Software zu nutzen, muss der Spieler aber eine standardisierte Vorlage als Schach-Formular verwenden, weil Reine – Chess nur mit dieser Vorlage funktioniert.

Diese Tatsache limitiert die Brauchbarkeit der Software, weil die meisten Schachturniere und Schachvereine ihre eigenen Vorlagen für das Schach-Formular festlegen. Dies wird zum einen gemacht, um der Veranstaltung mehr individuellen Charakter zu geben, und zum anderen werden häufig auch Sponsoren und Werbung auf den Schach-Formularen präsentiert.

Der Vorteil einer eigenen Vorlage ist aber, dass die Erkennung durch AI stark vereinfacht wird. Bei jeder Ecke des Schach-Formulars ist eine ArUco-Markierung [7] vorhanden. Diese Markierungen erlauben es, die Ausrichtung des Formulars optimal mit Bildtransformationen zu korrigieren, da die Markierungen immer mit ihren Bildkoordinaten und ihrer Rotation mitfotografiert werden. Diese Korrektur der Ausrichtung wird in den weiteren Schritten vorausgesetzt. Zusätzlich muss der Benutzer jedes Zeichen eines Schachzugs einzeln in eine der Zellen schreiben, was die Erkennung vereinfacht, aber auch ungewohnt für den Benutzer ist.

Abbildung 7: Standard Vorlage für Schach-Formular von Reine – Chess [6]

Der Verlauf einer typischen Anwendung von Reine – Chess ist also vereinfacht:

1. Bild aufnehmen
2. Ausrichten des Bildes mit ArUco-Markierungen
3. Individuelle Zeichen aus Bild ausschneiden
4. Preprocessing der Zeichen für das Convolutional-Neural-Network (CNN)
5. Klassifikation mit dem CNN (Eigenes CNN für jede Länge des Schachzuges)
6. Postprocessing der Klassifikation
7. Herunterladen des Spiels in PGN-Format

Limitationen

- Unterstützt nur eigene Vorlage als Schach-Formular.
- Die Zeichen für die Schachzüge müssen einzeln in die Zellen eingetragen werden.
- Ein Zug kann maximal 5 Zeichen enthalten, und es sind nur alphanumerische Zeichen erlaubt. «#, =, +, -» sind also nicht erlaubt, was dem Standard für die SAN-Notation widerspricht.

Es ist noch erwähnenswert, dass die Reine – Chess-Applikation zusätzlich zur Erkennung von Schach-Formularen auch andere Dienste wie beispielsweise die Analyse des Schachspiels mit einem Schachcomputer bereitstellt. Very Chess könnte so auch zur Analyse erweitert werden (siehe 9 Ausblick).

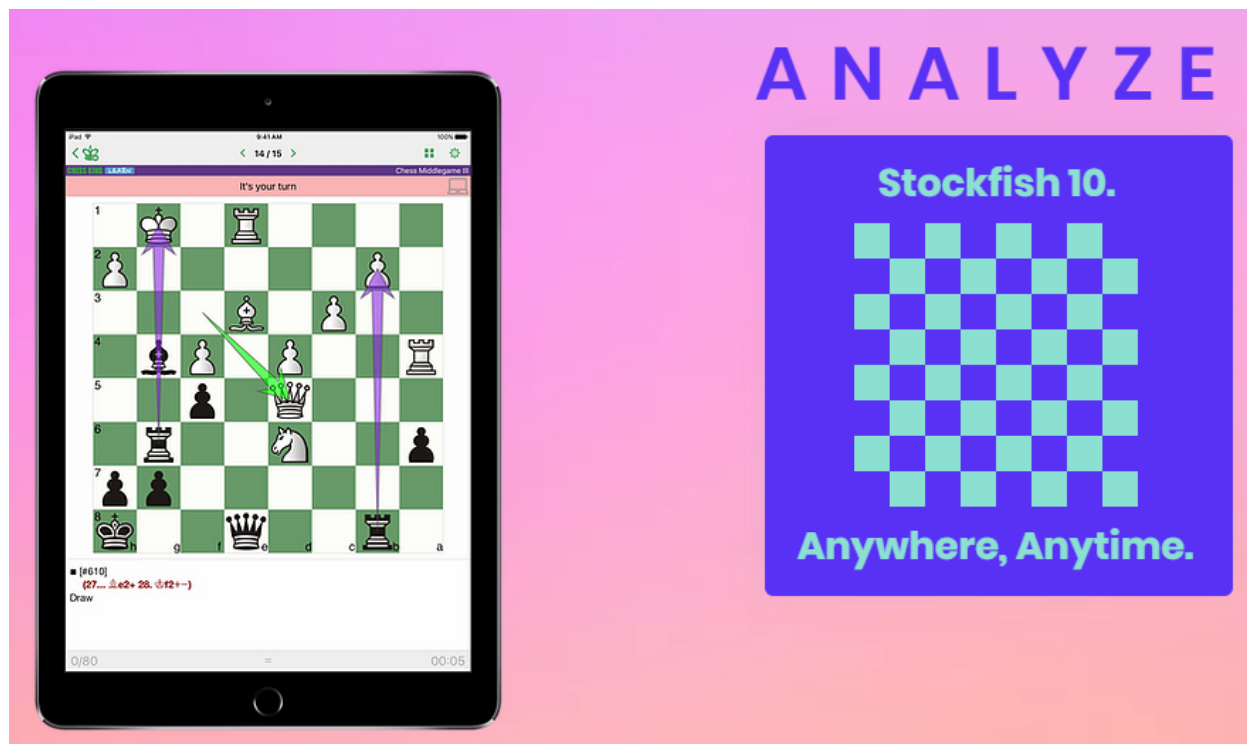


Abbildung 8: Analyse Tool von Reine – Chess mit Schachcomputer [6]

2.2 CheSScan

Ein anderes Produkt auf dem Markt, welches gleich wie Very Chess beliebige Vorlagen für Schach-Formulare zulässt und die Digitalisierung vereinfacht, ist CheSScan [8]. CheSScan ist eine Mobile-App verfügbar für iOS und Android. Hier wird auch wieder zuerst ein Bild vom Schach-Formular gemacht. Weiter wird das Bild automatisch analysiert und die erkannten Züge werden angezeigt (siehe Abbildung 9: CheSScan Benutzeroberfläche für Android). Der Benutzer kann dann nachträglich die Züge einzeln korrigieren indem er sie überschreibt oder auf dem visuellen Schachbrett spielt. Schlussendlich wird wieder die PGN-Datei für das Spiel exportiert. Wie schon Reine – Chess bietet CheSScan ebenfalls ein integriertes Analyse Tool.



Abbildung 9: CheSSScan Benutzeroberfläche für Android [8]

Da CheSSScan nicht Open-Source ist, kann nicht genau untersucht werden, wie die Erkennung im Detail funktioniert. Die App gibt auch keine Rückmeldung, wenn das Formular nicht erkannt wird. Die folgenden Limitationen sind daher lediglich Beobachtungen, die durch eigene Experimente erlangt wurden.

Limitationen

- Wird ein Schach-Formular nicht erkannt, gibt es keine Rückmeldung an den Benutzer, was genau nicht funktioniert, sondern lediglich eine Standardmeldung. Dies macht es dem Benutzer schwer, herauszufinden, warum die Erkennung fehlschlägt.
- Die Beobachtungen zeigen, dass viele Arten von Schach-Formularen die Erkennung mit CheSSScan erschweren/verunmöglichen. Es braucht ein Formular mit klarer Tabellenstruktur und durchgezogenen Linien.
- Es kann nur schön und getrennt (jedes Zeichen mit Abstand vom nächsten) geschriebene Züge erkennen, die nicht kursiv geschrieben sind, was viele Handschriften ausschliesst.

2.3 Chess Score Pad

Eine Alternative dazu, die Schach-Formulare zu digitalisieren ist, das Schachspiel direkt digital einzugeben, während dem es läuft. Ein Produkt, welches diese Alternative anbietet, heisst Chess Score Pad [9]. Chess Score Pad ist nur auf iOS verfügbar.

Die Applikation ist eine Mobile-App, die eine Benutzeroberfläche bietet, mit welcher die Züge im Verlauf des Spiels eingegeben werden. Dabei prüft die Software direkt auch die Legalität des Zuges und wendet die richtige Notation an.

Diese Vorgehensweise hat den Vorteil, dass das Spiel direkt digitalisiert wird, und wie erwähnt, sind ungültige Spiele nicht möglich, sei es wegen illegalen Zügen oder falscher Notation. Für eigene Spiele unter Freunden und Schachvereine, die diese Vorgehensweise einführen, stellt Chess Score Pad eine gute Lösung dar.

Problematisch ist aber, dass der Standard in praktisch allen Schachinstitution ist, die Schachzüge in Papierform aufzuschreiben. In diesen Fällen müsste man die Schachzüge zuerst auf das Schach-Formular schreiben, und dann zusätzlich im Chess Score Pad eintragen, was die Nützlichkeit der Software einschränkt. Ein anderer Nachteil ist, dass der Akku der verwendeten Tablets bzw. Mobiltelefonen durch die Nutzung aufgebraucht wird. Es ist also eventuell nötig, eine Steckdose in der Nähe zu haben, um das Gerät aufzuladen. Je nach Zeitkontrolle können Schachspiele nämlich bis zu 8 Stunden dauern.

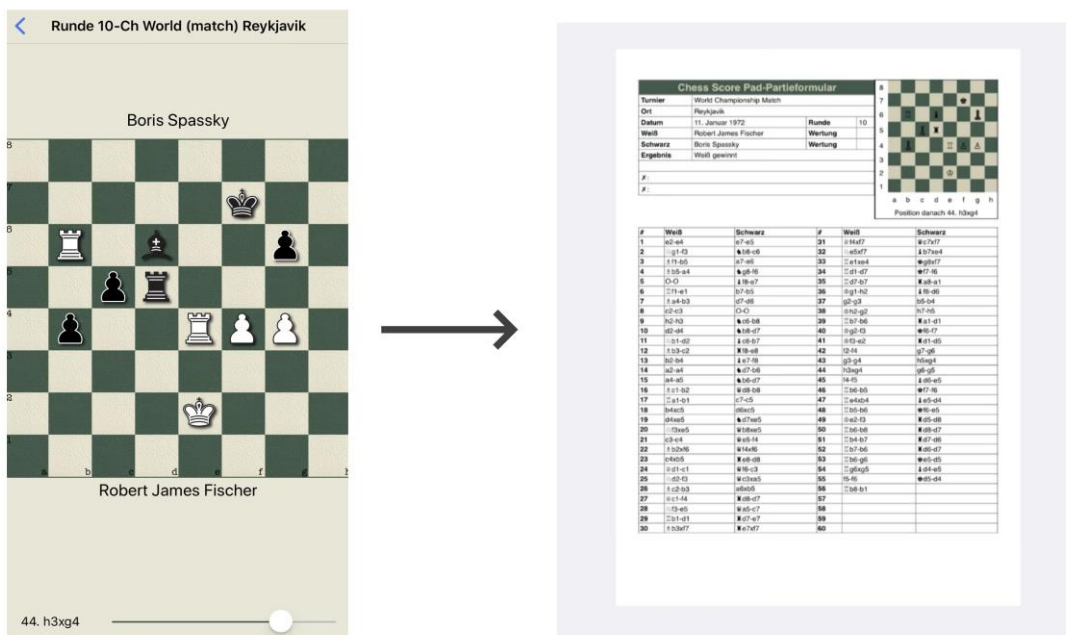


Abbildung 10: Chess Score Pad – Benutzeroberfläche mit Export als Schach-Formular [9]

2.4 DGT-Schachbretter

Eine Variante, die Schachzüge direkt während des Spiels zu digitalisieren, wird schon in allen grösseren Turnieren und vereinzelt im privaten Gebrauch angewendet. Diese Variante beinhaltet die Nutzung eines DGT-Schachbrettes [10], wobei DGT für Digital Game Technology steht. Ein DGT-Schachbrett ist ein Schachbrett, das mit Sensoren und einem eigenen Computersystem erkennt, wohin die physikalischen Figuren gezogen werden. Die Züge werden somit automatisch in Echtzeit aufgezeichnet. Der Computer, der die Züge mittels der Sensoren erkennt, ist meist ein separates Gerät. Der Computer kann beispielsweise in der Schachuhr sein.



Abbildung 11: DGT-Schachbrett verbunden mit Smart-Schachuhr [10]

Diese Bretter haben auch noch den speziellen Anwendungsfall, dass sie bei Turnieren die Schachspiele live für die Zuschauer übertragen.

«DGT Schachbretter sind von Schacholympiaden, Weltmeisterschaften im Schach und vielen weiteren Schachveranstaltungen auf der ganzen Welt nicht mehr wegzudenken.» [11]

Das Hauptproblem der DGT-Schachbretter liegt am Preis. Sie sind viel teurer als übliche Schachbretter, und werden deswegen auch fast ausschliesslich an hochrangigen Turnieren benutzt. Zusätzlich ist das Aufsetzen der Bretter aufwändiger, und es wird ein Stromanschluss benötigt.

2.5 Erkenntnisse

Unsere Untersuchungen haben ergeben, dass es noch keine Lösung zur Digitalisierung von Schach-Formularen gibt, die bequem, schnell und benutzerfreundlich ist. Die Lösungen, welche die Schach-Formulare ersetzen, scheitern daran, dass die Schachinstitutionen das Aufschreiben der Züge auf ein Papier-Formular voraussetzen. Man müsste diese Voraussetzungen ändern, doch Änderungen bei der Kultur des Schachspiels waren noch nie einfach, also wird das Papier-Formular sicherlich noch bis in naher Zukunft bestehen.

Bei den Produkten, die wie Very Chess das Schach-Formular einlesen und dann durch OCR/ICR-Methoden versuchen, das Spiel zu erkennen, gibt es ähnliche Probleme. Zum einen gibt es wieder Konflikte damit, dass die Schachinstitute eigene Schach-Formulare zur Nutzung für ihre Turniere voraussetzen. Es ist also sehr schwierig, ein eigenes Schach-Formular, dass bei der Erkennung helfen soll, einzuführen.

Für das Produkt CheSScan, welches beliebige Schach-Formulare erlaubt, funktioniert häufig die Erkennung gar nicht. Ausserdem gibt es keine hilfreiche Rückmeldung. Auch wenn es funktioniert, werden dem Benutzer einfach die vorgeschlagenen Züge gegeben, ohne Korrekturmöglichkeiten während der Erkennung anzubieten. Hat es also Fehler, was je nach Handschrift praktisch gegeben ist, muss der Benutzer das Spiel wieder von vorne nachspielen.

Diese Erkenntnisse schliessen darauf, dass es für ein Produkt wie Very Chess nach wie vor eine Marktlücke gibt. Eine benutzerfreundliche Software zur Digitalisierung von Schach-Formularen, welche die meisten Schach-Formulare annimmt und für jede Art von Handschrift bei der Erkennung durch ihre Vorschläge einen Mehrwert bietet. Dabei muss der Benutzer klar durch das Spiel geführt werden. Korrekturen durch den Benutzer müssen so bequem und intuitiv wie möglich gemacht werden. Gibt es Fehler bei der Erkennung, muss eine klare Rückmeldung erfolgen, idealerweise mit Instruktionen, um die Erkennung möglich zu machen.

3 Theoretische Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen der Arbeit erläutert.

3.1 Zeichenerkennung

In diesem Projekt wird Software für die Zeichenerkennung verwendet, um den Inhalt der Schach-Formulare einzulesen. Generell unterscheidet man zwischen optischer und intelligenter Zeichenerkennung. Zu beachten ist aber, dass der Übergang von optischer zur intelligenten Zeichenerkennung fließend ist.

3.1.1 Optische Zeichenerkennung (OCR)

Die optische Zeichenerkennung (OCR) sorgt dafür, dass aus eingescannten Dokumenten oder Bilddateien der Text erkannt und erfasst wird. OCRs führen pro Zeichen Preprocessing-Schritte gefolgt von der eigentlichen Erkennung aus. Das Preprocessing hat das Ziel, die Chance auf eine erfolgreiche Erkennung zu erhöhen. Dazu gehören Techniken wie Line Removal, Zoning und weitere. Die Erkennung des Zeichens wird mit Hilfe von Postprocessing in Form von Nachschlagewerken verbessert. Dazu wird geprüft, ob aus den Zeichen ein passendes Wort gebildet werden kann. Dies ist wichtig für Zeichen, die oft verwechselt werden [12].

Die Implementation einer eigenen Zeichenerkennungssoftware wird in der vorherigen Bachelorarbeit beschrieben. Dazu wird ein Convolutional Neural Network trainiert und mit entsprechenden Pre- und Postprocessing-Techniken eingesetzt. Das Ergebnis funktioniert zwar, ist aber nicht in der Lage, Zeichen zu erkennen, bei welchen die Linien nicht verbunden sind. Dies ist jedoch gewöhnlich für handgeschriebene Texte. Folgend ist die Implementation nicht geeignet für die Zeichenerkennung von Schach-Formularen.

3.1.2 Intelligente Zeichenerkennung (ICR)

In dieser Arbeit werden intelligente Zeichenerkennungsservices (ICR) eingesetzt. Diese werden benötigt, um handgeschriebene Zeichen auszulesen. Diese Art der Erkennung ist entsprechend komplexer als bei herkömmlichen OCRs. Jedoch benutzen diese unter anderem auch die gleichen Pre- sowie Postprocessing-Techniken. ICRs verwenden für die Erkennung der Zeichen selbst lernende Neural Networks, welche ihre Datenbank automatisch aktualisieren, um neue handgeschriebene Muster zu erkennen [13].

3.2 Schachnotation

Die international anerkannte und übliche Notation zur Aufzeichnung von Schachzügen ist die Standard Algebraic Notation [14], kurz SAN. SAN wird auch von der FIDE [15], der internationalen Schachföderation, als Standard für ihre Schachveranstaltungen vorausgesetzt.

Das Grundprinzip von SAN ist, dass nur genau die Informationen notiert werden, welche im Kontext der Schachposition nötig sind. Die Notation ist kompakt, und das Aussprechen der Züge ist natürlicher. Das heisst aber auch, dass die notierten Züge nicht allein für sich stehen, sondern nur zusammen mit dem Spielverlauf Sinn ergeben.

SAN ordnet jeder Figur ausser dem Bauern ein Zeichen zu, dieses wird dann immer am Anfang des Zuges geschrieben. Wird ein Bauer gezogen, fällt dieser Schritt weg.

Figur	Zeichen
König	K
Dame	Q
Turm	R
Springer / «Pferd»	N
Läufer	B

Tabelle 1: Zeichen für Figuren in SAN-Notation

Als letztes wird im Normalfall das Zielfeld der Figur für den Zug notiert. Felder im Schach werden mit den Koordinaten von «a» bis «h» (Spalten) und 1 bis 8 (Reihen) definiert. In den meisten Schachpositionen und dazugehörigen Zügen reicht dies aus.

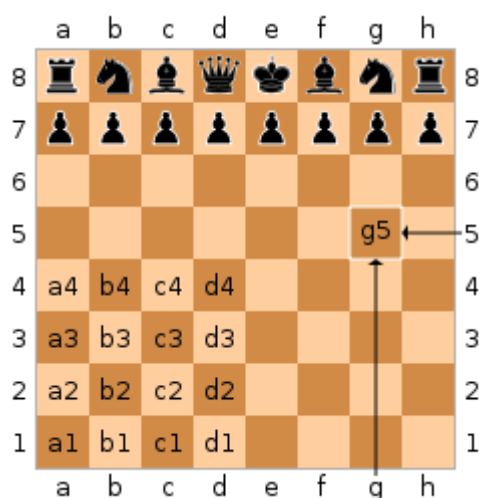


Abbildung 12: SAN-Definition der Felder [14]

Es bestehen nun aber Ausnahmen, gibt es in einer Schachposition mehrere Möglichkeiten, den Figurentyp auf das Zielfeld zu ziehen, muss vom Quellfeld noch die entsprechende Koordinate vor dem Zielfeld notiert werden. Bei der folgenden Illustration gibt es beispielsweise 2 Springer, die auf das Zielfeld «d2» ziehen können. Die Notation wäre normalerweise «Nd2», doch jetzt muss der Spieler mit «Nbd2» oder «Nfd2» spezifizieren, welchen Springer er beabsichtigt zu ziehen.

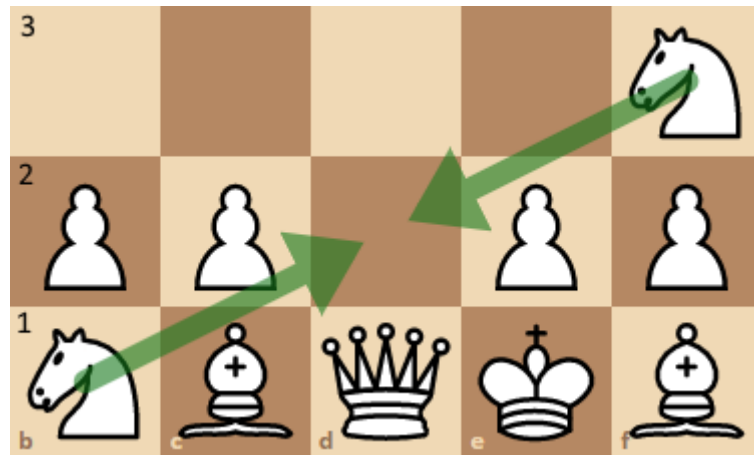


Abbildung 13: Beispiel wo beide Springer auf Zielfeld «d2» ziehen können

Es gibt noch weitere Spezialfälle, bei welchen zusätzliche Notationen definiert sind. Wird beispielsweise eine Figur geschlagen, wird noch ein «x» zwischen Figur und Zielfeld notiert. Weitere solche Fälle sind: Schachgebot, Schachmatt, Promotion des Bauern und die Rochade (für weitere Informationen siehe [14]).

SAN wird auch verwendet, um die Spiele im Dateiformat PGN (Portable Game Notation [16]) für die Digitalisierung, zusammen mit verschiedenen Meta-Informationen, zu speichern.

```
[Event "Gibraltar Masters 2019"]
[Site "Caleta ENG"]
[Date "2019.01.22"]
[Round "1.123"]
[White "Teh,Wee Zhun"]
[Black "Bulmaga,I"]
[Result "0-1"]
[WhiteElo "1818"]
[BlackElo "2407"]
[ECO "E52"]

1.d4 Nf6 2.c4 e6 3.Nc3 Bb4 4.e3 O-O 5.Bd3 d5 6.Nf3 b6 7.O-O Bb7 8.Bd2 Bd6
9.cxd5 exd5 10.Rc1 a6 11.Qc2 Re8 12.Rfd1 Nbd7 13.Bf1 Ne4 14.g3 Qe7 15.Be1 Rac8
16.Bg2 f5 17.Nd2 g6 18.Nf1 Ndf6 19.Na4 Ng5 20.a3 Nf7 21.Qb3 Ba8 22.Qd3 Bb7
23.b4 Qe6 24.Qb3 Bf8 25.Nb2 Ne4 26.Nd3 g5 27.a4 c6 28.a5 b5 29.f3 Ned6 30.Bf2 Nc4
31.Re1 Qg6 32.Rcd1 Rc7 33.f4 g4 34.Ne5 Qf6 35.Re2 Nfxe5 36.fxe5 Qe7 37.Be1 Bc8
38.Bc3 Be6 39.Nd2 Nxe5 40.Rf2 Nf7 41.Bf1 Bc8 42.Re1 Bh6 43.Rfe2 Qd6 44.Nb1 Ng5 0-1
```

Abbildung 14: Inhalt einer PGN-Datei mit SAN-Notation unterhalb

Dieses Dateiformat kann von allen gebräuchlichen Software-Produkten für Schach eingelesen werden. PGN wird auch von Very Chess, als Resultat der Digitalisierung des Spiels, zur Verfügung gestellt.

4 Vorgehen

Folgend wird das Vorgehen während des Projektes beschrieben. Dabei werden auch die Tools vorgestellt, die zur Fertigstellung der Arbeit beigetragen haben. Die Tools wurden im Verlauf des Projektes selbst entwickelt, um simple Arbeitsschritte zu automatisieren. Diese können in Zukunft auch für die Weiterentwicklung von Very Chess weiterverwendet werden.

4.1 Arbeitsweise

Wegen des Coronavirus konnte die Arbeit nicht wie üblicherweise vor Ort, an einem zugeteilten Bachelorarbeitsplatz, realisiert werden. Deshalb wurde die gesamte Arbeit von Zuhause aus erbracht. Um das Team inklusive des Dozenten auf dem aktuellen Stand bezüglich der Arbeit zu bringen, hatten wir jede Woche eine ca. einstündige Sitzung auf Microsoft Teams [17]. Während diesen Sitzungen wurden auch Probleme besprochen und Ideen ausgetauscht. Ausserhalb dieser wöchentlichen Sitzungen war es jedoch auch nötig zu kommunizieren, weswegen Discord [18] für Sprachchats und Instant-Messaging innerhalb des Teams verwendet wurde.

Für die Versionierung des Quellcodes von Very Chess und unseren erstellten Tools wird Git genutzt. Die Repositories sind auf GitHub abgelegt. Zusätzlich wird GitHub auch genutzt, um alle Teilaufgaben in Issues aufzuteilen, und diese dann im Project-Board zu verwalten.

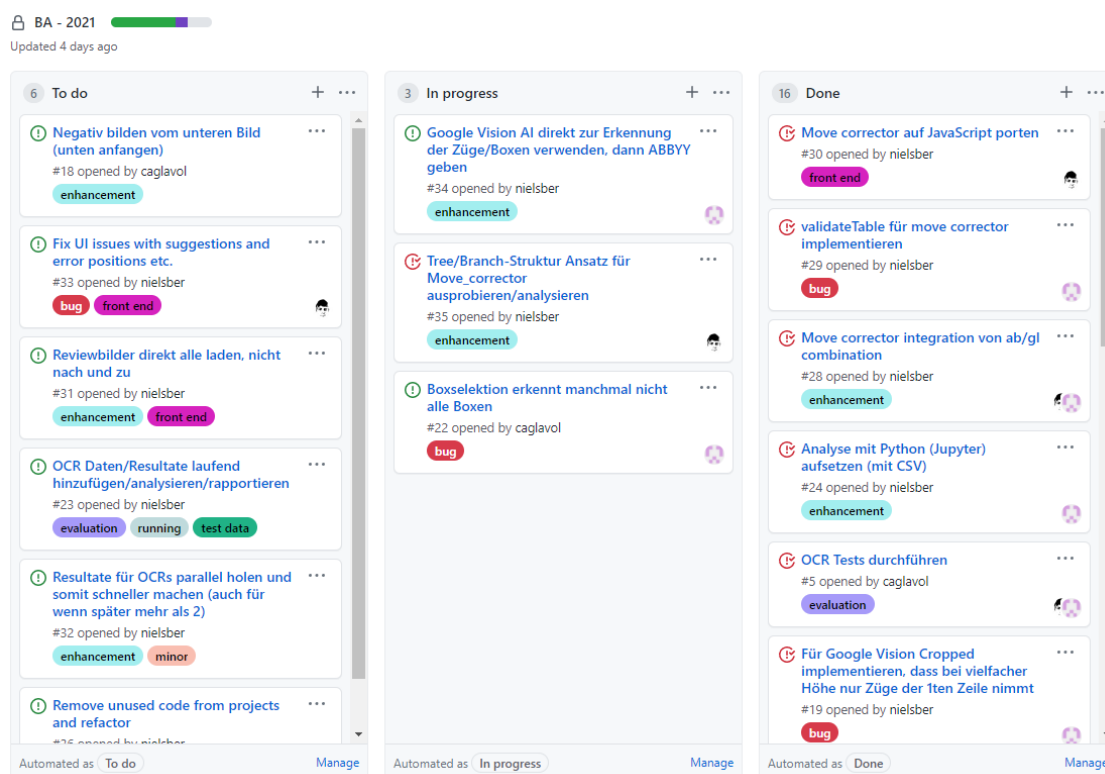


Abbildung 15: Momentaufnahme Project-Board von GitHub für Very Chess

Als letztes dient OneDrive dazu, alle Dokumente und andere Dateien zu speichern, die nicht zum Quellcode gehören. OneDrive ermöglicht zusätzlich, mehreren Mitgliedern des Teams gleichzeitig an Dokumenten zu arbeiten, und verwaltet auch die Versionierung dieser.

4.2 Tooling

Für die Evaluation des neuen Programmcodes werden Tools entwickelt, welche die einzelnen Schritte der Applikation für ein ganzes Datenset durchführen. Dies erspart dem Entwickler die Zeit, die er ansonsten für die manuelle Durchführung der Tests und Evaluationen benötigt. Zu beachten ist, dass sich Very Chess im gleichen Verzeichnis befinden muss. Ansonsten funktionieren die Tools unter Umständen nicht. Mehr Information zu der Inbetriebnahme der Tools sind im Anhang unter Anleitungen zu finden (siehe 11.2.1 Anleitungen).

4.2.1 ICR Preview Tools

Die ICR Preview Tools sind einfache HTML Pages, welche es über einen simplen Upload erlauben, ein Bild an einen ICR-Service zu schicken. Die Erkennungen werden anschliessend im Bild eingezeichnet. Somit kann eingesehen werden, ob ein ICR Probleme mit einem gewissen Schach-Formular hat, oder ob der nachfolgende Programmcode fehlerhaft ist. Es existieren zwei Versionen, eine für die Anzeige der Boxen, sowie eine weitere für die Anzeige der erkannten Zeichen. Dieses Tool unterstützt besonders die Entwicklung der Box-Erkennung und existiert für alle verwendeten ICRs (siehe 5.4 ICR-).

CHESS SCORE SHEET THE REGENCYCHESS COMPANY

http://www.regencychess.co.uk

DATE 22.01.2011

WHITE Puranit

BLACK Logthetis, &

EVENT Gib Masters

ROUND BOARD WHITE WIN BLACK WIN

1	Nf3	NfC	21	C4	Na5	41
2	93	d5	22	Rady	N63	42
3	Bg2	et	23	Qe2	N66	43
4	0-0		24	R61	Bet	44
5	013	0-0	25	N8	Na4	45
6	N- %	C5	26	Nxd4	cxele	46
7	e4	Nc6	27	usas	Nd5	47
8	Red	65	28	cxd5	Rx61	48
9e5		Nd7	29	R407	lexd549	
10	NF1	QC	30	er	f6	50
11	Bf4	367	31	BL 3	Be7	51
12	h4	Rfc8	32	Bf5		52
13	N162	I Qd8	33		1-0	53
14	N5	lat	34			54
15	N8f3	64	35			55
16	Qd2	a5	36			56
17	Ng4	Bf8	37			57
18	h 5	24	38			58
19	a3	txa339	39			59
20	6xa3	Ra68	40			60

© The Regency Chess Company 2012 e-mail: info@regencychess.co.uk http://www.regencychess.co.uk



Abbildung 16: Anzeige der erkannten Zeichen im ICR Preview Tool für Google Vision API

Abbildung 17: Anzeige der Boxen im ICR Preview Tool für Google Vision API

4.2.2 CSV-Generator

Der CSV-Generator führt die komplette Pipeline (Box-Erkennung und Zug-Korrektur) von Very Chess für die ausgewählten Schach-Formulare aus. Dieses Tool steht in der Form eines Python-Scripts zur Verfügung. Als Input werden die Bilder der Schach-Formulare und die passenden PGNs benötigt. Beim Start des Tools führt es die Very Chess-Pipeline für alle Schach-Formulare aus und erhält die ICR-Erkennungen pro Schach-Formular als Rückgabewert. Die Erkennungen werden anschliessend in ein CSV geschrieben. Das CSV beinhaltet die Spielnummer, die Zugnummer, den tatsächlichen Schachzug aus dem PGN, sowie alle ICR Erkennungen.

game	ply	pgn	ab	gl	az	rk	gl2
1	1	Nf3	Nf3	N43	Nf3	Nf3	N43
1	2	Nf6	Nf6	Nf6	Nff	Nf6	Nf6
1	3	g3	g3	93	93	3	93
1	4	d5	d5	15		15	15
1	5	Bg2	b5	Be2	Bg2	Ba2	Be2
1	6	e6	e6	e6	et	et	e6
1	7	O-O	O-O	O-O	O-O	O-O	O-O
1	8	Be7	Be7	Be7	Bet	Be7	Be7
1	9	d3	d3	3	23	d3	3
1	10	O-O	O-O	O-O	O-O	O-O	O-O

Abbildung 18: Ausschnitt des CSV-Exports aus dem CSV-Generator

Die Laufzeit des CSV-Exports kann bis zu 30 Minuten betragen. Dies ist abhängig von der Anzahl der zu exportierenden Schach-Formulare. Die genannte Laufzeit gilt für das komplette Datenset von 50 Schach-Formularen. Eine parallele Implementation wurde bereits versucht, jedoch erlauben die ICRs teils nur wenige Anfragen innerhalb kurzer Zeit.

4.2.3 Zug-Korrektur Analyse Tool

Das Zug-Korrektur Analyse Tool ist eine Kopie der eigentlichen Zug-Korrektur von Very Chess. Anstatt in JavaScript, ist es in Python implementiert, da sich Python für Auswertungen eignet. Dieses Tool nimmt den CSV-Export vom CSV-Generator als Input und führt den Corrector-Algorithmus darauf aus (siehe 5.6.4 Corrector-Algorithmus). Somit können verschiedenste Parameter im Code angepasst und getestet werden, ohne wiederholt ICR-Anfragen zu tätigen. Dies spart nicht nur Zeit, sondern auch Kosten bei der Entwicklung, besonders bei der Nutzung von mehreren ICRs. Für die Evaluation der Resultate wird Cross-Validation eingesetzt. Die Confusion-Matrizen aller ICRs werden mit allen ausser den aktuell zu testendem Spiel erstellt. Diese werden anschliessend für das Testing des aktuellen Spiels genutzt. Dieser Vorgang wird für alle Schachspiele im CSV durchgeführt.

Das Tool gibt die Anzahl der falschen Erkennung des ICR-Ensembles aus. Zusätzlich enthält die Ausgabe die Anzahl der falschen Erkennungen, nachdem sie durch den Corrector-Algorithmus korrigiert werden, sowie die prozentuale Angabe dieser Werte pro Schachspiel. Am Schluss des Outputs ist das Total ersichtlich, das aussagt, wie viele der Erkennungen noch manuell durch den Benutzer korrigiert werden müssen.

```
[Game 40] In wrong: 60 Out wrong: 5 Reduction: 54.55% -> 4.55%
[Game 41] In wrong: 40 Out wrong: 1 Reduction: 54.05% -> 1.35%
[Game 42] In wrong: 43 Out wrong: 1 Reduction: 47.78% -> 1.11%
[Game 43] In wrong: 22 Out wrong: 0 Reduction: 27.50% -> 0.00%
[Game 44] In wrong: 23 Out wrong: 0 Reduction: 45.10% -> 0.00%
[Game 45] In wrong: 16 Out wrong: 1 Reduction: 41.03% -> 2.56%
[Game 46] In wrong: 13 Out wrong: 0 Reduction: 31.71% -> 0.00%
[Game 47] In wrong: 20 Out wrong: 1 Reduction: 45.45% -> 2.27%
[Game 48] In wrong: 17 Out wrong: 0 Reduction: 41.46% -> 0.00%
[Game 49] In wrong: 24 Out wrong: 0 Reduction: 30.00% -> 0.00%
[Game 50] In wrong: 12 Out wrong: 0 Reduction: 32.43% -> 0.00%
[Total >ALL] In wrong: 1794 Out wrong: 87 Reduction: 56.58% -> 2.74%
```

Abbildung 19: Ausgabe des Zug-Korrektur Analyse Tools

Ausserdem kann durch das Setzen einer weiteren Option, die Confusion-Matrix aller ICRs als CSV exportiert werden. Eine weitere Option, die innerhalb des Codes ausgewählt wird, erlaubt die genaue Analyse eines konkreten Spiels. In diesem Fall werden pro Spielzug die Erkennungen des ICR-Ensembles ausgegeben. Der Entscheid der Zug-Korrektur, dessen Confidence, sowie der tatsächliche Schachzug werden ebenfalls angezeigt.

```
#1 01. In: Nf3 N43 Nf3 Nf3 N43 Out: Nf3 Pgn: Nf3
Conf: 1.00
#1 02. In: Nf6 Nf6 Nff Nf6 Nf6 Out: Nf6 Pgn: Nf6
Conf: 1.00
#1 03. In: g3 93 93 3 93 Out: g3 Pgn: g3
Conf: 0.91
#1 04. In: d5 15 <> 015 15 Out: d5 Pgn: d5
Conf: 0.51
```

Abbildung 20: Detailausgabe des Zug-Korrektur Analyse Tools

5 Implementation

Dieses Kapitel beschäftigt sich mit der Analyse der Ausgangslage und die darauffolgende Designentscheide. Die wichtigsten Implementationsschritte werden festgehalten und erläutert.

5.1 Datenset

Für die Entwicklung und Evaluation der Applikation wird ein Datenset benötigt, welches aus handgeschriebenen Schach-Formularen besteht.

5.1.1 Vorheriges Datenset

Folgend wird das Datenset der vorherigen Bachelorarbeit evaluiert. Es besteht aus zwei Teilen. Ein Teil mit 21 Schach-Formularen für die Evaluation der Erkennungen von ABBYY Cloud [19]. Diese 21 Schach-Formulare verwenden das gleiche Layout und das exakt gleiche Schachspiel. 12 weitere Schach-Formulare werden für die Evaluation der vorherigen Box-Erkennung eingesetzt. Diese besitzen unterschiedliche Layouts [4].

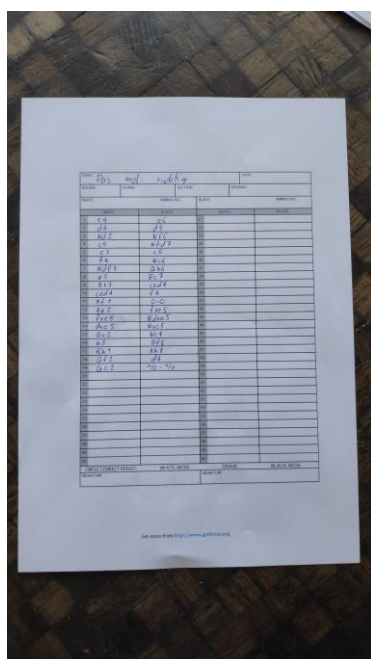


Abbildung 22: Schach-Formular aus dem Layout-Datenset [4]

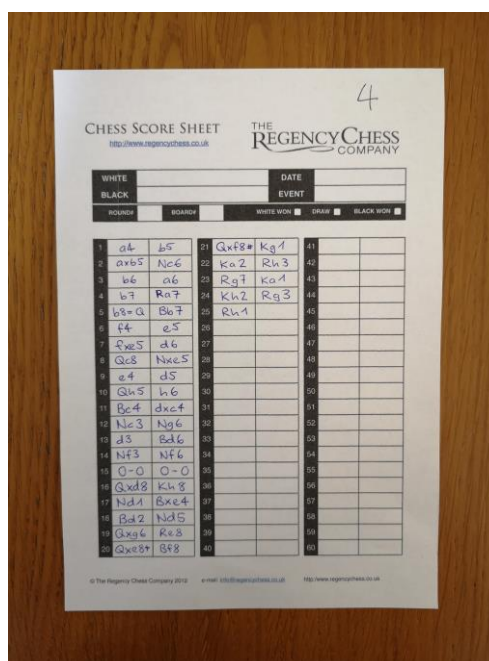


Abbildung 21: Schach-Formular aus dem Erkennungs-Datenset [4]

Das Hauptproblem des bestehenden Datensets ist der wiederholte Einsatz des gleichen Schachspiels für alle Schach-Formulare. Eine Evaluation der Applikation mit nur einem konkreten Schachspiel macht eine unzureichende Aussage über die Richtigkeit und Qualität der Zug-Korrektur. Deshalb wird im nächsten Schritt ein neues Datenset erstellt.

5.1.2 Neues Datenset

Das im Verlauf der Arbeit erstellte Datenset wird für die Evaluation der kompletten Applikation eingesetzt. Im ersten Schritt besteht es aus 20 selbstgeschriebenen Schach-Formularen. Danach wird es mithilfe von freiwilligen Teilnehmern erweitert auf 34, und in einer letzten Iteration mit einem Total von 50 handgeschriebenen Schach-Formularen vervollständigt. Die letzten 8 Schach-Formulare werden dabei vom Datenset der letzten Bachelorarbeit übernommen, um zusätzliche Layouts aufzunehmen [4]. 44 Schach-Formulare haben das Layout der Regency Chess Company. Dies ist das Standardlayout, welches bereits hauptsächlich in der vorherigen Bachelorarbeit genutzt wurde. Es werden 5 weitere Layouts mit den letzten 8 Schach-Formularen hinzugefügt.

EVENT				DATE			
ROUND		BOARD		SECTION		OPENING	
WHITE				BLACK			
PAIRING NO.		PAIRING NO.		PAIRING NO.		PAIRING NO.	
WHITE	BLACK	WHITE	BLACK	WHITE	BLACK	WHITE	BLACK
1			31				
2			32				
3			33				
4			34				
5			35				
6			36				
7			37				
8			38				
9			39				
10			40				
11			41				
12			42				
13			43				
14			44				
15			45				
16			46				
17			47				
18			48				
19			49				
20			50				
21			51				
22			52				
23			53				
24			54				
25			55				
26			56				
27			57				
28			58				
29			59				
30			60				
CIRCLE CORRECT RESULT		WHITE WON		DRAW		BLACK WON	
SIGNATURE				SIGNATURE			

Abbildung 24: Beispiel eines alternativen Schach-Formulars [4]

CHES SCORE SHEET THE REGENCY CHESS COMPANY
<http://www.regencychess.co.uk>

WHITE			DATE
BLACK			EVENT
ROUND#	BOARD#	WHITE WON <input type="checkbox"/>	DRAW <input type="checkbox"/>
		BLACK WON <input type="checkbox"/>	

1		21		41	
2		22		42	
3		23		43	
4		24		44	
5		25		45	
6		26		46	
7		27		47	
8		28		48	
9		29		49	
10		30		50	
11		31		51	
12		32		52	
13		33		53	
14		34		54	
15		35		55	
16		36		56	
17		37		57	
18		38		58	
19		39		59	
20		40		60	

© The Regency Chess Company 2012 e-mail: info@regencychess.co.uk <http://www.regencychess.co.uk>

Abbildung 23: Standard Schach-Formular [4]

Jedes Schachspiel, welches in das Datenset aufgenommen wird, ist einmalig vorhanden. Die PGNs der Schachspiele sind alle von gültigen und offiziellen Spielen, welche öffentlich zugänglich sind [20]. Dabei wird darauf geachtet, dass sowohl lange wie auch kurze Schachspiele im Datenset vertreten sind.

Die 50 Schach-Formulare sind von 9 verschiedenen Personen verfasst worden. Dabei sind pro Person 2 bis 10 Schach-Formulare geschrieben worden. Die Spiele sind 38 bis 118 Spielzüge lang. Grundsätzlich galt die normale Handschrift zu benutzen. Die Idee war die gleiche Schrift wie im Alltag zu verwenden, um das Resultat nicht zu verfälschen. Trotzdem ist anzunehmen,

dass die in den Turnieren ausgefüllten Schach-Formulare weniger lesbar sind. Im Folgenden sind zwei Beispiele vom Datenset, welche bereits ausgerichtet sind. Diese sind als schwer lesbar eingestuft.

Abbildung 25: Schach-Formular «game_26»

Abbildung 26: Schach-Formular «game_33»

Wie bereits in den zwei vorherigen Abbildungen zu sehen ist, werden im Datenset für das Ausfüllen der Schach-Formulare mehrere Schriftfarben verwendet. Im Datenset sind folgende Schriftfarben vertreten:

- blau
- grau (Bleistift)
- rot
- schwarz
- grün

Weitere Details zum Datenset sind im Anhang zu finden (siehe 11.2.5 Datenset).

5.1.3 Testing-Verfahren

Das Datenset besteht aus den handgeschriebenen Schach-Formularen und den entsprechenden PGNs. Für die Evaluation werden die zum Input gehörenden PGN-Züge gegen die Predictions der Algorithmen geprüft. Für die Evaluation wird Cross-Validation verwendet (siehe 4.2.3 Zug-Korrektur Analyse Tool).

5.2 Überblick

Der Workflow der ursprünglichen Applikation wird angepasst und sieht nun wie folgt aus. Der Preprocessing-Schritt wird komplett entfernt, während direkt nach dem Upload des Schach-Formulars die Anfrage an das ICR-Ensemble getätigt wird. Die erhaltenen Erkennungen des ICR-Ensembles werden für die Box-Erkennung benötigt. Im Anschluss der Box-Erkennung selektiert der Benutzer die Box mit dem letzten gültigen Schachzug. Dies startet eine Anfrage an ABBYY Cloud mit den erzeugten Boxen. Im Anschluss werden die Erkennungen des ICR-Ensembles (inklusive ABBYY Cloud) aufbereitet und der Zug-Korrektur im Frontend zur Verfügung gestellt. Diese spielt das Schachspiel mithilfe des vordefinierten Algorithmus durch und bietet dem Benutzer die Möglichkeit, manuelle Korrekturen vorzunehmen. Führt der Benutzer eine Korrektur durch, wird der darauf aufbauende Spielverlauf von der Zug-Korrektur erneut ermittelt.

In der folgenden Abbildung wird der angepasste Workflow von Very Chess illustriert. Dabei stehen die grün gekennzeichneten Prozesse für Vorgänge im Frontend und die weiss gekennzeichneten Prozesse für Vorgänge im Backend.

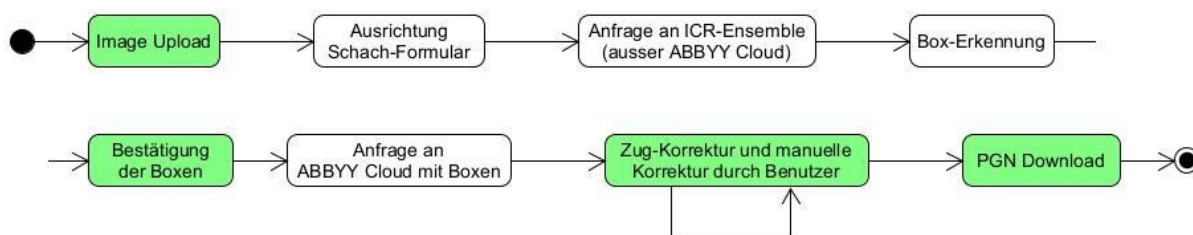


Abbildung 27: Angepasster Workflow für Very Chess

5.3 Preprocessing

Hier werden die Bildbearbeitungsschritte der Applikation behandelt. Diese werden für die vorherige Box-Erkennung verwendet und für die Aufbereitung des Schach-Formulars für ABBYY Cloud.

5.3.1 Ausrichtung

Der Algorithmus für die Ausrichtung der Schach-Formulare wird beibehalten [4]. Dazu wird weiterhin der Mobile Document Scanner [21] der vorherigen Arbeit verwendet (siehe 1.3 Übersicht zur vorherigen Applikation Very Chess). Bei Versagen der Ausrichtung wurde bisher ein Fehler ausgegeben. Um nun mehr Schach-Formulare zu erlauben, wird bei einem Fehler mit dem ursprünglichen Bild fortgefahren. Dies ist nun möglich, weil die neue Box-Erkennung auch mit nicht ausgerichteten Schach-Formularen funktioniert.

5.3.2 Entfernte Preprocessing-Schritte

Extraktion der Tabelle

Die Erkennung der Tabelle mit OpenCV [22] gehört zur ursprünglichen Lösung und wird mit der neuen Box-Erkennung nicht mehr benötigt, da nun die Erkennungen der ICRs für die Box-Erkennung verwendet werden.

Entfernung der Schatten

Die Schatten im Bild wurden entfernt, bevor es an ABBYY Cloud geschickt wird. Dieser Schritt wird nicht mehr benötigt, da solche Preprocessing-Techniken bereits von den ICRs eingesetzt werden (siehe 3.1 Zeichenerkennung).

SCORE SHEET						DATE (DD.MM.YYYY)
EVENT: FarbenFroh & Lang					TIME CONTROL	
ROUND	BOARD	SECTION	OPENING (ECO CODE)			
WHITE			PAIRING NO.	RATING		
BLACK			PAIRING NO.	RATING		
	WHITE	BLACK		WHITE	BLACK	
1	e4	c6	21	Bc2	Bc2	
2	d4	d5	22	Ne4	Nb8	
3	Nd2	dxe4	23	Nd6+	Bxd6	
4	Nxe4	Nd7	24	Rxd6	Nd7	
5	Bc4	Ngf6	25	Rhd1	Bxf3	
6	Ng5	e6	26	gxf3	Vxe5	
7	Qe2	Nb6	27	Bxd4+	Kc7	
8	Bb3	a5	28	Bb5	Rob8	
9	a3	a4	29	f4	Nf3	
10	Ba2	b6	30	Rd7+	Kf6	
11	N5f3	c5	31	f5	Vxh2	
12	Bf4	Nbd5	32	Fxe6	Fxe6	
13	Be5	Qa5+	33	Bc6	Vg4	
14	Qd2	Gxd2+	34	f4	Ve3	
15	Vxd2	Vg4	35	Rg1	g5	
16	Ngf3	Nxe5	36	Fg5+	bxg5	
17	dxe5	a6	37	Ke1	Nf5	
18	c4	Ne7	38	Bc4	Rbd8	
19	O-O-O	Bb7	39	Rb7	Rd6	
20	Bb7	Nc6	40	h4	Rh3	

CIRCLE GAME RESULT: WHITE WON DRAW BLACK WON
 Signature (White) Signature (Black) Signature (Arbiter)

Abbildung 28: Box-Erkennung ohne Ausrichtung

Ausserdem weisen die Erkennungen vom bearbeiteten Bild im Vergleich zum Ursprünglichen keine Verbesserung vor. Um an diese Erkenntnisse zu gelangen, sind Stichproben mit dem ICR Preview Tool durchgeführt worden (siehe 4.2.1 ICR Preview Tools).

Entfernung der Tabellenlinien und Vervollständigung der Zeichen

Die Boxausschnitte, welche aus der Box-Erkennung erstellt werden, beinhalten meist Linien der Tabelle. Häufig überraget ein Zeichen wie zum Beispiel «g» über die untere Zellenlinie. Dieser Preprocessing Schritt entfernt die Linie vom Boxausschnitt und vervollständigt das überragende Zeichen, um anschliessende das bereinigte Bild an ABBYY Cloud zu senden [23].

Die aktuelle Implementation der Linienentfernung ist noch nicht einsatzbereit. Die Linien werden entfernt, jedoch bleiben in einigen Bildern Artefakte bestehen. Diese Artefakte sorgen für eine Verschlechterung der Resultate bei ABBYY Cloud.

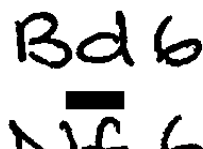


Abbildung 29: Artefakte bei der Linienentfernung

In diesem Preprocessing-Schritt werden die Ausschnitte in Schwarz-Weiss-Bilder umgewandelt. Dies bewirkt, dass selbst Teile der Schachzüge zu weiss umgewandelt und somit Teil des Hintergrunds werden. Besonders betroffen sind Schach-Formulare, welche mit Bleistift geschrieben sind, weil diese einen niedrigen Kontrast zum Hintergrund aufweisen.

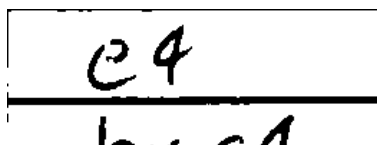


Abbildung 30: Entfernung von Zeichenkonturen

Für diese Arbeit wird dieser Preprocessing-Schritt entfernt, da die aktuelle Lösung zu instabil ist. Eine genauere Analyse dieser Funktionalität ist nötig, um das beschriebene Fehlverhalten zu beheben. Dies ist aber nicht Teil dieser Arbeit.

5.4 ICR-Anbieter

In diesem Kapitel werden ICR-Anbieter geprüft, welche potenzielle Mitglieder des ICR-Ensembles sind. Die folgenden OCRs und ICRs sind kompatibel mit Python. Sie stellen eine Python Library als Wrapper zur Verfügung.

5.4.1 Tesseract

Auf dem Markt existieren einige ICR-Anbieter, jedoch bieten die meisten keine Open-Source Lösung an. Tesseract [24] ist eine Open-Source Software, dessen Funktionsumfang als OCR leider nicht für die Arbeit genügt (siehe 3.1 Zeichenerkennung). Sie wurde in der vorherigen Arbeit zur Erkennung der Nummerierung genutzt.

5.4.2 ABBYY Cloud

In der vorherigen Bachelorarbeit wird bereits ein ICR namens ABBYY Cloud eingesetzt. Für die Weiterentwicklung wird sie beibehalten. ABBYY Cloud ist ein OCR/ICR-Anbieter. Es bietet die Erkennung eines bestimmten Bereichs bzw. Textfelds für handgeschriebene Zeichen an. Der Erkennungsbereich kann mit der Eingabe von Koordinaten auf den gewünschten Bereich eingeschränkt werden [19].

Eigenschaft	Beschreibung
Preis	<ul style="list-style-type: none"> • 500 Seiten gratis pro Monat, keine zusätzlichen Seiten • Danach 30\$ pro Monat und 0.06\$ pro Seite
Erkennungsmodus	<ul style="list-style-type: none"> • Ganzseitige Erkennung für gedruckte Zeichen • Textfeld-Erkennung für handgeschriebene Zeichen
Limitationen	<ul style="list-style-type: none"> • Keine ganzseitige Erkennung für handgeschriebene Zeichen • Relativ lange Laufzeit im Vergleich zu anderen ICRs
Sprachunterstützung	<ul style="list-style-type: none"> • Bietet die Einschränkung der Erkennungen durch ein selbstdefiniertes Character-Set und einen Regex-Filter
Weiteres	<ul style="list-style-type: none"> • Pro Bildupload sind mehrere Textfeld-Anfragen möglich

Tabelle 2: Eigenschaften von ABBYY Cloud [19]

5.4.3 Google Vision API

Anders als ABBYY Cloud setzt Google Vision API die Angabe eines Erkennungsbereiches bzw. Textfeldes nicht voraus. Bei Google Vision API ist jedes auf ein Bild anwendbares Feature eine kostenpflichtige Einheit. In dieser Arbeit wird nur das Texterkennungsfeature von Google Vision API eingesetzt [25].

Eigenschaft	Beschreibung
Preis	<ul style="list-style-type: none"> 1000 Einheiten gratis pro Monat, danach 1.50\$ pro 1000 weitere Einheiten
Erkennungsmodus	<ul style="list-style-type: none"> Ganzseitige Erkennung für handgeschriebene und gedruckte Zeichen
Limitationen	<ul style="list-style-type: none"> Keine Textfeld-Erkennung
Sprachunterstützung	<ul style="list-style-type: none"> Bietet die Einschränkung der Erkennungen durch vordefinierte Character-Sets

Tabelle 3: Eigenschaften von Google Vision API [25]

5.4.4 Azure Cognitive Services

Die Eigenschaften der Azure Cognitive Services von Microsoft ähneln der von Google Vision API. Es ist ebenfalls auf die Erkennung eines vollständigen Bildes spezialisiert [26].

Eigenschaft	Beschreibung
Preis	<ul style="list-style-type: none"> 5000 Seiten gratis pro Monat, danach CHF 0.98 pro 1000 weitere Anfragen
Erkennungsmodus	<ul style="list-style-type: none"> Ganzseitige Erkennung für handgeschriebene und gedruckte Zeichen
Limitationen	<ul style="list-style-type: none"> Keine Textfeld-Erkennung
Sprachunterstützung	<ul style="list-style-type: none"> Unterstützt nur romanische Sprachen

Tabelle 4: Eigenschaften von Azure Cognitive Services [26]

5.4.5 Amazon Rekognition

Als letztes werden die Eigenschaften des ICR-Dienstes von Amazon aufgelistet. Speziell bei Amazon Rekognition ist die Limitierung von 50 Erkennungen pro Bildupload. Diese Limitation hat einen negativen Einfluss auf die erhaltenen Erkennungen beim Datenset. Demzufolge birgt das ICR noch Verbesserungspotenzial (siehe 9 Ausblick) [27].

Eigenschaft	Beschreibung
Preis	<ul style="list-style-type: none"> 5000 Seiten gratis pro Monat für ein Jahr, danach 1.20\$ pro 1000 weitere Anfragen
Erkennungsmodus	<ul style="list-style-type: none"> Ganzseitige Erkennung für handgeschriebene und gedruckte Zeichen
Limitationen	<ul style="list-style-type: none"> Keine Textfeld-Erkennung Limitiert auf höchstens 50 Erkennungen pro Bildupload
Sprachunterstützung	<ul style="list-style-type: none"> Erkennungen nur auf Englisch

Tabelle 5: Eigenschaften von Amazon Rekognition [27]

5.4.6 Anwendung des ICR-Ensembles

Das ICR-Ensemble wird sowohl für die Box-Erkennung wie auch die Zug-Korrektur benötigt. Sie verbessert die Resultate dieser Hauptfunktionalitäten. Erkennen die ICRs an einer Position das Gleiche, verstärkt dies die Annahme, dass dieser Wert korrekt ist. Unterscheiden sie sich, besteht weiterhin die Möglichkeit, dass eines der Erkennungen korrekt ist.

5.5 Box-Erkennung

Nach dem Erhalt der Erkennungen vom ICR-Ensemble werden die notierten Schachzüge geortet. Das Ziel der Box-Erkennung ist es, Boxen zu definieren, welche die Schachzüge möglichst einschliessen, ohne weitere Störelemente zu beinhalten. In den folgenden Kapiteln wird die vorherige Implementation der Box-Erkennung analysiert. Anschliessend wird anhand der bestehenden Schwachstellen und Evaluation der Schach-Formulare ein neuer Ansatz definiert, welche die bestehende Lösung ablöst. Die neue Box-Erkennung wird im Laufe der Arbeit angepasst, um alle Spiele im Datenset zu lesen.

5.5.1 Vorherige Box-Erkennung

Die Ausgangslösung erzeugt die Boxen durch die Erkennung der Tabellenkonturen mit OpenCV, jedoch bringt diese Variante der Zug-Erkennung einige Nachteile mit sich. Sie funktioniert nur mit tabellarischen Formularen mit durchgezogenen Linien und ist anfällig für Fehler bei Biegungen in den Tabellenkonturen. Die Horizontalen und Vertikalen der Tabelle müssen also senkrecht zu den Bildrändern stehen, ansonsten funktioniert die Erkennung der Boxen nur teilweise oder gar nicht. Zusätzlich funktioniert die Erkennung nicht bei gestrichelten Tabellenlinien. Mit experimentellen Anpassungen der Kernels und Toleranzen wird versucht, die Box-Erkennung auf die ersten 20 Schach-Formulare des Datensets erfolgreich einzusetzen. Jedoch wird keine Einstellung gefunden, welche auf alle 20 Schach-Formulare gleichzeitig funktioniert. Es werden deshalb zwei verschiedene Konfigurationen der Box-Erkennung eingesetzt, um die 20 Schach-Formulare zu erkennen. Die vorherige Box-Erkennung wird somit als zu statisch eingestuft und eine neue Lösung wird eingeführt.

CHESS SCORE SHEET
<http://www.regencychess.co.uk>

THE REGENCY CHESS COMPANY

WHITE: *Martinez* DATE: *22.04.2019*
 BLACK: *Lochte* EVENT: *Gib Masters*

ROUND: *1* BOARD: WHITE WON DRAW BLACK WON

1	e4	c5	21	g4	Ne3	41		
2	Nf3	e6	22	Qe2	Nxe4	42		
3	g3	Nc6	23	Bf3	Nb6	43		
4	Bg2	g6	24	Rhg1	Kh7	44		
5	d4	Bg7	25	Bxb5	Bb7	45		
6	dxc5	Qa5+	26	Nxf7	gxb5	46		
7	c3	Qxc5	27	Ng5+		47		
8	Bc2	Qb5	28			48		
9	Qc2	Ng7	29			49		
10	Na3	Qa5	30			50		
11	Nc4	Qc7	31			51		
12	0-0	0-0	32			52		
13	Nd6	Ne5	33			53		
14	Nxe5	Bxe5	34			54		
15	Bf4	Nc6	35			55		
16	h4	h5	36			56		
17	Kf1	b6	37			57		
18	Bxe5	Nxe5	38			58		
19	a4	Nc4	39			59		
20	e5	Rxc5	40			60		

© The Regency Chess Company 2012 e-mail: info@regencychess.co.uk <http://www.regencychess.co.uk>

Abbildung 31: Vorherige Box-Erkennung angewendet auf das Schach-Formular «game_1»

5.5.2 Box-Erkennung mit ICR

Um die genannten Probleme zu umgehen und gleichzeitig auch Schach-Formulare ohne Tabellenstrukturen zu erlauben, wird nun ein neuer Ansatz verfolgt. Diese Lösung basiert auf die Hinzunahme von neuen ICRs, welche die ganzseitige Erkennung der Schach-Formulare erlauben. Von den ICRs erhält die Applikation alle erkannten Texte, dazugehören die handgeschriebenen, sowie die maschinengeschriebenen Zeichen. Die ICRs sind normalerweise

nicht in der Lage, zwischen handgeschriebenen und maschinengeschriebenen Texten zu unterscheiden bzw. steht diese Informationen dem Benutzer nicht zur Verfügung (siehe 5.4 ICR-Anbieter). Zu beachten ist ausserdem, dass für alle ICRs eine Parse-Funktion implementiert wird, welche die von den ICRs erhaltenen Daten in eine einheitliche Struktur für die Box-Erkennung übersetzt.

5.5.3 Merkmale eines Schach-Formulars

Um die Zug-Erkennung möglichst für alle Schach-Formulare zu realisieren, werden mehrere Formulare mit unterschiedlichen Layouts analysiert (siehe 5.1.2 Neues Datenset). Aus dem Vergleich der Formulare sind folgende Schlussfolgerungen für die Box-Erkennung zu schliessen.

1. Die Schachzüge sind tabellarisch strukturiert. Ein Schach-Formular beinhaltet 2 oder mehr Tabellen, die nebeneinander platziert sind. Die Tabellen beinhalten die Schachzüge.
2. Eine Tabelle ist unterteilt in zwei Spalten. Die linke Spalte beinhaltet die Spielzüge von Weiss. Die rechte Spalte beinhaltet die Spielzüge von Schwarz.
3. Die Tabellen weisen innerhalb des gleichen Schach-Formulars die gleiche Y-Position, Höhe und Zeilenanzahl auf.
4. Jede Tabelle wird links begleitet von einer Spalte mit der Nummerierung der Züge, die fortlaufend Nummerierungsspalte genannt wird.
5. Der Abstand zwischen zwei aufeinander folgende Zeilen ist konstant.

5.5.4 Erkennung der Nummerierung

Aus diesen Erkenntnissen ist zu schliessen, dass sich die Schachzüge zwischen den Spalten mit den Nummerierungen befinden. Eine Ausnahme ist die letzte Tabelle mit Schachzügen, diese hat entsprechend nur links vorangehend eine Spalte mit Nummern.

Zu beachten ist, dass bei solchen Ortungen Toleranzen benötigt werden. Um die Lösung möglichst auf alle Schach-Formulare anwendbar zu machen, wird generell auf statische Werte verzichtet und relative Toleranzen durch Trial-and-Error ermittelt. Toleranzen dienen in der Box-Erkennung zur Definition der maximal erlaubten Distanz zwischen zwei Punkten.

Suche nach Nummern und Bildung der Spalten

Der Algorithmus sucht als erstes alle erkannten Zahlen. Aus diesen werden nun anhand ihrer X-Positionen Spalten gebildet. Die höchste aller Breiten unter den Erkennungen wird als Toleranz genutzt, um die Spalten zu bilden. Pro Erkennung wird nun geprüft, ob eine Spalte existiert, die innerhalb der Toleranz platziert ist und zu dieser hinzugefügt. Ist dies nicht der Fall, wird eine

neue Spalte mit dieser Erkennung an erster Stelle erstellt. Die Position der Spalte ist durch ihre erste Erkennung definiert. Dies garantiert, dass nur senkrechte Spalten gebildet werden.

CHESS SCORE SHEET				DATE
EVENT			TIME CONTROL	
ROUND	BOARD	SECTION	OPENING	
WHITE			PAIRING NO.	RATING
BLACK			PAIRING NO.	RATING
WHITE	BLACK	WHITE	BLACK	
c4	c5	Nxg6	hxg6	
Be2	d6	h4	Bf8	
f4	Nf6	h5	Ne7	
d3	Nc6	Nxd6	Rc7	
Nf3	a6	hxg6	Nxg6	
O-O	b5	Nxb7	Rxb7	
c3	Bb7	Bf3	Rbb8	
Kh1	e6	Be4	Ve7	
Be3	Be7	Qh3	g8	
Nbd2	Rc8	Bxc5	Nf5	
Bg1	O-O	Bxf8	Rxf8	
Qe2	Qd7	Bxf5	Kg7	
Rd1	Rfd8	Qf3	exf5	
Qg3	Nh5	Kg1	Qe6	
Qg4	Nf6	a3	Qb3	
Qh3	Bf8	Qe2	a5	
e5	Nd5	d4	b4	
Qg3	Nce7	axb4	axb4	
Nh4	Ng6	c4	Rbc8	
Ne4	Be7	Rc1	Rfd8	
CIRCLE CORRECT RESULT				
WHITE WON		DRAW	BLACK WON	
SIGNATURE		SIGNATURE		

Abbildung 32: Bildung der Zahlenspalten auf das Schach-Formular «game_43»

Entfernung der Duplikate

Falls Erkennungen von mehreren ICRs für die Box-Erkennung verwendet werden, sind duplizierte Erkennungen in den Spalten vorhanden. Damit die nächsten Schritte für die Ortung reibungslos funktionieren, werden an dieser Stelle die Duplikate entfernt. Erkannt werden sie durch die Überschneidung ihrer Flächen. Überschneidet die Y-Position des Mittelpunkts einer Erkennung die Fläche einer anderen, sind diese identisch.

Filterung der Spalten

Um die Nummerierung aus den resultierenden Spalten zu erhalten, wird jede Spalte anhand ihrer Y-Position gefiltert. Für die Filterung wird das 5. Merkmal benutzt. Die Erkennungen werden von oben nach unten geprüft, ob ihre Nachbarn innerhalb der Toleranz liegen. Ist dies nicht der Fall, wird die entsprechende Erkennung von der Spalte entfernt. Die Toleranz ist definiert als Median

der Abstände zweier aufeinanderfolgenden Erkennungen. Dies verhindert, dass ungewöhnlich hohe oder tiefe Werte eine negative Wirkung auf die Filterung haben. Diese Filterung wird auf alle Spalten eingesetzt. Alle Spalten mit weniger als 5 Erkennungen werden als nächstes entfernt. Grund dafür ist, dass Beobachtungen zeigten, dass für die Nummerierung immer mehr als 5 Erkennungen vorhanden sind. Dies ergibt Spalten, dessen Erkennungen innerhalb ihrer Spalte mindestens einen gleichmässigen Abstand zu einer anderen Erkennung haben und aus mindestens 5 Erkennungen bestehen.

CHESS SCORE SHEET				DATE	
EVENT				TIME CONTROL	
ROUND	BOARD	SECTION	OPENING		
WHITE			FABRNO	RATING	
BLACK			FABRNO	RATING	
	WHITE	BLACK		WHITE	BLACK
1	c4	c5	21	Nxg6	hxg6
2	Be2	d6	22	h4	Bf8
3	f4	Nf6	23	h5	Ne7
4	d3	Nc6	24	Nxd6	Rc7
5	Nf3	a6	25	hxg6	Nxg6
6	0-0	b5	26	Nxb7	Rxb7
7	c3	Bb7	27	Bf3	Rb6
8	Kh1	e6	28	Be4	Ve7
9	Be3	Be7	29	Qh3	g6
10	Nbd2	Rc8	30	Bxc5	Nf5
11	Bg1	0-0	31	Bxf8	Rxf8
12	Qe1	Qd7	32	Bxf5	Kg7
13	Rd1	Rfd8	33	Qf3	exf5
14	Qg3	Nh5	34	Kg1	Qe6
15	Qg4	Nf6	35	a3	Qb3
16	Qh3	Bf8	36	Qe2	a5
17	e5	Nd5	37	d4	b4
18	Qg3	Nce7	38	axb4	axb4
19	Nh4	Ng6	39	c4	Rbc8
20	Ne4	Be7	40	Rc1	Rfd8

UNKLE CORRECT RESULT
 WHITE WON DRAW BLACK WON

SIGNATURE: *[Handwritten]* SIGNATURE: *[Handwritten]*

Abbildung 33: Entfernung der Duplikate und Filterung der Spalten auf das Schach-Formular «game_43»

Bildung der Nummerierungsspalten

Meist wird das bisherige Resultat nur die gesuchten Spalten mit Nummerierungen sein. Es ist jedoch möglich, dass andere Spalten diese Bedingungen auch erfüllen. Um diese zu entfernen, wird geprüft, ob es Überschneidungen mit Erkennungen gibt, die nicht zur Spalte gehören. Ist dies der Fall, wird die Spalte entfernt, da Spalten mit Nummerierungen aufgrund ihrer Position grundsätzlich keine anderen Erkennungen überschneiden. In der Praxis ist es wiederum möglich, dass eine Zahl als Buchstabe erkannt wird, und somit in diesem Schritt die Nummerierungsspalte

schneidet. Darum wird eine Toleranz von 10 Überschneidungen definiert. Mit dieser Toleranz funktioniert die Box-Erkennung auf dem Datenset. Das Ergebnis der Filterung sind die gesuchten Spalten mit den Nummerierungen.

CHESS SCORE SHEET					DATE
EVENT				TIME CONTROL	
ROUND	BOARD	SECTION	OPENING		
WHITE			PAIRING NO.	RATING	
BLACK			PAIRING NO.	RATING	
	WHITE	BLACK		WHITE	BLACK
1	e4	c5	21	Nxg6	hxg6
2	Bc2	d6	22	h4	Bf8
3	f4	Nf6	23	b5	Ne7
4	d3	Nc6	24	Nxd6	Rc7
5	Nf3	a6	25	hxg6	Nxg6
6	O-O	b5	26	Nxb7	Rxb7
7	c3	Bb7	27	Bf3	Rbb8
8	Kh1	e6	28	Be4	Ve7
9	Be3	Be7	29	Qh3	g6
10	Nbd2	Rc8	30	Bxc5	Nf5
11	Bg1	O-O	31	Bxf8	Rxf8
12	Qe1	Qd7	32	Bxf5	Kg7
13	Rd1	Rfd8	33	Qf3	exf5
14	Qg3	Nh5	34	Kg1	Qe6
15	Qg4	Nf6	35	a3	Qb3
16	Qh3	Bf8	36	Qe2	a5
17	e5	Nd5	37	d4	b4
18	Qg3	Nce7	38	axb4	axb4
19	Nh4	Ng6	39	c4	Rbc8
20	Ne4	Be7	40	Rc1	Rfd8

CIRCLE CORRECT RESULT

WHITE WON DRAW BLACK WON

SIGNATURE

Abbildung 34: Bildung der Nummerierungsspalten auf das Schach-Formular «game_43»

Ein anderer Ansatz für die Ortung der Nummerierungen ist die Bildung von Zahlensequenzen, mithilfe von den ICRs erhaltenen Werten. Dieses Verfahren ist durch die Richtigkeit der ICR-Ausgaben eingeschränkt. Wird zum Beispiel die ursprüngliche Zahl «12» vom ICR missinterpretiert als «1» unterbricht dies die Zahlensequenz und die Spalte wird nicht korrekt erkannt. Auf dem Datenset funktioniert diese Lösung nicht zuverlässig. Selbst wenn einzelne solcher Fälle toleriert werden, trifft dies für Schach-Formulare mit mangelnder Qualität zu oft ein. Grundsätzlich ist somit anzunehmen, dass die erkannten Werte der ICRs zu inkonsistent sind, um sie für die Ortung der Boxen einzusetzen.

5.5.5 Erstellung der Boxen

Nachdem die Positionen der Nummerierungen bekannt sind, ist es eine Leichtigkeit, die Spielzüge daraus zu orten. Aus dem 1. und 4. Merkmal ist festzustellen, dass die Spielzüge zwischen den Nummerierungen positioniert sind, ausser die letzte Tabelle. Diese wird nicht gefolgt von einer weiteren Nummerierung.

Die Schwierigkeit dieser Teilaufgabe ist, keine Spielzüge zu verpassen, selbst wenn die ICRs den Spielzug nicht erkennen. Die Korrektur des Schachspiels funktioniert ansonsten nicht. Darum wird eine Tabelle gebildet, dessen Zellen die jeweiligen Spielzüge beinhalten. Sobald die Tabellenstruktur bekannt ist, ist auch die Reihenfolge der Schachzüge gegeben. Die Tabelle ist durch folgende Angaben definiert:

- Anzahl der Spalten
- Anzahl der Zeilen
- X-Position der Tabelle
- Y-Position der Tabelle
- Breite der Tabelle
- Höhe der Tabelle

Die Anzahl der Spalten ist als einziger Wert für alle Schach-Formulare vorgegeben. Wie im 2. Merkmal beschrieben wird, ist eine Tabelle mit Spielzügen immer in zwei Spalten aufgeteilt. Daraus folgt, dass eine Zeile zwei Spielzüge beinhaltet, einen von Weiss und einen von Schwarz.

Die Anzahl der Zeilen wird bestimmt durch die grösste Anzahl von Erkennungen über alle Nummerierungsspalten. Dieses Maximum wird auf 10 gerundet, weil die Anzahl von Spielzügen pro Tabelle bei allen bekannten Schach-Formularen im Datenset eine Vielzahl von 10 ist (siehe 5.1 Datenset). Die Rundung führt zum korrekten Wert, weil mit einer Toleranz von 5, mindestens einer der Nummerierungsspalten genug Erkennungen enthält, um mit der tatsächlichen Zeilenanzahl übereinzustimmen. Ansonsten ist anzunehmen, dass die Qualität des Bildes zu niedrig ist. Dieses Vorgehen basiert auf der Annahme, dass für die benutzen Schach-Formulare das 3. Merkmal gilt. Unterscheiden sie sich in der Zeilenanzahl funktioniert dieser Ansatz nicht.

Die Rekonstruktion der Tabellen und Zellen werden iterativ für das definierte Datenset optimiert. Daraus entstehen die folgenden aufeinander aufbauenden Lösungsvarianten. Zu beachten ist, dass nur die Hauptversionen vorgestellt werden. Während dem Verlauf der Arbeit sind einige kleinere Optimierungen vorgenommen worden, die hier nicht aufgeführt werden.

Iteration 1: Einfache Tabelle

Die erste Iteration verfolgt den einfachsten Ansatz. Hierbei werden die Dimensionen und die Position der Tabelle durch die umschliessenden Nummerierungsspalten definiert. Die X-Position und Breite der Tabelle werden so gewählt, dass die Tabelle die benachbarten Nummerierungen berührt. Die Y-Position der Tabellen entspricht dem höchsten Punkt der Nummerierungen. Die grösste Höhe der Nummerierungsspalten wird als Höhe der Tabellen definiert. Alle konstruierten Tabellen haben somit die gleiche Y-Position und Höhe, unterscheiden sich aber in der X-Position und Breite. Die Ausnahme ist die letzte Tabelle, welche nicht von beiden Seiten mit Nummerierungsspalten umgeben ist. In diesem Fall wird die gleiche Tabellenbreite wie bei der vorherigen Tabelle benutzt. Die Tabellen werden, anhand der Zeilen- und Spaltenanzahl in gleich grosse Zellen aufgeteilt.

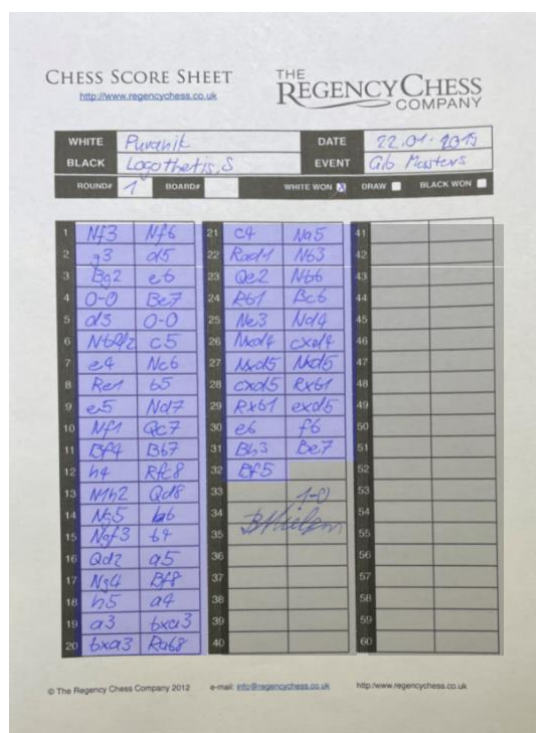


Abbildung 35: Box-Erkennung mit einfachen Tabellen angewendet auf das Schach-Formular «game_1»

Die Probleme dieser Lösung sind im Bild ersichtlich. Die Tabelle inkludiert die Ränder der Tabelle und die Zellen sind verschoben, was die Qualität der Erkennungen bei ABBYY Cloud einschränkt.

Iteration 2: Optimierte Tabelle

In dieser Iteration wird die Tabelle aus der ersten Iteration auf die Erkennungen zwischen den Nummerierungen angepasst. Diese Aufgabe wird als Optimierungsproblem erkannt und als solche gelöst. Dies erlaubt eine elegante Lösung durch Einsatz von bestehenden Algorithmen. Der Nachteil dieser Lösungsvariante ist die verlängerte Laufzeit, weil das Optimum iterativ angenähert wird. Jedoch existiert in diesem Fall kein erzielbares Optimum. Es ist grundsätzlich anzunehmen, dass die optimale Tabelle, welche genau alle Schachzüge umfasst und eine minimale Fläche besitzt, nicht erzielbar ist. Grund dafür ist, dass möglicherweise ein Schachzug oder ein Teil eines Schachzuges von keinem der ICRs erkannt wird und sich gleichzeitig ausserhalb der optimierten Tabelle befindet. Darum wird eine Zwischenlösung gesucht.

Dazu wird ein evolutionärer Algorithmus eingesetzt, der iterativ die Parameter einer Funktion anpasst, um das globale Minimum möglichst anzunähern [28]. Die Funktion des Optimierungsproblems wird wie folgt definiert. Die Parameter der Funktion sind die X-Position, Y-Position, Breite und Höhe der Tabelle. Sie werden vom Algorithmus angepasst, um einen möglichst kleinen Funktionswert zu erzielen. Das Funktionsergebnis ist die Fläche der Erkennungen, die ausserhalb der Tabelle sind, addiert mit der Quadratwurzel der Tabellenfläche. Hierbei wird die Quadratwurzel der Tabellenfläche benutzt, weil sie ansonst einen zu grossen Einfluss auf den Funktionswert hat. Das primäre Ziel ist es, alle Erkennungen mit der Tabelle zu umfassen. Die Minimierung der Tabellenfläche ist zweitrangig. Dem Algorithmus wird für jeden Parameter ein Wertebereich übergeben. Mit einem eingeschränkten Wertebereich sind weniger Versuche nötig, um ein optimiertes Ergebnis zu finden, was wiederum Laufzeit einspart. In der folgenden Tabelle sind die Wertebereiche für die Parameter aufgelistet.

Parameter	Wertebereich	
	Untere Grenze	Obere Grenze
X-Position	$X\text{-Position} - 0.2 * \text{Breite}$	$X\text{-Position} + 0.2 * \text{Breite}$
Y-Position	$Y\text{-Position} - 0.1 * \text{Höhe}$	Y-Position
Breite	$\text{Breite} * 0.8$	$\text{Breite} * 1.2$
Höhe	Höhe	$\text{Höhe} * 1.1$

Tabelle 6: Wertebereiche der Parameter für die Tabellenoptimierung

Die Y-Position wird maximal um 10% der Höhe nach oben verschoben. Die Höhe der Tabelle wird maximal um 10% gestreckt. Eine Verkleinerung und Verschiebung nach unten ist nicht erlaubt, weil die Tabelle durch die strenge Filterung nur zu klein sein kann. Falls Erkennungen in der ersten und letzten Zeile vorhanden sind, werden durch die Optimierung kleine Tabellen auf die korrekte Höhe und Position korrigiert. Die Wertebereiche der X-Position und Breite sind

weniger eingeschränkt, da durch die hohe Zeilenanzahl mehr Daten für die Optimierung vorhanden sind. Die X-Position sowie die Breite können bis zu 20% der Breite in beide Richtungen verschoben werden. Dies führt zu einer Tabelle, dessen seitliche Tabellenränder auch die Erkennungen beinhaltet und eine möglichst kleine Fläche einnimmt.

Abbildung 36: Box-Erkennung mit optimierten Tabellen auf Schach-Formular «game_1»

Die in der Iteration 1 bestimmten Probleme werden in dieser Implementierung weitgehend gelöst. Nur die Linie, welche die Tabelle in 2 Spalten teilt, ist weiterhin in den Boxen vorhanden. In gewissen Schach-Formularen sind die Boxen soweit verschoben, dass ein Teil des nachfolgenden Schachzuges auch in derselben Box liegt. Jedoch ist die Zuweisung der Box zum entsprechenden Schachzug auf dem Schach-Formular trotzdem noch möglich. Dazu wird geprüft, welche Box den grössten Teil der Erkennung beinhaltet.

Iteration 3: Optimierte Tabelle mit ICR-Boxen

Dieser Ansatz nimmt die Boxen, die aus der optimierten Tabelle entstehen und prüft für jede Box, ob innerhalb eine ICR-Erkennung vorhanden ist. Ist keine vorhanden, wird die Box so belassen. Ist wiederum eine vorhanden, ersetzt die Box der ICR die bestehende Box. Es ist möglich, dass eine Box mehrere ICR Erkennungen beinhaltet, besonders wenn mehreren ICRs benutzt werden. In solchen Fällen werden die Boxen zusammengeführt, um eine grössere Box zu bilden.

CHESS SCORE SHEET http://www.regencychess.co.uk THE REGENCY CHESS COMPANY

WHITE	Purcell	DATE	22.04.2019
BLACK	Logothetis, S	EVENT	G6 Masters
ROUND	1	BOARD	
		WHITE WON	<input checked="" type="checkbox"/>
		DRAW	<input type="checkbox"/>
		BLACK WON	<input type="checkbox"/>

1	Nf3	Nf3	21	e4	1d5	41	
2	e3	d5	22	Rc4	Nc3	42	
3	Bc2	e6	23	d2	Nb6	43	
4	O-O	Bc7	24	Rd1	Bc6	44	
5	e5	O-O	25	Nc3	Nd4	45	
6	Nb9	e5	26	Nc4	exd6	46	
7	e4	Nc6	27	Nc5	Nc6	47	
8	Rc1	e5	28	exd6	Rc6	48	
9	e5	Nd7	29	Rx5d1	exd6	49	
10	Nf4	Qe7	30	e6	f6	50	
11	Bd4	Bd7	31	Bc3	Bc7	51	
12	h4	Rd7	32	Bf5		52	
13	Nh2	Qd8	33		f-d	53	
14	Nc5	h6	34			54	
15	Nf3	h7	35			55	
16	Qd2	Q5	36			56	
17	Nd4	Bd8	37			57	
18	h5	h4	38			58	
19	g3	hxc3	39			59	
20	bxc3	Rc6	40			60	

© The Regency Chess Company 2012 e-mail: info@regencychess.co.uk http://www.regencychess.co.uk

Abbildung 37: Box-Erkennung mit optimierten Tabellen und ICR-Boxen auf Schach-Formular «game_1»

Diese Lösung bildet die optimalen Boxen für die weitere Erkennung mit ABBYY Cloud. Jedoch nur für die Boxen, die eine Erkennung beinhalten. Wird kein Schachzug innerhalb der Box erkannt, wird das Resultat für diese Box nicht verbessert. Hinzukommt das selbst durch die Nutzung mehrere ICRs nicht sichergestellt ist, dass der komplette Schachzug innerhalb der Box erkannt wird. Sind Teile des Schachzuges in einer unerkennlichen Handschrift notiert, hilft diese Technik nicht weiter.

Iteration 4: Optimierte Tabelle und Boxen

Die letzte und aktuelle Implementation der Box-Erkennung optimiert die Boxen mit dem evolutionären Algorithmus. Die Boxen nähern sich dadurch den Erkennungen an, erreichen diese jedoch nicht ganz. Die ursprüngliche Box verschiebt sich und passt sich so an, dass die Erkennung immer vollständig von der Box umfasst wird. Wie bereits für die Tabelle werden die gleichen Parameter und die gleiche Optimierungsfunktion verwendet. Nur die Wertebereiche werden neu definiert.

Parameter	Wertebereich	
	Untere Grenze	Obere Grenze
X-Position	X-Position	X-Position + 0.2 * Breite
Y-Position	Y-Position - 0.2 * Höhe	Y-Position + 0.2 * Höhe
Breite	Breite * 0.8	Breite
Höhe	Höhe * 0.8	Höhe * 1.1

Tabelle 7: Wertebereiche der Parameter für die Box-Optimierung

Die Box wird nicht weiter nach links verschoben, da sie aufgrund der vorherigen Schritte bereits links von der Erkennung positioniert ist. Eine Verschiebung in die rechte Richtung um maximal 20% der Breite ist jedoch erlaubt und wird meist benötigt, um die Erkennung anzunähern. Die Breite wird ebenfalls nicht erhöht, weil die initiale Breite in jedem Fall grösser als die Erkennung ist. Verkleinert werden darf sie bis zu 20%. Vertikal wird der Optimierung grössere Änderungen an der Box erlaubt. Die Y-Position darf wiederum um maximal 20% der Höhe vergrössert und verkleinert werden. Die Höhe kann um den gleichen Wert verkleinert werden, jedoch nur um 10% vergrössert. Der Grund ist derselbe wie für die Breite, nur wird eine 10% Toleranz eingebaut, um Sonderfälle, in welchen die initiale Box kleiner ist als die Gesuchte, abzufangen.

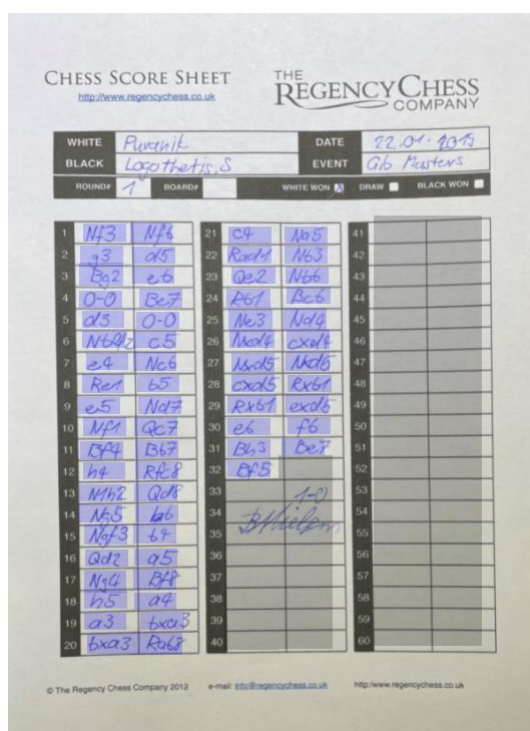


Abbildung 38: Box-Erkennung mit optimierten Tabellen und Boxen auf Schach-Formular «game_1»

5.5.6 Weitere Verbesserungen

Die folgenden Verbesserungen sind in der aktuellen Box-Erkennung vorhanden und wurden in den bisherigen Kapiteln nicht erklärt.

- Erkennungen, welche über mehrere Schachzüge oder Nummerierungen spannen, werden in mehrere Boxen aufgeteilt. Dazu wird geprüft, ob die Erkennungen einen Tabellenrand oder die Mittellinie der initialen Tabelle schneiden. Ist dies der Fall, werden die entsprechenden Erkennungen geteilt.
- Die Boxen werden iterativ von oben nach unten durch den in Iteration 4 beschriebenen Algorithmus optimiert. Zusätzlich wird nun die Y-Position jeder Box durch die optimierte Höhe und Y-Position der Box oberhalb angepasst, bevor diese selbst optimiert wird.
- ICR-Erkennungen werden zusätzlich gefiltert. Erkennungen, die nur aus folgenden Zeichen bestehen, werden entfernt: «|, l, L, |». Die ICR verwechseln die Tabellenlinien oft mit diesen Zeichen.

5.5.7 Verwendung der Boxen

Die erzeugten Boxen werden wie auch schon in der vorherigen Applikation für ABBYY Cloud benötigt. Dazu werden die Koordinaten der Boxen mit dem Schach-Formular an ABBYY Cloud gesendet, um die Schachzüge innerhalb der Boxen zu erkennen [4].

Zusätzlich werden die Boxen nun auch für die Zuweisung der restlichen ICR-Erkennungen benötigt. Dafür wird ein simpler Algorithmus implementiert, welche für jede Erkennung prüft ob sie eine der Boxen schneidet. Ist dies der Fall wird die Erkennung der nächstgelegenen Box zugewiesen. Falls die ICR-Anbieter für die Erkennungen die Koordinaten pro Symbol zur Verfügung stellen, werden diese benutzt, um eine möglichst präzise Zuweisung zu erhalten. Google Vision API bietet dies im aktuellen ICR-Ensemble als einziger an.

5.6 Zug-Korrektur

5.6.1 Confusion-Modul

Nun da wir für jeden ICR-Anbieter die erkannten Zeichen erhalten, ist das Ziel, daraus ein gültiges Schachspiel nachzuspielen. Ein Problem mit der Ausgabe der ICRs ist, dass sie selten genau den richtigen Zug im richtigen Format erkennen. So wird beispielsweise sehr häufig ein „a“ für ein „g“ verwechselt, weil das „g“ über den Rand der Zelle des Schach-Formulars ragt.

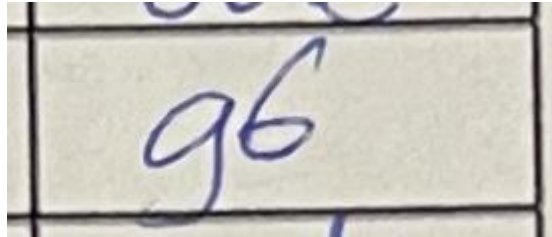


Abbildung 39: Beispiel für Verwechslungsgefahr mit „a“ und „g“

Ein anderes häufig beobachtetes Beispiel für eine Verwechslung ist „b“ mit „h“ und umgekehrt, weil je nach Handschrift beispielsweise das „b“ unten nicht ganz abgeschlossen wird.

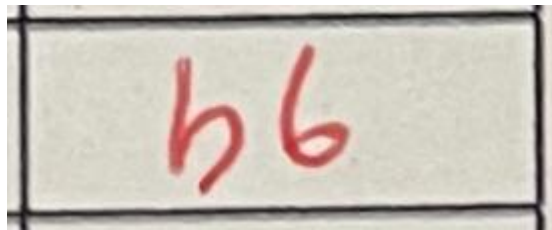


Abbildung 40: Beispiel für Verwechslungsgefahr mit „b“ und „h“

Das Ziel des Confusion-Moduls ist es nun, diese Verwechslungen zu verstehen und relevante Informationen über die typischen Verwechslungen an das Confidence-Modul weiterzugeben.

Eine gute Methode, um einen Überblick für die Verwechslungen zu erhalten, ist die Confusion-Matrix. Sie zeigt für jedes gültige Zeichen, wie oft es mit jedem anderen Zeichen verwechselt wird. Dabei zeigt die vertikale Achse das gewünschte Zeichen, und die horizontale Achse stellt dar, welches Zeichen vom ICR wirklich erkannt wurde. Bei einer perfekten Erkennung hätten alle Zellen, ausser die in der Diagonale, den Wert 0.

	+	-	1	2	3	4	5	6	7	8	=	B	K	N	O	Q	R	a	b	c	d	e	f	g	h	x
#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
+	51	0	0	0	0	4	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
-	0	75	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	115	0	0	2	0	0	0	0	0	0	0	0	0	0	0	4	0	0	7	0	0	0	1	0
2	0	0	0	179	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	1	0	0	0	0
3	0	0	1	1	389	0	0	0	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0
4	0	0	0	0	0	407	0	0	0	0	0	0	0	0	0	0	0	15	0	0	1	9	0	0	5	0
5	0	0	0	0	3	0	366	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
6	0	0	0	0	0	0	0	315	0	0	0	1	0	1	0	0	0	58	0	2	5	1	1	0	0	0
7	0	0	0	1	0	0	0	0	198	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	0	156	0	1	0	0	1	1	0	2	0	2	5	4	2	1	0	0
=	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	8	0	2	3	0	4	0	474	0	0	0	0	3	0	1	0	0	0	0	0	0	0
K	0	0	1	0	0	0	0	0	0	0	0	0	96	0	0	0	0	0	1	0	0	0	0	0	0	0
N	0	0	3	0	0	0	0	1	0	0	0	0	0	534	0	0	0	0	0	0	0	0	0	0	0	1
O	0	0	2	0	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0
Q	0	0	0	2	0	0	0	4	0	0	0	2	0	0	7	321	0	13	0	0	0	0	0	0	0	0
R	0	0	1	1	0	0	0	0	0	1	0	0	1	0	0	0	364	1	0	0	0	2	0	0	0	0
a	0	1	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	205	0	0	5	0	0	0	0	0
b	0	0	2	1	2	0	2	201	0	0	0	0	0	0	2	0	0	0	105	0	2	0	0	0	0	0
c	0	0	3	1	0	0	0	9	0	0	0	0	0	0	1	0	0	3	0	332	0	24	0	0	0	0
d	0	0	7	4	0	0	0	0	0	0	0	0	0	0	1	0	0	26	0	3	516	5	0	0	0	0
e	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	531	0	0	0	1
f	10	0	5	1	0	16	0	9	5	0	0	0	0	0	0	0	0	0	0	0	0	6	293	0	0	0
g	0	0	0	0	7	2	0	2	0	1	0	0	0	0	0	1	0	14	1	0	0	8	0	156	0	0
h	0	0	4	2	1	6	5	11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	150	0
x	1	0	1	0	0	1	0	5	0	1	0	0	0	0	0	1	0	3	0	0	0	3	0	0	0	534

Abbildung 41: Confusion-Matrix mit Google Vision API als ICR

Das Confusion-Modul an sich ist so aufgebaut, dass es als Basis nun diese Confusion-Matrix über das ganze Datenset pro ICR berechnet. Aus technischen Gründen wird aber keine Matrix, sondern eine Dictionary benutzt, weil die Matrix meist zu dünnbesetzt ist, also fast nur 0-Werte enthält (siehe Abbildung 41: Confusion-Matrix mit Google Vision API als ICR). Die Schlüsselwerte bilden sich dann aus den Verwechslungen, und der eigentliche Wert ist die Anzahl der Verwechslungen. Dabei ist zu beachten, dass nur die ICR-Ausgaben evaluiert werden, welche die gleiche Länge haben wie der echte Zug.

```
'eg' : 5  
'32' : 70  
'QB' : 38  
'xa' : 2  
'ec' : 20  
'ag' : 6
```

Abbildung 42: Struktur für Verwechslungen

Mit den Verwechslungen wird nun für jedes mögliche Zeichen ein Wert gesucht, der aussagt, wie relativ gesehen wahrscheinlich es ist, dass eine Verwechslung vorliegt. Dafür wird zuerst das Verwechslungsverhältnis ermittelt.

$$\text{Verwechslungsverhältnis} = \frac{\text{Anzahl Verwechslungen}}{\text{Anzahl richtiger Erkennungen}}$$

Für die Berechnung ist auch noch der Parameter α , auch Minimum-Confusion-Value genannt, nötig. Er sagt aus, wie stark man standardmässig schon davon ausgeht, dass eine Verwechslung wahrscheinlich ist. Der Verwechslungswert berechnet sich dann wie folgt:

$$\text{Verwechslungswert} = (1 - \alpha) * \text{Verwechslungsverhältnis} + \alpha$$

Der Parameter α muss dabei grösser als 0 und kleiner als 1 sein und sorgt so auch dafür, dass es keine Probleme mit der Multiplikation/Division durch 0 gibt bei der weiteren Verarbeitung.

Wenn man nun diese Berechnung für alle möglichen Zeichen durchführt, erhält man eine Liste, die für jeden ICR-Anbieter individuell aussagt, welche Verwechslungen relativ zu allen anderen Verwechslungen wie wahrscheinlich ist. Es folgt eine Tabelle mit den Werten für den ICR-Anbieter Google Vision API, generiert aus dem Datenset mit α auf 0.35 gesetzt.

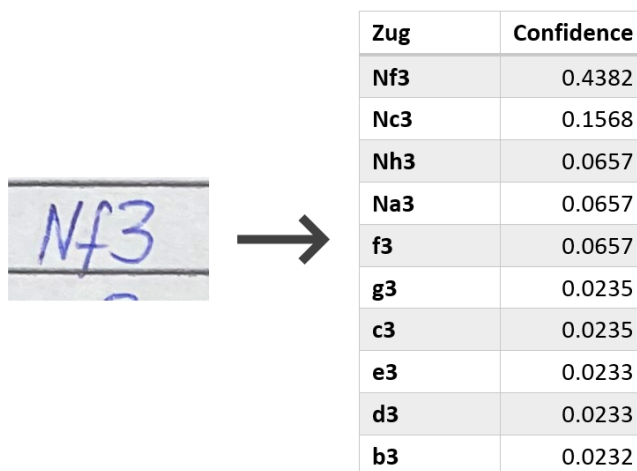
Verwechslung	Relative Häufigkeit
O0	9.94
b6	1.55
Oo	1.24
#x	1.00
+t	0.83
Kk	0.60
g9	0.57
5s	0.52
OI	0.51
O1	0.51
cC	0.48
=r	0.48
6b	0.47

Tabelle 8: Verwechslungshäufigkeiten für Google Vision API

5.6.2 Confidence-Modul

Das Confidence-Modul verwendet die Ausgabe des Confusion-Moduls, um jeweils für einen legalen Schachzug und einer ICR-Ausgabe zu sagen, wie nahe sich diese visuell sind.

Die Ausgabe des Moduls ist eine Zahl über 0 und kleiner-gleich 1, welche aussagt, wie stark die ICR-Ausgabe dem legalen Schachzug visuell ähnelt. Diese Ausgabe wird als Confidence bezeichnet, das Confidence-Modul gibt dann die Confidence des höchsten Zug-Kandidaten zurück. Im folgenden Bild wäre das die Confidence 0.4382 mit Prediction «Nf3».



Zug	Confidence
Nf3	0.4382
Nc3	0.1568
Nh3	0.0657
Na3	0.0657
f3	0.0657
g3	0.0235
c3	0.0235
e3	0.0233
d3	0.0233
b3	0.0232

Abbildung 43: Confidence für Kandidaten eines Zuges mit Google Vision API als ICR

Für die Bestimmung der Confidence gibt es drei mögliche Fälle, die ICR-Ausgabe ist entweder kleiner, grösser oder gleich lang wie der gegebene legale Schachzug-Kandidat. Ist die Länge der ICR-Ausgabe nicht identisch mit dem legalen Schachzug, wird sie so angepasst, dass die Länge wieder gleich ist, dabei muss aber der Confidence-Wert reduziert werden, weil dies eine falsche Erkennung darstellt.

Hat die ICR-Ausgabe die gleiche Länge wie der legale Schachzug, wird Zeichen für Zeichen ein Produkt gebildet. Die Faktoren in diesem Produkt werden vom Confusion-Modul genommen. Ist für eine gegebene Verwechslung kein Wert im Confusion-Modul vorhanden, wird für den Faktor der Parameter Minimum-Confidence-Value eingesetzt, dieser Parameter legt also fest, wie stark die Confidence-Einbusse ist, wenn eine noch ungesehene Verwechslung vorliegt, und verhindert gleichzeitig das Multiplizieren/Dividieren mit 0.

Die Längen-Anpassung der ICR-Ausgabe wird so realisiert, dass für jede mögliche Kombination mit der korrigierten Länge die Confidence berechnet wird, und von diesen Kandidaten die höchste Confidence gewählt wird. Ist die ICR-Ausgabe kleiner, wird das Zeichen «x» für alle möglichen Kombinationen eingefügt, im Fall wo die ICR-Ausgabe grösser ist, werden an allen möglichen Positionen Zeichen rausgeschnitten. Die Wahl für «x» als Ersatzzeichen wurde bestimmt, weil häufig das «x» bei Zügen nicht geschrieben bzw. nicht erkannt wird, obwohl es vorhanden ist.

Ist der Zug beispielsweise «Bxb4» und die ICR-Ausgabe «Bb», geht das Modul alle Kombinationen durch. Das wären in diesem Fall:

Kombination	Kandidat-Ausgabe
0011	Bbxx
0101	Bxbx
0110	Bxxb
1001	xBbx
1010	xBxb
1100	xxBb

Tabelle 9: Beispiel für Kombinationen der ICR-Ausgabe «Bb» für Länge 4

Die maximale Confidence für alle Fälle dieses Beispiels ist «Bxbx», also wird dieser Fall als neue Ausgabe genommen, dabei wird für jedes eingefügte Zeichen die Confidence mit einem Parameter multipliziert, der kleiner als 1 ist. Dieser Parameter heisst Length-Penalty und hat einen Wert von 0.15 für dieses Beispiel. Es wird also die Confidence aus dem legalen Zug «Bxb4» und der neuen ICR-Ausgabe «Bxbx» genommen, und zweimal mit dem Length-Penalty multipliziert («Confidence neue Ausgabe» * 0.15 * 0.15), was die schlussendliche Confidence stark reduziert.

Ist die ICR-Ausgabe länger als der legale Schachzug, passiert dasselbe wie beim Beispiel oben, es werden statt «x» an der Stelle eingefügt die Zeichen rausgeschnitten. Das Multiplizieren mit der Length-Penalty bleibt gleich.

Ist also der legale Schachzug «d4» und die ICR-Ausgabe «dle», werden die Möglichkeiten «dl», «de» und «le» berechnet. Die höchste Confidence für diesen Fall ist «de», nun wird also «de» als neue Ausgabe genommen.

5.6.3 Ensembling der ICR-Ausgaben

Bisher wurde beschrieben, wie die Confidence ermittelt wird, für welche die Ausgabe eines einzigen ICR-Anbieters genutzt wird. Als nächsten Schritt wird versucht, die Ausgaben der gewünschten ICR-Anbieter sinnvoll zusammenzuführen, so dass mittels diesem Ensembling ein Mehrwert entsteht.

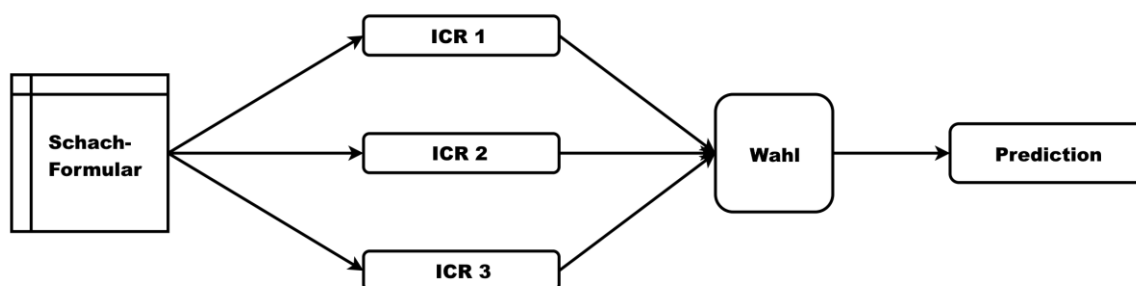


Abbildung 44: Ensembling-Modell für Erkennung des Schach-Formulars

Eine Weise, die Ausgaben der ICR-Anbieter zusammen zu nutzen, ist durch eine einfache Majority-Vote. Der Zug, bei welchem die meisten ICR-Ausgaben übereinstimmen, wird genommen. Dieses Vorgehen hat aber einige Nachteile, zum einen gibt es kein System, bei welchem man besseren ICRs der Entscheidung mehr Gewicht zuordnen kann. Zum anderen wird völlig ignoriert, was die ICRs von den anderen Zug-Kandidaten halten, die nicht ihr Favorit ist.

Eine einfache Verbesserung ist ein System mit einer gewichteten Wahl, bei dem jedem ICR ein Gewicht relativ zu den anderen ICRs zugeteilt wird. Es wird dann der Zug als Prediction zurückgegeben, der die höchste gewichtete Summe aller ICRs hat. Für die Bestimmung der Gewichte können verschiedene Vorgehensweisen dienen: Manuelle Evaluierung, ein Optimierungsalgorithmus oder verschiedene AI-Algorithmen.

Was jetzt noch in die Entscheidung miteinflusst, ist die Confidence aller Züge jeder ICR. Die Confidence wurde bisher nur ermittelt, um die beste Prediction pro ICR einzeln zu finden. Die Idee ist nun aber, die Confidence aller Zug-Kandidaten pro ICR zusammenzuführen, bevor eine Prediction gemacht wird. Es werden also für jeden Zug mehrere Confidence-Werte pro Kandidaten-Schachzug, der in der Position möglich ist, miteinbezogen.

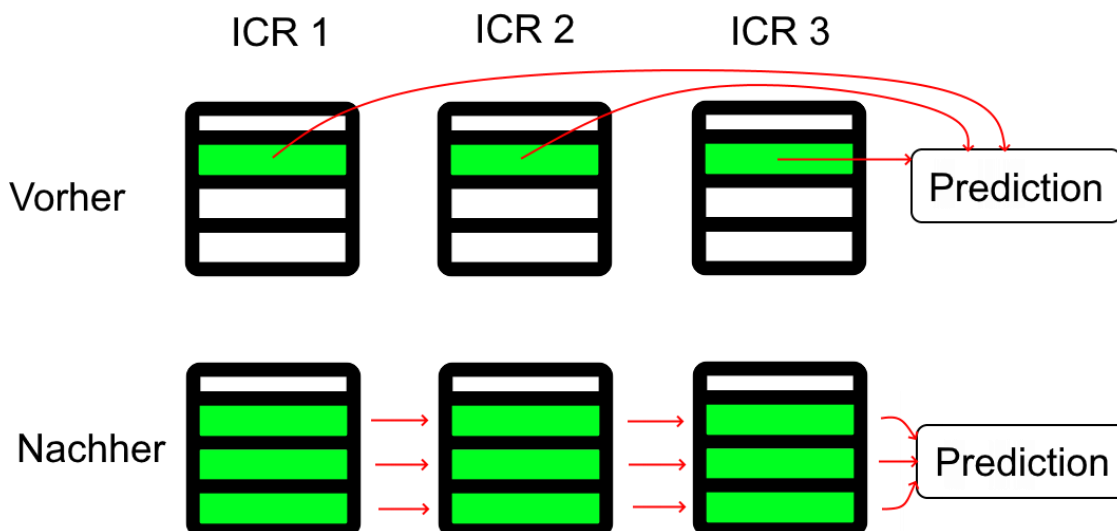


Abbildung 45: Illustration zur Einbeziehung aller Kandidaten pro ICR

In diesem Zusammenhang gibt es wieder mehrere Wege, die Confidence-Werte pro Kandidaten-Zug zu vereinen. Eine Methodik, die schon beim vorherigen Vorgehen nur für die jeweiligen Top-Kandidaten eingesetzt wurde, kann auch für den neuen Mechanismus übernommen werden. Diesmal wird eine gewichtete Summe für jeden Kandidaten einzeln gebildet, als Prediction nimmt man dann den Kandidaten, der die höchste gewichtete Summe erzielt.

Diese Idee hat bei den Tests auch die besten Resultate bezüglich der Precision-Score erzielt. Bei weiteren Experimenten kam jedoch eine noch bessere Variante auf: Statt die gewichtete Summe zu bilden, werden die Confidence-Werte pro Kandidat als Faktoren angesehen und miteinander multipliziert. Bei der Multiplikation ist die Precision-Score generell tiefer als mit einer gewichteten Summe, aber die Fall-Verteilung der Züge für den Skipping-Algorithmus (siehe 5.6.5 Skipping-Algorithmus) ist stark verbessert (siehe 7.3 Zug-Korrektur).

Es gibt noch eine kleine Optimierung, welche die Ausgabe verbessert. Ist die ICR-Ausgabe direkt identisch mit einem legalen Zug, wird der Parameter Direct-Hit-Bonus-Factor, der über 1 ist, mit der Confidence dieses Kandidaten-Zugs multipliziert. Ist die Ausgabe also direkt schon ein legaler Zug, wird der ICR-Ausgabe ein grösseres Gewicht gegeben.

5.6.4 Corrector-Algorithmus

Der Corrector-Algorithmus ist der Hauptbestandteil für die Zug-Korrektur, er verwendet das mit Ensembling erweiterte Confidence-Modul, um das Schachspiel zusammen mit den ICR-Ausgaben nachzuspielen.

Der Kern Bestandteil vom Corrector-Algorithmus ist der Spielbaum, bei welchem jeder Knoten eine mögliche Schachposition und jede Kante einen Zug darstellt.

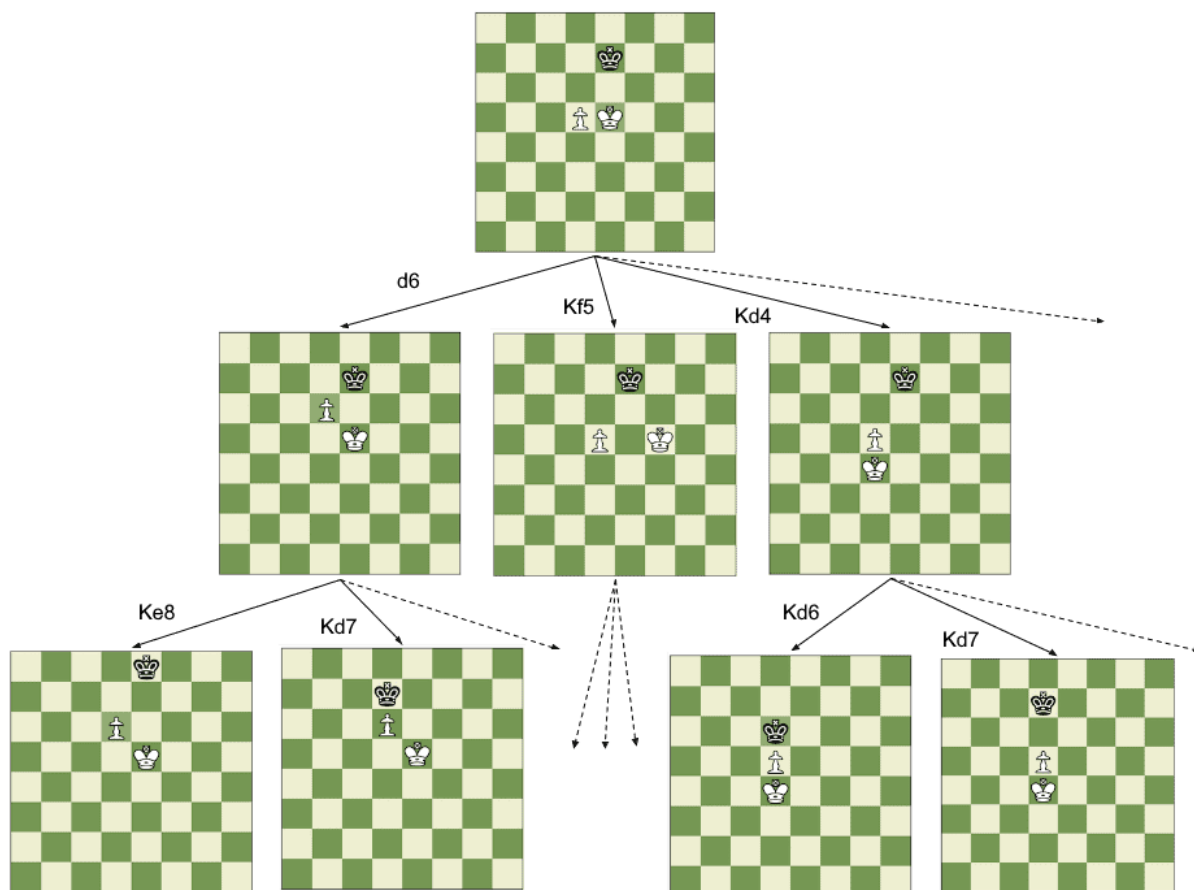


Abbildung 46: Visualisierung des Schach-Spielbaums [29]

Die Aufgabe des Corrector-Algorithmus ist nun, einen Pfad bzw. Folge von Zügen durch diesen Spielbaum zu finden, der visuell so ähnlich wie möglich wie die ICR-Ausgaben aussieht. Sind also die ICR-Ausgaben gut genug, sollte auch der gefundene Pfad sehr nahe beim echten Spielpfad liegen.

Was schnell klar wurde ist, dass beim Traversieren des Spielbaums sehr stark Äste abgeschnitten werden müssen (Pruning), um dem Benutzer in Echtzeit Resultate zu liefern. Dies liegt auch daran, dass der durchschnittliche Branching-Factor für das Schachspiel bei ca. 20 liegt, je nach Spielverlauf kann dieser Branching-Factor aber sehr stark variieren.

Die bereits vorhandene Version von Very Chess benutzte auch einen Spielbaum, der die Depth-Limited-Search verwendet, bei dieser Suchvariante musste aber die maximale Tiefe auf 3 gesetzt werden, um schnell genug ein Resultat zu erhalten. Der Algorithmus der ersten Version hat dann alle Pfade der Länge 3 nachgeschaut und eine Evaluationsfunktion darauf angewendet. Dann wurde der Pfad mit der höchsten Evaluation gewählt [4].

Problematisch war an diesem Vorgehen zum einen, dass die Tiefe 3 häufig keinen Mehrwert bringt, da im Schach die Zusammenhänge meist längere Zugfolgen umfassen. Eine Erkenntnis daraus ist, dass es wahrscheinlich besser ist, den Corrector-Algorithmus gierig zu gestalten, um zeitnah ein Resultat zu liefern.

Zum anderen war die Evaluationsfunktion nicht geeignet, denn diese basierte nur auf standardmässigen String-Vergleichen wie beispielsweise die Levenshtein-Distanz. Diese String-Vergleiche haben aber kein Konzept davon, wie visuell ähnlich sich die Strings sind.

Der Corrector-Algorithmus funktioniert also so, dass er für jeden Knoten nur die nächste Tiefe evaluiert. Dabei wird als Evaluationsfunktion für jede Kante die Confidence mithilfe des Confidence-Moduls berechnet. Das Confidence-Modul erhält als Eingabe die Kante, also einen legalen Schachzug, und die ICR-Ausgabe für diese Tiefe. Wie erwähnt wählt der Algorithmus dann einfach die Kante mit der höchsten Confidence. Dies geht rekursiv so weiter bis das Spiel endet oder die Tiefe des letzten erkannten Zuges erreicht wird.

Ein Schwachpunkt dieses Verfahrens ist, dass gerade wegen der gierigen Gestaltung, der Corrector-Algorithmus sich bei einem Fehler rasch vom echten Spielpfad entfernt, ohne einen Mechanismus, um wieder auf den die richtige Lösung zu kommen.

Deswegen kam die Entscheidung, dass der Benutzer den Pfad des Schachspiels zusammen mit dem Corrector-Algorithmus aufbaut. Die Aufgabe des Benutzers ist es dann, die gewählten Züge vom Algorithmus zu überprüfen, damit der Pfad frühzeitig korrigiert wird, und so der Algorithmus wiederum dem Benutzer bessere Predictions bietet.

Bei dieser Vorgehensweise ist aber ein grosser Nachteil, dass der Benutzer jeden Schachzug anschauen, überprüfen und gegebenenfalls korrigieren muss. Die Digitalisierung des Schach-Formulars ist so also weniger automatisiert.

5.6.5 Skipping-Algorithmus

Wie erwähnt, muss der Benutzer mit dem Corrector-Algorithmus jeden Zug einzeln durchgehen, der Skipping-Algorithmus soll bei diesem Problem Abhilfe schaffen. Hierbei wird ausgenutzt, dass das Confidence-Modul eine glatte Ausgabe zwischen 0 und 1 zurückgibt.

Bei Zügen, wo die Confidence wertemässig über eine bestimmte Schwelle landet, wird hier der Benutzer nicht mehr danach gefragt, den Zug zu überprüfen. Die Begründung dafür ist, dass die Wahrscheinlichkeit, dass ein Zug trotz einer hohen Confidence falsch ist, sehr gering ist. Diese Schwelle ist definiert als der Parameter Skip-Threshold.

Mit dieser Methodik gibt es also vier verschiedene Fälle:

- Skipped: Zug wird übersprungen und stimmt überein
- False-Positive: Zug wird übersprungen und ist nicht korrekt
- Validated: Zug wird nicht übersprungen und validiert durch den Benutzer
- Corrected: Zug wird nicht übersprungen und korrigiert durch den Benutzer

Man nimmt also in Kauf, dass es im schlimmsten Fall auch Züge gibt, die wegen der hohen Confidence übersprungen werden, aber trotzdem nicht korrekt sind, diese Fälle heissen False-Positives. Tritt ein False-Positive auf, muss es der Benutzer selbst merken und die vorherigen Züge nochmal manuell überprüfen, bis er den fehlerhaften Zug gefunden und korrigiert hat, dieser Vorgang ist also meist mühevoll und zeitaufwändig. Ein False-Positive-Fall ist also sehr ungünstig.

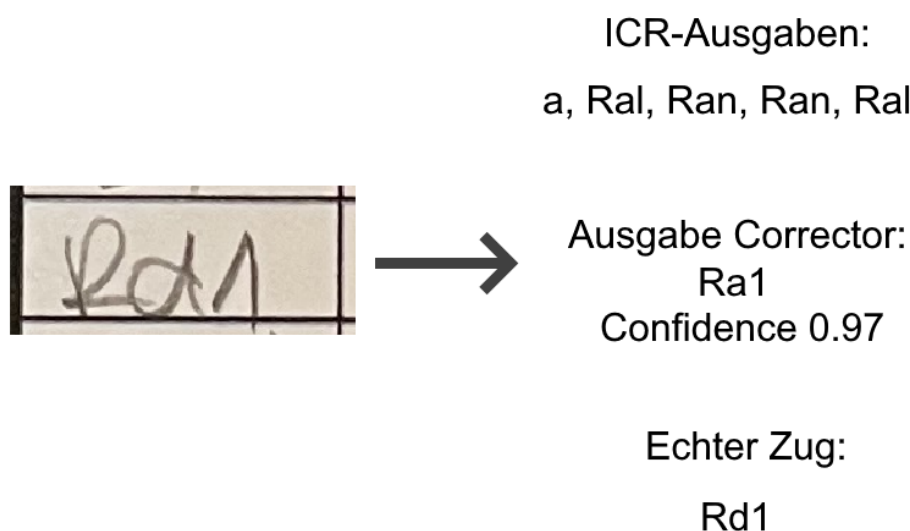


Abbildung 47: Beispiel für False-Positive-Fall (Confidence Hoch, Prediction aber falsch)

Möglich ist aber auch, vor allem bei schöner Handschrift, dass alle Züge wegen hoher Confidence übersprungen werden und korrekt sind, in diesem Fall wird das Spiel vollautomatisch digitalisiert. Folgend ist ein Schach-Formular, bei welchem der Vorgang perfekt funktioniert.

SCORE SHEET					DATE (DD.MM.YYYY)	
EVENT <i>Farbenfroh & Lang</i>					TIME CONTROL	
ROUND	BOARD	SECTION	OPENING (ECO CODE)			
WHITE			PAIRING NO.	RATING		
BLACK			PAIRING NO.	RATING		
	WHITE	BLACK		WHITE	BLACK	
1	<i>e4</i>	<i>c6</i>	21	<i>Bc2</i>	<i>Bc7</i>	
2	<i>d4</i>	<i>d5</i>	22	<i>Ne4</i>	<i>Nb8</i>	
3	<i>Nd2</i>	<i>dxe4</i>	23	<i>Nd6+</i>	<i>Bxd6</i>	
4	<i>Nxe4</i>	<i>Nd7</i>	24	<i>Rxd6</i>	<i>Nd7</i>	
5	<i>Bc4</i>	<i>Ngf6</i>	25	<i>Rhd1</i>	<i>Bxf3</i>	
6	<i>Ng5</i>	<i>e6</i>	26	<i>gxh3</i>	<i>Nxe5</i>	
7	<i>Qe2</i>	<i>Nb6</i>	27	<i>Bxd4+</i>	<i>Ke7</i>	
8	<i>Bb3</i>	<i>a5</i>	28	<i>Bb5</i>	<i>Rab8</i>	
9	<i>a3</i>	<i>a4</i>	29	<i>f4</i>	<i>Nf3</i>	
10	<i>Ba2</i>	<i>h6</i>	30	<i>Rd7+</i>	<i>Kf6</i>	
11	<i>N5f3</i>	<i>c5</i>	31	<i>f5</i>	<i>Nxh2</i>	
12	<i>Bf4</i>	<i>Nbd5</i>	32	<i>fxe6</i>	<i>fxe6</i>	
13	<i>Be5</i>	<i>Qa5+</i>	33	<i>Bc6</i>	<i>Ng4</i>	
14	<i>Qd2</i>	<i>Qxd2+</i>	34	<i>f4</i>	<i>Ne3</i>	
15	<i>Nxd2</i>	<i>Ng4</i>	35	<i>Rg1</i>	<i>g5</i>	
16	<i>Ngf3</i>	<i>Nxe5</i>	36	<i>fxg5+</i>	<i>bxg5</i>	
17	<i>dxe5</i>	<i>b6</i>	37	<i>Re1</i>	<i>Nf5</i>	
18	<i>c4</i>	<i>Ne7</i>	38	<i>Be4</i>	<i>Rbd8</i>	
19	<i>O-O-O</i>	<i>Bb7</i>	39	<i>Rb7</i>	<i>Rd6</i>	
20	<i>Bb1</i>	<i>Nc6</i>	40	<i>b4</i>	<i>Rh3</i>	

CIRCLE GAME RESULT: **WHITE WON** **DRAW** **BLACK WON**

Signature (White) Signature (Black) Signature (Arbiter)

Abbildung 48: Schach-Formular, das vollautomatisch digitalisiert wird

Je nachdem wie hoch der Skip-Threshold gewählt wird, ändert sich natürlich auch die Verteilung der vier beschriebenen Fälle. Der Skip-Threshold wird so balanciert, dass es möglichst viele Skipped-Fälle gibt, ohne dass ein zu hoher Anteil False-Positives auftritt. So wird die Benutzerfreundlichkeit verbessert und das Schach-Formular schneller digitalisiert (siehe 7.4 Zeitersparnis).

6 Zusätzliche Änderungen

6.1 Benutzeroberfläche

Folgend wir gezeigt, wie die aktualisierte Applikation Very Chess als Ganzes, zusammen mit der Benutzeroberfläche, hilft, das Schach-Formular eines Benutzers zu digitalisieren.

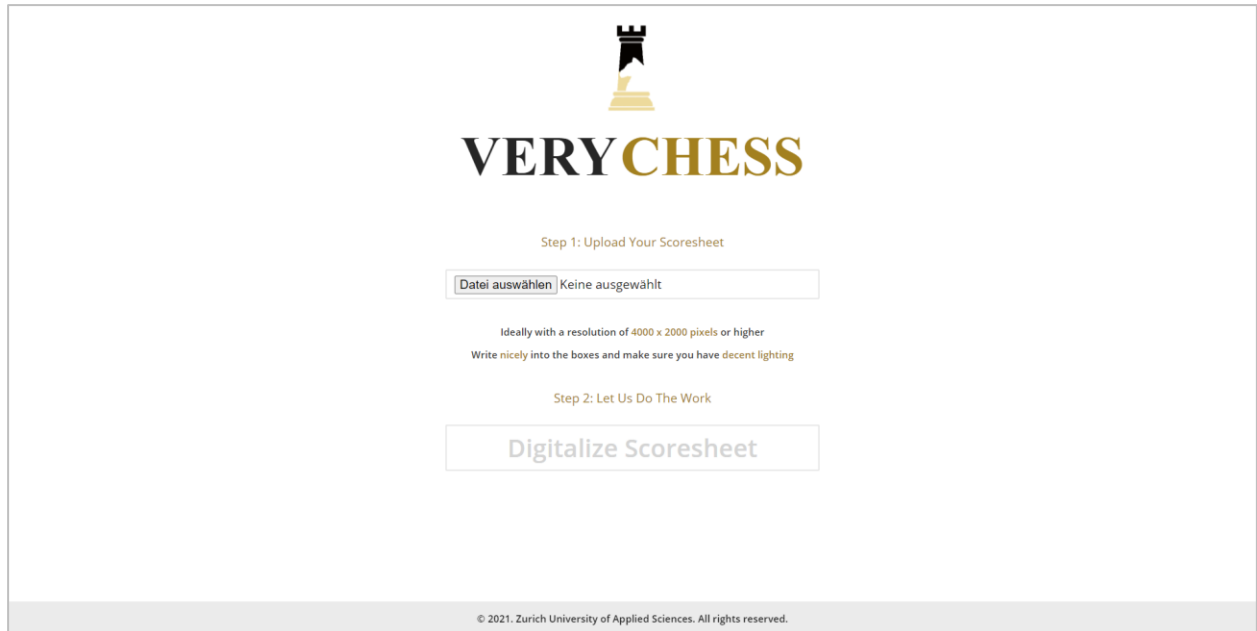


Abbildung 49: Very Chess – Upload-Page

Als erstes macht der Benutzer ein Bild von seinem Schach-Formular und lädt es auf Very Chess hoch.

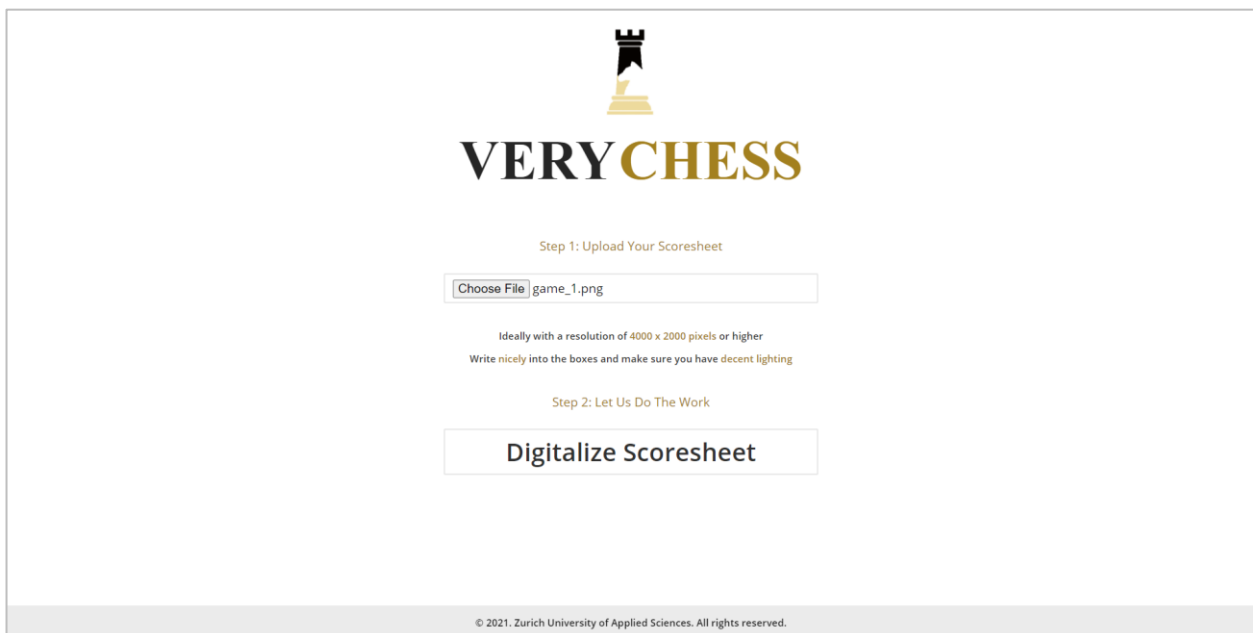


Abbildung 50: Upload-Page mit Bild - bereit für nächsten Schritt

Für den nächsten Schritt werden dem Benutzer die erkannten Boxen gezeigt. Dabei wird immer versucht, alle Boxen anzuzeigen. Dazu gehören auch die Boxen, bei denen keine Zeichen erkannt wurden. Ist also doch ein Zug vorhanden, der nicht erkannt wurde, kann dieser trotzdem gewählt werden. Die Boxen ohne Erkennungen sind jedoch von schlechterer Qualität.

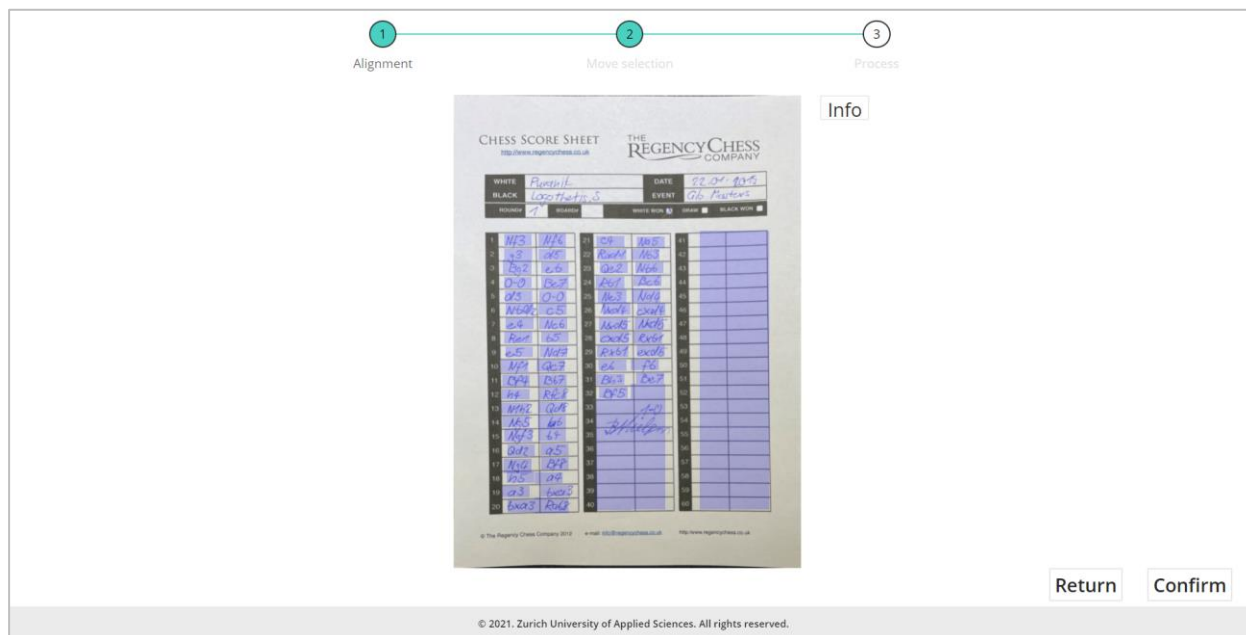


Abbildung 51: Anzeige der erkannten Boxen

Nun wählt der Benutzer also den letzten Zug des Spiels manuell aus, alle späteren Boxen werden für das weitere Vorgehen nicht mehr beachtet.

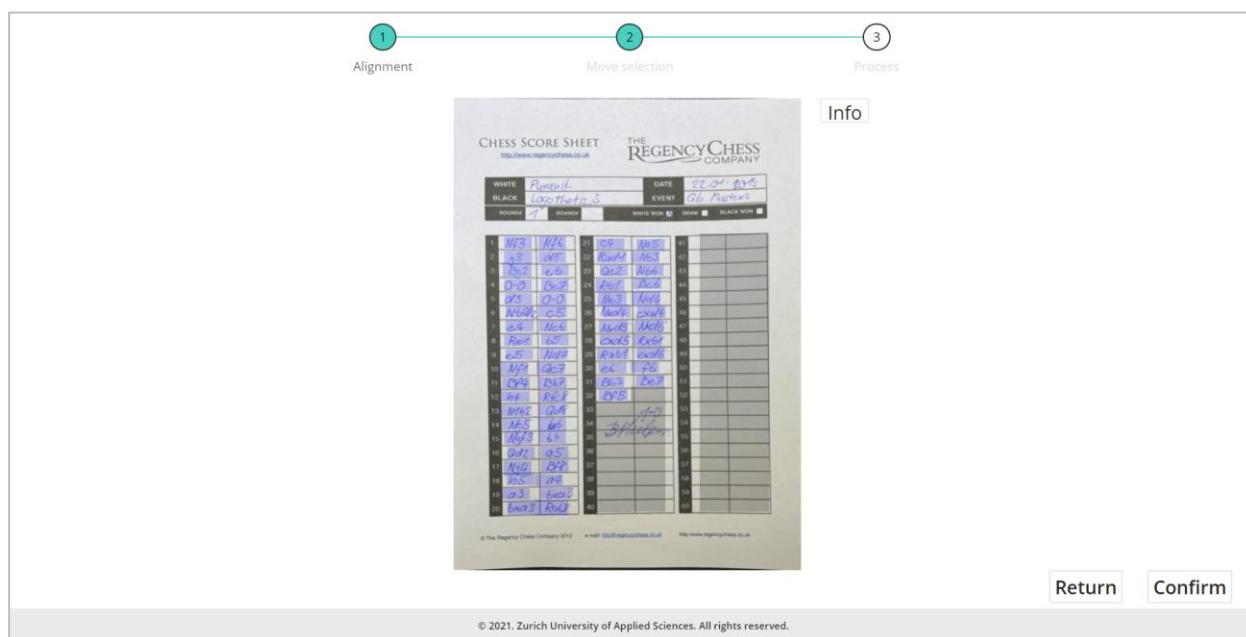


Abbildung 52: Ausgewählte Boxen des Spiels

Als letztes gelangt der Benutzer zur neuen Editor-Page, bei welcher das Schachspiel aufgebaut wird. Für das «game_1» des Datensets sieht die initiale Editor-Page wie folgt aus.

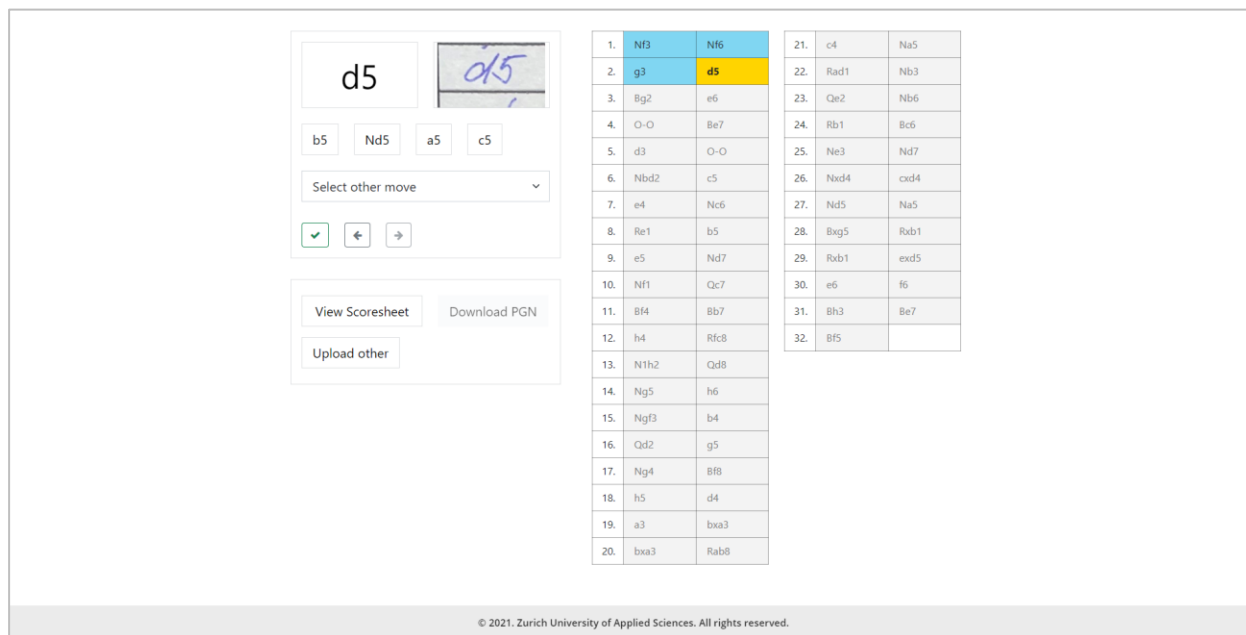


Abbildung 53: Initiale Editor-Page mit Eingabe «game_1» des Datensets

Wie zu sehen ist, wurden die ersten 3 Züge des Spiels schon übersprungen, alle übersprungenen Züge sind blau markiert. Die Zug-Korrektur ist sich also sehr sicher, dass diese Züge stimmen. Beim Zug «d5» wird der Benutzer nun gebeten, diesen entweder zu validieren oder zu korrigieren. Man sieht links den Zug, der gerade selektiert ist. Dort sieht man auch die Prediction «d5» und das Preview-Bild des Zuges rechts davon. Da «d5» in diesem Kontext korrekt ist, kann die grosse Schaltfläche für «d5» angeklickt werden. So wird der Zug validiert.

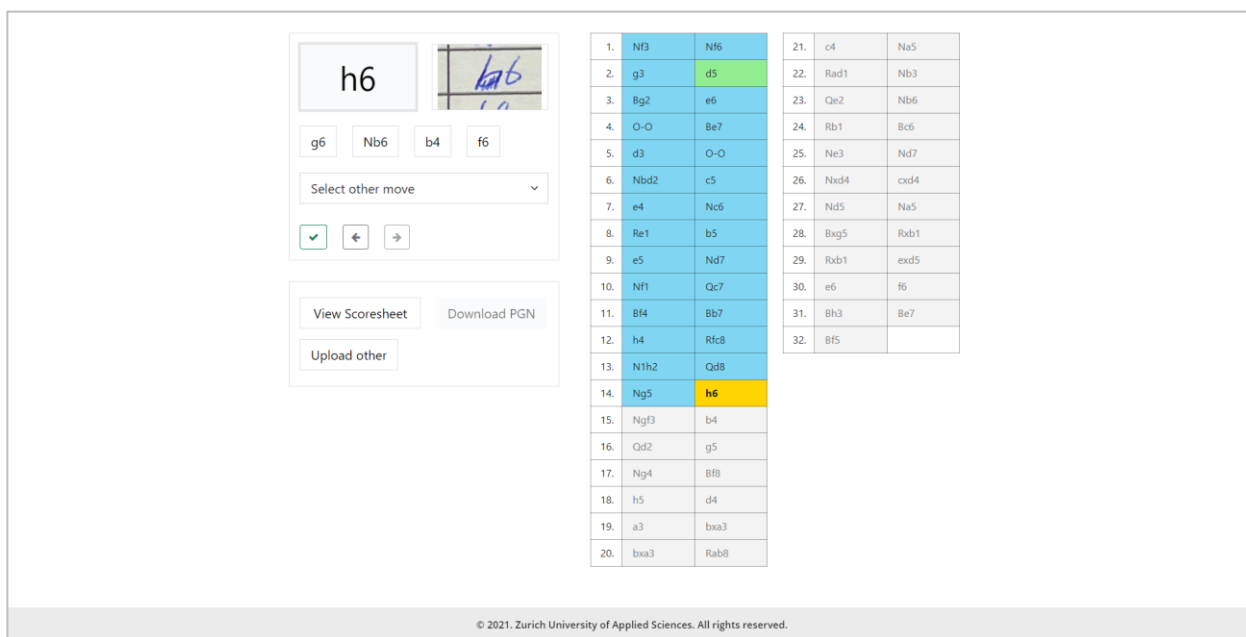


Abbildung 54: Editor-Page nach Validierung von «d5»

Nach der Korrektur vom Zug «d5» wurden wieder mehrere Züge übersprungen. Wie vorher wird für Zug «h6» der Benutzer angefragt. Es ist auch zu erkennen, warum sich die Zug-Erkennung eventuell nicht sicher ist, da dieses «h» vorher fälschlicherweise ein «b» war und durchgestrichen wurde. Folgend wird «h6» und auch «b4» vom Benutzer gutgeheissen.

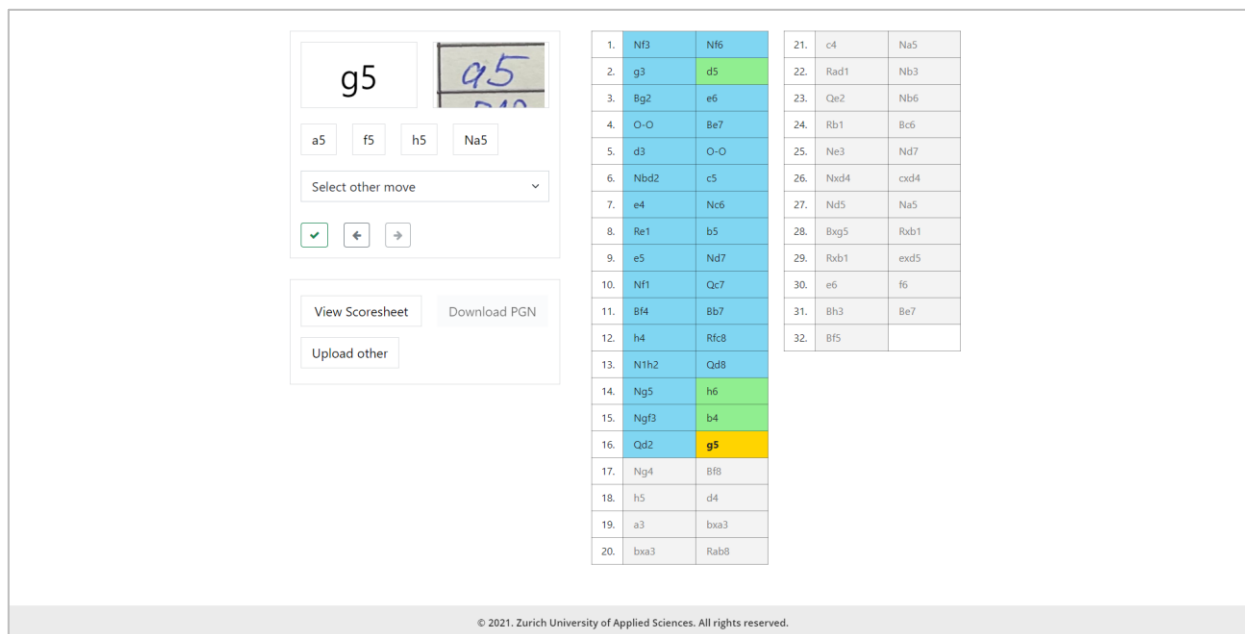


Abbildung 55: Editor-Page nach weiterer Validierung von «h6» und «b4»

Es erscheint der erste Fall, wo die Prediction der Zug-Korrektur falsch ist. Bei diesem Fall ist die typische Verwechslung von „a“ und „g“ aufgetreten (siehe 5.6 Confusion-Modul). Um die Korrektur benutzerfreundlicher und schneller zu machen, werden dem Benutzer die 4 Züge gezeigt, welche die Zug-Korrektur als nächstes gewählt hätte. In den meisten Fällen ist der Zug, der eine Korrektur benötigt, in diesen 4 Vorschlägen vorhanden. Für unsere Situationen ist dies auch der Fall. Der Benutzer kann also durch das Klicken der „a5“-Schaltfläche den Zug korrigieren.

Diese Validierungen bzw. Korrekturen werden nun solange gemacht, bis das Spiel vollständig aufgebaut wurde. In den Fällen, wo der Benutzer etwas übersehen hat, oder ein False-Positive aufgetreten ist. Kann jederzeit auf die entsprechenden Züge navigiert werden und diese geändert werden. Man kann sich auch durch die Schaltfläche «View Scoresheet» das gesamte Schach-Formular anzeigen lassen, falls dies für die Fehlerkorrektur nötig ist.

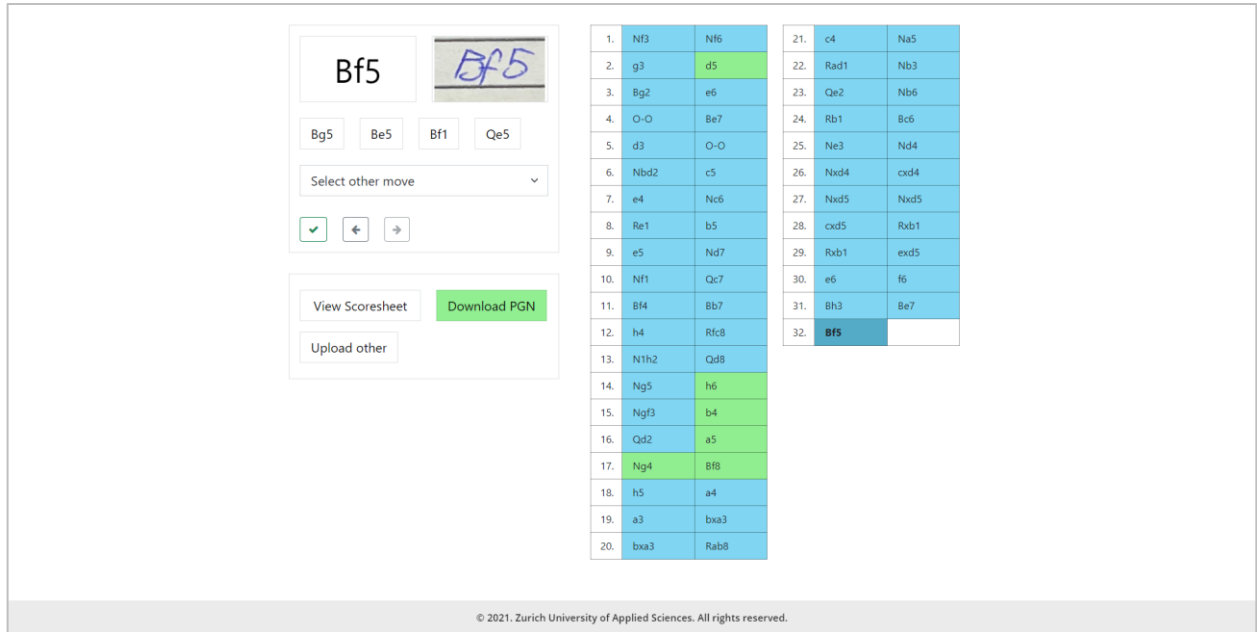


Abbildung 57: Editor-Page mit vollständig aufgebautem Schachspiel

Wurden alle Züge entweder übersprungen, validiert oder korrigiert. Kann das Schachspiel im PGN-Format heruntergeladen werden.

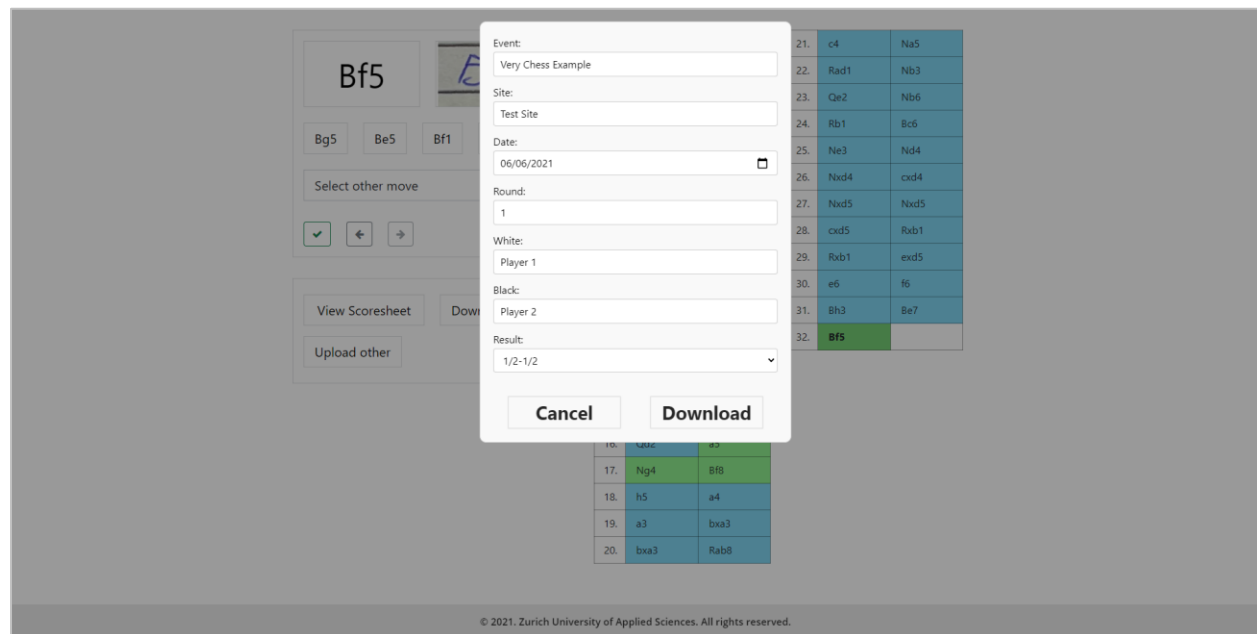


Abbildung 56: Dialog für das Herunterladen des Spiels mit Eingabe von Metadaten

Vor dem Download kann der Benutzer noch die Metadaten des Spiels wie oben angezeigt nachtragen. Standardmässige Daten, wie das Resultat des Spiels, werden auch vom PGN-Format vorausgesetzt. Wird für diese nichts Eingegeben, werden Default-Werte eingefügt. Die resultierende PGN-Datei sieht für «game_1» wie folgt aus.

```
[Event "Very Chess Example"]
[Site "Test Site"]
[Date "2021.06.06"]
[Round "1"]
[White "Player 1"]
[Black "Player 2"]
[Result "1/2-1/2"]

1.Nf3 Nf6 2.g3 d5 3.Bg2 e6 4.O-O Be7 5.d3 O-O 6.Nbd2 c5 7.e4
Nc6 8.Re1 b5 9.e5 Nd7 10.Nf1 Qc7 11.Bf4 Bb7 12.h4 Rfc8 13.N1h2
Qd8 14.Ng5 h6 15.Ngf3 b4 16.Qd2 a5 17.Ng4 Bf8 18.h5 a4 19.a3
bxa3 20.bxa3 Rab8 21.c4 Na5 22.Rad1 Nb3 23.Qe2 Nb6 24.Rb1 Bc6
25.Ne3 Nd4 26.Nxd4 cxd4 27.Nxd5 Nxd5 28.cxd5 Rxb1 29.Rxb1 exd5
30.e6 f6 31.Bh3 Be7 32.Bf5 1/2-1/2
```

Abbildung 58: Inhalt PGN-Datei für «game_1» des Datensets

6.2 Softwarearchitektur

Für das bestehende Projekt Very Chess wurden im Verlauf der Arbeit mehrere Verbesserungen implementiert. Der Code ist nun strikter modularisiert, damit einzelne Module für die anderen Tools importiert werden können. Zum Beispiel nutzt das Zug-Korrektur Analyse Tool die Funktionen von Very Chess (siehe 4.2.3 Zug-Korrektur Analyse Tool).

Die Applikation wird ausserdem für mehrere Benutzer tauglicher gemacht, indem die meisten benutzerabhängigen globalen Werte an die Session gebunden werden. Unter gewissen Umständen treten weiterhin Fehler auf, bei denen der Benutzer die Seite neu laden muss. Sie ist deshalb weiterhin nicht produktionstauglich (siehe 9 Ausblick).

7 Resultate

In diesem Kapitel werden die Resultate der Implementation aufgezeigt. Dabei werden die Ergebnisse der aktuellen Applikation, wie auch die der Zwischenstände erfasst. Zu beachten ist, dass die Resultate aufeinander aufbauen.

7.1 ICR-Anbieter

Evaluiert man, wie oft die ICR-Anbieter direkt den richtigen Zug ausgeben, ohne Zug-Korrektur und mit der neusten Box-Erkennung, kommen für das Datenset folgende Ergebnisse dabei raus.

ICR-Anbieter	Anteil direkt richtig erkannt
ABBYY Cloud	54.34%
Google Vision API	45.95%
Azure Cognitive Services	33.43%
Amazon Rekognition	33.43%

Tabelle 10: Genauigkeit direkter Vergleich ICR-Anbieter

Zu beachten ist, dass hier nur ein String-Vergleich gemacht wird, sobald ein Zeichen nicht stimmt, oder die Längen sich unterscheiden, gilt es als Fehler. Es wird keine Rücksicht genommen, ob eine Prediction sehr ähnlich ist wie der erwartete Zug. Dies wird erst mit der Zug-Korrektur ausgenutzt. Diese Evaluation ist als erster Schritt für den Vergleich mit den kommenden Resultaten zu sehen. Sie bildet auch kein wirkliches Schachspiel ab, sondern vergleicht jeden Zug einzeln. Hier schneidet ABBYY Cloud auch speziell gut ab, weil man dort schon ein Character-Set und einen Regex zur Filterung mitgibt (siehe 5.4.2 ABBYY Cloud).

7.2 Box-Erkennung

Im Folgenden werden die verschiedenen Iterationen der Box-Erkennung verglichen (siehe 5.5.5 Erstellung der Boxen). Für den Vergleich werden die ersten 20 Schach-Formulare des Datensets verwendet. Der Grund dafür ist die vorherige Box-Erkennung. Diese wird nur so weit optimiert, um die ersten 20 Schach-Formulare einzulesen. Ein Vergleich mit der unveränderten Initialversion ist nicht möglich, da diese die Mehrheit der Schach-Formulare des Datensets nicht vollständig erkennt (siehe Abbildung 31: Vorherige Box-Erkennung angewendet auf das Schach-Formular «game_1»). Als Vergleichswert wird der Output des Zug-Korrektur eingesetzt. Dabei werden für alle Iterationen die gleichen Zug-Korrektur Parameter als Baseline verwendet. Diese entsprechen nicht der aktuellen Version der Zug-Korrektur.

Iteration	Name	Anteil richtig
0	Vorherige Box-Erkennung	97.68%
1	Einfache Tabelle	94.95%
2	Optimierte Tabelle	95.97%
3	Optimierte Tabelle mit ICR-Boxen	96.25%
4	Optimierte Tabellen und Boxen	97.25%

Tabelle 11: Box-Erkennung – Anteil richtig für die ersten 20 Schach-Formulare

Folgend wird die vorherige Box-Erkennung sowie die Box-Erkennung mit optimierten Tabellen und Boxen für jeden ICR-Anbieter im Detail angezeigt.

ICR-Anbieter	Anteil richtig
ABBYY Cloud	90.18%
Google Vision API	88.51%
Azure Cognitive Services	85.22%
Ensemble	97.68%

Tabelle 12: Box-Erkennung Iteration 0 – Anteil richtig pro ICR für die ersten 20 Schach-Formulare

ICR-Anbieter	Anteil richtig
ABBYY Cloud	72.35%
Google Vision API	91.67%
Azure Cognitive Services	80.98%
Ensemble	97.25%

Tabelle 13: Box-Erkennung Iteration 4 – Anteil richtig pro ICR für die ersten 20 Schach-Formulare

7.3 Zug-Korrektur

Für die letzte Iteration der Zug-Korrektur wurden alle Parameter mittels Trial-and-Error manuell gesetzt. Folgend werden die resultierenden Parameter dazu mit ihren finalen Werten präsentiert.

Parameter	Wert
Minimum-Confusion-Value	0.35
Minimum-Confidence-Value	0.15
Length-Penalty	0.15
Direct-Hit-Bonus-Factor	1.2
Skip-Threshold	0.82

Tabelle 14: Aktuelle Parameter für Zug-Korrektur

Wird nun die Zug-Korrektur angewendet, ohne dass der Benutzer den laufenden Spielverlauf auf Anfrage korrigiert, treten, je länger das gesamte Spiel ist, immer mehr Folgefehler auf. Die resultierende Genauigkeit wird dadurch stark beeinträchtigt, und kann sogar schlechter sein als die direkte Genauigkeit (siehe 7.1 ICR-Anbieter). Ein zentraler Unterschied ist aber, dass jetzt als Ausgabe ein valides Schachspiel zurückgegeben wird, nicht einfach eine Folge von Erkennungen, die im Kontext der SAN-Notation ohnehin nicht für sich selbst stehen können. Weiter kann jetzt das Ensemble mit dieser Einschränkung verwendet werden, um erste Eindrücke zur Leistung des Ensembles zu erhalten.

ICR-Anbieter	Anteil richtig ohne Korrektur
ABBYY Cloud	31.35%
Google Vision API	55.25%
Azure Cognitive Services	30.40%
Amazon Rekognition	31.73%
Ensemble	82.10%

Tabelle 15: Zug-Korrektur - Anteil richtig ohne Benutzer-Korrektur

Schlussendlich folgen die Resultate, wo, wie auch vorgesehen, der Benutzer die Korrekturen fortlaufend macht, wenn die Zug-Korrektur mittels des Skipping-Algorithmus eine Anfrage an den Benutzer macht.

ICR-Anbieter	Anteil richtig mit Korrektur
ABBYY Cloud	72.72%
Google Vision API	90.92%
Azure Cognitive Services	81.52%
Amazon Rekognition	64.40%
Ensemble	97.26%

Tabelle 16: Zug-Korrektur - Anteil richtig mit Benutzer-Korrektur

Interessant ist jetzt noch, wie die Verteilung der 4 Fälle für den Skipping-Algorithmus aussieht. Da diese Verteilung stark von der Wahl des Skip-Threshold abhängt, folgt eine Visualisierung, die zeigt, wie das Verhältnis des Anteils der Skipped-Fälle zum Auftreten der False-Positives steht.

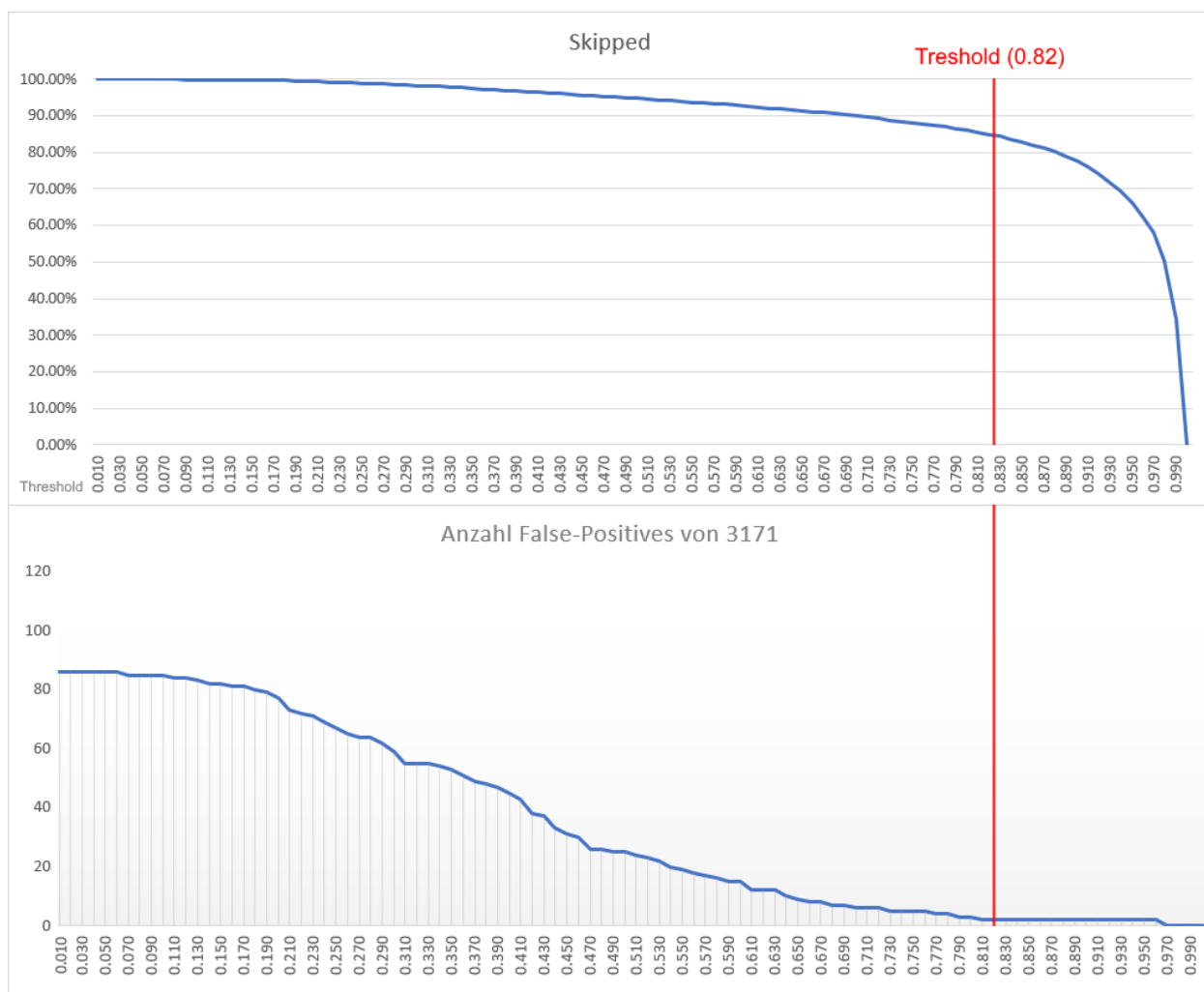


Abbildung 59: Verteilung von Skipped und False-Positives mit Threshold 0.82

Der gewählte Skip-Threshold mit dem Wert 0.82 führt dann zu folgenden Anteilen pro Fall.

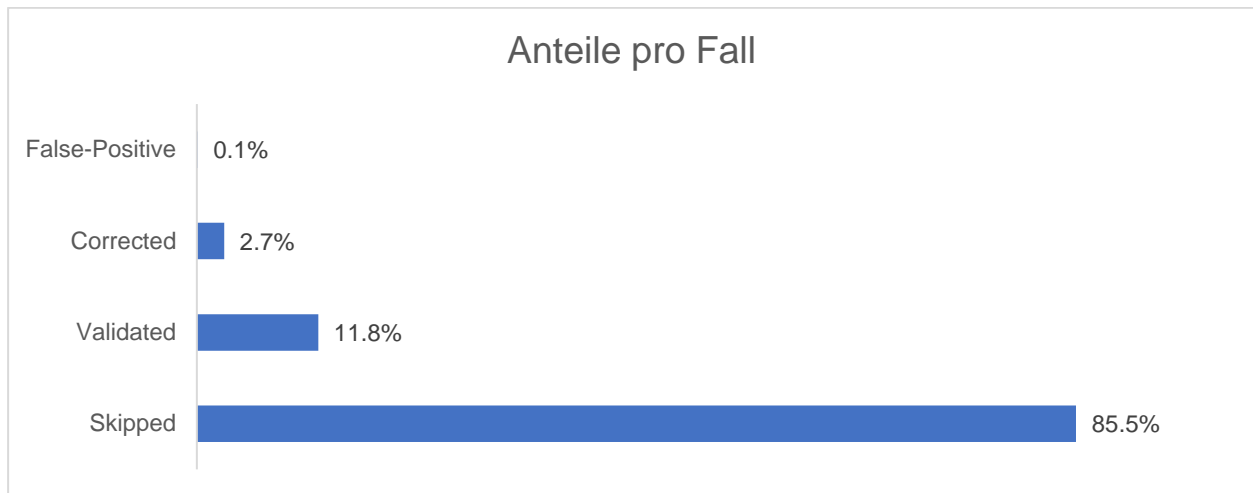


Abbildung 60: Anteile pro Fall für Skipping-Algorithmus mit Threshold 0.82

Schlussendlich folgen noch die absoluten Werte für jeden Fall mit dem gewählten Threshold 0.82. Für False-Positives sind lediglich 2 Fälle vorhanden, diese 2 False-Positives werden bei der Diskussion präsentiert (siehe 8 Diskussion).

Fall-Kategorie	Anzahl aus 3171 Zügen
False-Positive	2
Corrected	84
Validated	374
Skipped	2711

Abbildung 61: Absolute Anzahl Fälle für Skipping-Algorithmus mit Threshold 0.82

7.4 Zeitersparnis

Um einen Einblick darin zu bekommen, wie viel Zeit durch das Benutzen von Very Chess im Vergleich zur herkömmlichen Variante, bei welcher man die Züge manuell abliest und in ein virtuelles Schachbrett eingibt, gespart wird, wurden beide Vorgehensweisen für 3 Schach-Formulare des Datensets durchgeführt, und dabei die Zeit gestoppt.

Als erstes Schach-Formular wurde «game_1» vom Datenset ausgewählt, es ist relativ schön geschrieben und beinhaltet 63 Züge.

Testbenutzer	Herkömmlich (mm:ss)	Very Chess (mm:ss)
Bernt Nielsen	7:20	1:12
Volkan Caglayan	9:50	1:27
Person A	10:09	2:05
Person B	11:22	2:14

Tabelle 17: Zeitmessungen für «game_1»

Als zweites kommt das längere Spiel vom Schach-Formular «game_40» zum Einsatz. Es enthält 110 Züge und ist schwieriger zu lesen.

Testbenutzer	Herkömmlich (mm:ss)	Very Chess (mm:ss)
Bernt Nielsen	11:12	2:42
Volkan Caglayan	14:32	2:53
Person A	15:49	3:54
Person B	14:40	3:17

Tabelle 18: Zeitmessungen für «game_40»

Nun zum Schluss wird noch einer der zwei Schach-Formulare gewählt, in dem ein False-Positive auftritt. Nämlich «game_28» mit insgesamt 50 Zügen und mittelmässig schöner Handschrift.

Testbenutzer	Herkömmlich (mm:ss)	Very Chess (mm:ss)
Bernt Nielsen	7:58	4:33
Volkan Caglayan	10:52	5:19
Person A	10:25	6:36
Person B	9:44	8:53

Tabelle 19: Zeitmessungen für «game_28»

8 Diskussion

In diesem Kapitel werden die Resultate der Arbeit diskutiert, dabei wird erläutert, wie die Resultate zustande kommen und was sie auszeichnet. Zusätzlich wird besprochen, welche Änderungen in Vergleich zur vorherigen Version am meisten ausmachen.

Box-Erkennung

Wie in den Resultaten zu sehen ist, erzielt die vorherige Box-Erkennung den höchsten Anteil richtig erkannter Schachzüge (siehe Tabelle 11: Box-Erkennung – Anteil richtig für die ersten 20 Schach-Formulare). Sie schneidet sogar besser ab als die aktuelle Box-Erkennung mit optimierten Tabellen und Boxen. Dies ist jedoch zu erwarten, da die Boxen der vorherigen Box-Erkennung besonders optimiert für ABBYY Cloud sind. Vergleicht man die beiden Box-Erkennungen, in denen nur ABBYY Cloud eingesetzt wird, ist zu erkennen, dass die vorherige Box-Erkennung die 4. Iteration der Box-Erkennung um 17.83% übertrifft (siehe Tabelle 12: Box-Erkennung Iteration 0 – Anteil richtig pro ICR für die ersten 20 Schach-Formulare und Tabelle 13: Box-Erkennung Iteration 4 – Anteil richtig pro ICR für die ersten 20 Schach-Formulare). Die Ergebnisse der restlichen ICR unterscheiden sich zwischen 3% bis 5%. Dieser Unterschied entsteht ist aufgrund der unterschiedlichen Boxen und ist zu erwarten.

Neben den Vergleichswerten der Resultate ist für die Box-Erkennung besonders die Stabilität der Lösung wichtig. Das Ziel ist es, möglichst für alle Schach-Formulare Boxen erkennen zu können, die der Zug-Korrektur zum Spiel verhelfen. Die vorherige Box-Erkennung konnte nur mit spezifischen Anpassungen die ersten 20 Schach-Formulare des Datensets korrekt erkennen. Wobei die aktuelle Box-Erkennung für alle 50 Schach-Formulare im Datenset funktioniert und somit die robustere Lösung darstellt.

Zug-Korrektur

Vergleicht man direkt die Ausgabe der ICRs mit den richtigen Zügen, wirkt das Resultat vorerst unbefriedigend. Nur etwa ein Drittel bis zur Hälfte der Züge wird je nach ICR genau richtig erkannt. Untersucht man jedoch die Ausgaben der ICRs, merkt man, dass häufig nur kleine Ungenauigkeiten und Verwechslungen zu den Fehlern beitragen.

Schaut man sich die Resultate für die Zug-Korrektur ohne Validierung/Korrektur durch den Benutzer an, scheint es sich sogar verschlechtert zu haben. Wie aber schon erwähnt kommt als Ausgabe eine gültige Folge von Zügen bzw. ein Schachspiel raus. In den Fällen, in denen sich die Genauigkeit verschlechtert hat, liegt es daran, dass sich am Anfang vom Spiel Fehler eingeschlichen haben. Diese führen dann dazu, dass die Folgezüge keine legalen Zug-Kandidaten enthalten, die dem erkannten Zug entsprechen, es kommt also zwingend zu

Folgefehlern. Deshalb kam auch die Entscheidung, das System mit der Benutzerinteraktion einzuführen.

Wird also das Spiel vorlaufend vom Benutzer validiert und bei Bedarf korrigiert, verbessern sich die Resultate beträchtlich. Nun erkennt man auch, wie wenig Aussagekraft der direkte Vergleich der ICRs mit den echten Zügen hat, obwohl ABBYY Cloud direkt 54.34% der Züge richtig erkennt und Google Vision API nur 45.95%, schneidet Google Vision API mit der Zug-Korrektur besser ab als ABBYY Cloud. Mit 90.92% für Google Vision API mit Zug-Korrektur, und nur 72.72% für ABBYY Cloud (siehe Tabelle 16: Zug-Korrektur - Anteil richtig mit Benutzer-Korrektur). Dies liegt zum einen dran, dass bei ABBYY Cloud schon viele Fehler wegen dem Character-Set und Regex-Filter verhindert werden, die bei Google Vision API erst bei der Zug-Korrektur bereinigt werden. Zum anderen bieten die Erkennungen von Google Vision API generell mehr Potenzial zur richtigen Interpretation bzw. mehr Information.

Wichtig sind aber nicht die ICRs individuell, sondern das Zusammenspiel der ICRs im Ensemble, mit einem Endresultat von 97.26% bietet das Ensemble einen klaren Mehrwert. Dies bedeutet für den Benutzer, wenn er ein ähnlich gut geschriebenes Schach-Formular verwendet, wie diese im Datenset, muss er erwartungsgemäss nur ca. 3% der Züge aktiv korrigieren.

Was hinzukommt, sind die Resultate für den Skipping-Algorithmus. Der Benutzer muss nur bei ca. 15% der Züge eingreifen. Dabei sind ca. 12% Fälle, wo er den Zug gutheisst. Am problematischsten sind aber die Fälle von False-Positives. Über die False-Positives, die mit dem Skip-Threshold 0.82 auftreten, kann statistisch nicht viel ausgesagt werden, da nur 2 Fälle für das Datenset vorhanden sind, welche auch für einen Menschen schwierig sind.

ICR-Ausgaben	
b6, 66, 56, b6, 66	a, Ra1, Ra1, Ra1, Ra1
Prediction	
b6 (0.97 Confidence)	Ra1 (0.97 Confidence)
Actual	
h6	Rd1




Abbildung 62: Die zwei False-Positives bei Threshold 0.82

Untersucht man die 2 Fälle, stellt man fest, dass beide bei den typischen Fehlerfällen einzuordnen sind. Es ist auch unglücklich, dass genau beide der sehr ähnlich aussehenden Züge in der Schachposition legal sind. Um mehr False-Positives für die Untersuchung zu erhalten, kann der Skip-Threshold gesenkt werden. Wird dies gemacht mit Threshold 0.68, kommen dabei 6 neue Fälle von False-Positives vor, und es werden 90.92% der Züge übersprungen.

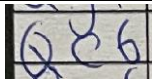
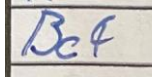
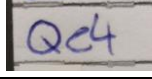


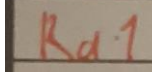
ICR-Ausgaben	Prediction	Confidence	Actual	Preview
c8, 66, Q:6, 66	b6	0.81	Qe6	
Bc4, Bet, Bef, Bef, Bet	Be4	0.79	Bc4	
Qc4, Q64, Qe4, Qc4, Q64	Qc4	0.77	Qe4	
5a, INCTI, INCF, NC7, INCTI	Nd4	0.73	Ne7	
a, Ra1, RE, RFC, Ra1	Ra1	0.70	Rfc1	
Bd1, Ra1, Rd1, Ra1	Rd1	0.68	Ra1	

Tabelle 20: Die 6 zusätzlichen False-Positives bei Threshold 0.68

Bei der Untersuchung der neuen 6 False-Positives gibt es wieder ein paar erwartungsgemässe Fälle, hier besonders die Verwechslung von «c» und «e» und «d» mit «a». Beim ersten Fall ist noch das Problem, dass der obere Zug (das geschriebene «g») noch in die Zelle ragt, und somit die Erkennung des «c» stört.

Zeitersparnis

Für die Resultate der Zeitersparnis ist zu bemerken, dass es eine grosse Rolle spielt, ob man sich mit der Umgebung schon vertraut ist, und ob einem die Schachregeln bzw. Schachnotation geläufig sind. Diese Experimente sind lediglich als Proof-of-Concept für das Potential der Zeitersparnis zu sehen.

Was bei den Resultaten für die Zeitersparnis gezeigt wird ist, dass Very Chess einen klaren Mehrwert bezüglich des Zeitaufwandes bieten kann (siehe 7.4 Zeitersparnis). Je nachdem, wie schön das Schach-Formular geschrieben wurde, ist die Zeitersparnis noch höher, speziell bei Fällen, wo direkt alle Züge übersprungen werden und korrekt sind. Was die Zeitersparnis aber stark eindämmt, sind das Auftreten von False-Positives, wie beim dritten Experiment ersichtlich. Dort mussten die Benutzer selbst den False-Positive erkennen und rückwirkend korrigieren, was unter Umständen viel Zeit kostet.

9 Ausblick

Die Weiterentwicklung von Very Chess hat viel Potenzial, es gibt noch viele mögliche Verbesserungen, um die Applikation marktgebräuchlicher zu machen, und die Genauigkeit der Erkennung zu erhöhen. Weiter kann auch die Anzahl übersprungener Züge erhöht werden, während dem die Anzahl False-Positives minimal gehalten wird.

Mit dem Ensemble besteht bereits ein einfacher Weg für Verbesserungen. Man muss lediglich neue ICR-Anbieter an das Ensemble anschliessen, und die Parameter dementsprechend anpassen. Was dazu realisiert werden kann, um die Zug-Korrektur weiter zu verbessern, ist eine automatische Methode, welche die verschiedenen Parameter der Zug-Korrektur anpasst. Es kann beispielsweise ein evolutionärer Optimierungsalgorithmus darauf angesetzt werden. Bisher wurden alle Parameter manuell mittels Trial-and-Error ermittelt.

Für die bestehende Applikation kann auch der ICR-Anbieter Amazon Rekognition so angepasst werden, dass nur Teilbilder des Schach-Formulars gesendet werden, die das Erkennungslimit von 50 nicht übersteigen, und die Ausgaben für diese Teilbilder später wieder zusammenführen.

Ein weiterer Ansatz ist, die Boxen, welche ABBYY Cloud übergeben werden, vorher mit verschiedenen Bildbearbeitungsmethoden zu verbessern, indem man beispielsweise Tabellenlinien entfernt oder Zeichen, die nicht zum Zug gehören, rausfiltert. Das Potenzial dabei ist schon bekannt, den bei der vorherigen Version von Very Chess erzielt ABBYY Cloud mit der Tabellenerkennung, die ersetzt wurde, bessere Resultate. Anstatt die Boxen mit Nachbearbeitung zu verbessern, kann man auch einfach beide Box-Erkennungen verwenden, also die aktuelle Variante und die vorherige Tabellen-basierte Variante. Es kann, ähnlich wie für die Zug-Korrektur, ein Ensemble mit den grundlegenden Varianten zur Box-Erkennung implementiert werden.

Was die Box-Erkennung und die Zug-Korrektur zusätzlich verbessern und robuster machen kann, ist ein erweitertes Datenset mit mehr Schach-Formularen, die von mehr verschiedenen Personen auf unterschiedlichen Formularvorlagen geschrieben werden. Ist die Applikation in Gebrauch, kann man die hochgeladenen Schach-Formulare zusammen mit den PGN-Dateien, die vom Benutzer erstellt wurden, auf dem Server speichern und zum Datenset hinzufügen.

Was an der Applikation noch technisch gesehen verbessert werden kann, sind Aspekte bezüglich des Deployment und die Multiuser-Funktionalität/Skalierbarkeit. Zurzeit gibt es viele technische Vorgänge, welche die Skalierbarkeit stark einschränken. Die Bilder der Schach-Formulare werden zum Beispiel immer noch als Dateien auf dem Filesystem des Servers gespeichert, statt in einer Datenbank. Auch die Interaktion zwischen Client und Server entspricht zurzeit nicht

modernen Standards. Der Server sollte so gestaltet werden, dass er als einfache API dient, und der Client wird so umgeändert, dass er unabhängig von der API als SPA (Single Page Application) aufgebaut ist, mit einem modernen Web-Framework.

Als letztes könnte die Applikation so erweitert werden, dass sie direkt auch bei der Analyse des erkannten Spiels hilft, statt dass man die PGN-Datei herunterlädt und auf einem anderen Analyse-Tool einliest. Diese Erweiterung ist auch bei einigen Konkurrenzprodukten, beispielsweise Reine – Chess (siehe 2.1 Reine – Chess) vorhanden.

10 Verzeichnisse

10.1 Literaturverzeichnis

- [1] lichess.org. (2021). *Analysis board - lichess.org* [Online]. URL: <https://lichess.org/analysis> [Stand: 09.06.2021]
- [2] ChessBase. (2021). *ChessBase - Chess database with analysis* [Online]. URL: <https://database.chessbase.com/> [Stand: 09.06.2021]
- [3] Schweizer Schachbund. (2021). *Home-de - SSB* [Online]. URL: <http://www.swisschess.ch/home-de.html> [Stand: 09.06.2021]
- [4] B. Horváth und C. Dreher, „Bachelor ' s Thesis Computer Science Document Digitization for Chess Scorecards“, ZHAW School of Engineering, 2020.
- [5] Pallets. (2010). *Flask - web development, one drop at a time* [Online]. URL: <https://flask.palletsprojects.com/en/2.0.x/> [Stand: 09.06.2021]
- [6] R. Sudharsan und A. Fung. (2020). *Reine - Chess Scoresheet Scanner* [Online]. URL: <https://www.reinechess.com/> [Stand: 09.06.2021]
- [7] R. Muñoz-Salinas. (2018). *ArUco: a minimal library for Augmented Reality applications based on OpenCV* [Online]. URL: <https://www.uco.es/investiga/grupos/ava/node/26> [Stand: 09.06.2021]
- [8] A. Nagdev. (2021). *CheSScan* [Online]. URL: <https://play.google.com/store/apps/details?id=com.nagdev.alok.chessscan&hl=en&gl=US> [Stand 09.06.2021]
- [9] CalcFxC. (2020). *Chess Score Pad - Electronic Chess Score Sheet* [Online]. URL: <https://apps.apple.com/us/app/chess-score-pad/id518018730> [Stand: 09.06.2021]
- [10] DGT. (2021). *Electronic Boards > - Digital Game Technology* [Online]. URL: <http://www.digitalgametechnology.com/index.php/products/electronic-boards> [Stand: 09.06.2021]
- [11] Schach Euro Dresden. (2021). *DGT Schachbretter* [Online]. URL: <https://www.euroschach.de/schachspiele/schachcomputer/dgt-schachbretter/> [Stand: 09.06.2021]

- [12] Nicomsoft OCR SDK Tutorials. (2012). *Optical Character Recognition (OCR) – How it works* [Online]. URL: <https://www.nicomsoft.com/optical-character-recognition-ocr-how-it-works/> [Stand 10.06.2021]
- [13] SoftGuide. (2021). *OCR, ICR* [Online]. URL: <https://www.softguide.de/software-tipps/ocr-icr> [Stand 10.06.2021]
- [14] J. Chris. (2021). *Standard algebraic notation - chess* [Online]. URL: <http://cfajohnson.com/chess/SAN/> [Stand: 09.06.2021]
- [15] FIDE. (2021). *About FIDE* [Online]. URL: <https://www.fide.com/fide/about-fide> [Stand: 09.06.2021]
- [16] chess.com. (2021). *Chess PGN (Portable Game Notation)* [Online]. URL: <https://www.chess.com/terms/chess-pgn> [Stand: 09.06.2021]
- [17] Microsoft. (2021). *Video Conferencing, Meetings, Calling | Microsoft Teams* [Online]. URL: <https://www.microsoft.com/en-ww/microsoft-teams/group-chat-software> [Stand: 09.06.2021]
- [18] Discord. (2021). *Discord | Your Place to Talks and Hang out* [Online]. URL: <https://discord.com/> [Stand: 09.06.2021]
- [19] ABBYY. (2021). *ABBYY Cloud OCR SDK* [Online]. URL: <https://www.abbyy.com/cloud-ocr-sdk/> [Stand: 10.06.2021]
- [20] PGNMentor. (2018). *Download PGN Files* [Online]. URL: <http://www.pgnmentor.com/files.html> [Stand: 08.06.2021]
- [21] A. Rosebrock. (2014). *How to Build a Kick-Ass Mobile Document Scanner in Just 5 Minutes* [Online]. URL: <http://www.pyimagesearch.com/2014/09/01/build-kick-ass-mobile-document-scanner-just-5-minutes/> [Stand: 08.06.2021]
- [22] OpenCV. (2021). *Home - OpenCV* [Online]. URL: <https://opencv.org/> [Stand: 10.06.2021]
- [23] A. Abduli, „Project work Computer Science Document Digitization for Chess Scorecards“, ZHAW School of Engineering, 2020.
- [24] Google Open Source. (2021). *Tesseract OCR* [Online]. URL: <https://opensource.google/projects/tesseract> [Stand: 10.06.2021]
- [25] Google Cloud. (2021). *Vision AI | Mit ML Informationen aus Bildern gewinnen* [Online]. URL: <https://cloud.google.com/vision/> [Stand: 10.06.2021]

- [26] Microsoft. (2021). *Computer Vision documentation - Quickstarts, Tutorials, API Reference - Azure Cognitive Services | Microsoft Docs* [Online]. URL: <https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/> [Stand: 10.06.2021]
- [27] Amazon. (2021). *Amazon Rekognition – Image and Video Analysis – Amazon Web Services (AWS)* [Online]. URL: <https://aws.amazon.com/rekognition/> [Stand: 10.06.2021]
- [28] The SciPy community. (2021). *scipy.optimize.differential_evolution — SciPy v1.6.3 Reference Guide* [Online]. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html [Stand: 10.06.2021]
- [29] Ex Machina. (2019). *An AI with Intuition?* [Online]. URL: <https://zxul767.dev/alpha-go/> [Stand: 09.06.2021]

10.2 Abbildungsverzeichnis

Abbildung 1: Seite zum Hochladen des Schach-Formulars für vorherige Applikation [4]	2
Abbildung 2: Ausrichtung eines Schach-Formulars [4].....	3
Abbildung 3: Bestätigung der letzten Zelle, in welcher der Zug steht [4]	3
Abbildung 4: Entfernung der Kopfzeilen durch Benutzer [4].....	4
Abbildung 5: Benutzeroberfläche zur Bearbeitung des vorgeschlagenen Spiels [4].....	4
Abbildung 6: Formular für das Herunterladen des Spiels [4].....	5
Abbildung 7: Standard Vorlage für Schach-Formular von Reine – Chess [6]	8
Abbildung 8: Analyse Tool von Reine – Chess mit Schachcomputer [6]	9
Abbildung 9: CheSScan Benutzeroberfläche für Android [8].....	10
Abbildung 10: Chess Score Pad – Benutzeroberfläche mit Export als Schach-Formular [9]	11
Abbildung 11: DGT-Schachbrett verbunden mit Smart-Schachuhr [10]	12
Abbildung 12: SAN-Definition der Felder [14]	15
Abbildung 13: Beispiel wo beide Springer auf Zielfeld «d2» ziehen können.....	16
Abbildung 14: Inhalt einer PGN-Datei mit SAN-Notation unterhalb.....	17
Abbildung 15: Momentaufnahme Project-Board von GitHub für Very Chess	18
Abbildung 16: Anzeige der erkannten Zeichen im ICR Preview Tool für Google Vision API	19
Abbildung 17: Anzeige der Boxen im ICR Preview Tool für Google Vision API.....	19
Abbildung 18: Ausschnitt des CSV-Exports aus dem CSV-Generator	20
Abbildung 19: Ausgabe des Zug-Korrektur Analyse Tools.....	21
Abbildung 20: Detailausgabe des Zug-Korrektur Analyse Tools	21
Abbildung 21: Schach-Formular aus dem Erkennungs-Datenset [4].....	22
Abbildung 22: Schach-Formular aus dem Layout-Datenset [4]	22
Abbildung 23: Standard Schach-Formular [4]	23
Abbildung 24: Beispiel eines alternativen Schach-Formulars [4].....	23
Abbildung 25: Schach-Formular «game_26».....	24
Abbildung 26: Schach-Formular «game_33».....	24
Abbildung 27: Angepasster Workflow für Very Chess.....	25
Abbildung 28: Box-Erkennung ohne Ausrichtung.....	26
Abbildung 29: Artefakte bei der Linienentfernung	27
Abbildung 30: Entfernung von Zeichenkonturen	27
Abbildung 31: Vorherige Box-Erkennung angewendet auf das Schach-Formular «game_1» ...	31
Abbildung 32: Bildung der Zahlenspalten auf das Schach-Formular «game_43»	33
Abbildung 33: Entfernung der Duplikate und Filterung der Spalten auf das Schach-Formular «game_43»	34

Abbildung 34: Bildung der Nummerierungsspalten auf das Schach-Formular «game_43».....	35
Abbildung 35: Box-Erkennung mit einfachen Tabellen angewendet auf das Schach-Formular «game_1»	37
Abbildung 36: Box-Erkennung mit optimierten Tabellen auf Schach-Formular «game_1».....	39
Abbildung 37: Box-Erkennung mit optimierten Tabellen und ICR-Boxen auf Schach-Formular «game_1»	40
Abbildung 38: Box-Erkennung mit optimierten Tabellen und Boxen auf Schach-Formular «game_1»	41
Abbildung 39: Beispiel für Verwechslungsgefahr mit "a" und "g"	43
Abbildung 40: Beispiel für Verwechslungsgefahr mit „b“ und „h“	43
Abbildung 41: Confusion-Matrix mit Google Vision API als ICR.....	44
Abbildung 42: Struktur für Verwechslungen.....	45
Abbildung 43: Confidence für Kandidaten eines Zuges mit Google Vision API als ICR	46
Abbildung 44: Ensembling-Modell für Erkennung des Schach-Formulars.....	48
Abbildung 45: Illustration zur Einbeziehung aller Kandidaten pro ICR	49
Abbildung 46: Visualisierung des Schach-Spielbaums [29]	50
Abbildung 47: Beispiel für False-Positive-Fall (Confidence Hoch, Prediction aber falsch).....	52
Abbildung 48: Schach-Formular, das vollautomatisch digitalisiert wird	53
Abbildung 49: Very Chess – Upload-Page.....	54
Abbildung 50: Upload-Page mit Bild - bereit für nächsten Schritt.....	54
Abbildung 51: Anzeige der erkannten Boxen.....	55
Abbildung 52: Ausgewählte Boxen des Spiels.....	55
Abbildung 53: Initiale Editor-Page mit Eingabe «game_1» des Datensets.....	56
Abbildung 54: Editor-Page nach Validierung von «d5»	56
Abbildung 55: Editor-Page nach weiterer Validierung von «h6» und «b4».....	57
Abbildung 56: Dialog für das Herunterladen des Spiels mit Eingabe von Metadaten	58
Abbildung 57: Editor-Page mit vollständig aufgebautem Schachspiel.....	58
Abbildung 58: Inhalt PGN-Datei für «game_1» des Datensets	59
Abbildung 59: Verteilung von Skipped und False-Positives mit Threshold 0.82	63
Abbildung 60: Anteile pro Fall für Skipping-Algorithmus mit Threshold 0.82	64
Abbildung 61: Absolute Anzahl Fälle für Skipping-Algorithmus mit Threshold 0.82.....	64
Abbildung 62: Die zwei False-Positives bei Threshold 0.82.....	68

10.3 Tabellenverzeichnis

Tabelle 1: Zeichen für Figuren in SAN-Notation.....	15
Tabelle 2: Eigenschaften von ABBYY Cloud [19].....	28
Tabelle 3: Eigenschaften von Google Vision API [25].....	29
Tabelle 4: Eigenschaften von Azure Cognitive Services [26].....	29
Tabelle 5: Eigenschaften von Amazon Rekognition [27].....	30
Tabelle 6: Wertebereiche der Parameter für die Tabellenoptimierung.....	38
Tabelle 7: Wertebereiche der Parameter für die Box-Optimierung.....	41
Tabelle 8: Verwechslungshäufigkeiten für Google Vision API.....	46
Tabelle 9: Beispiel für Kombinationen der ICR-Ausgabe «Bb» für Länge 4.....	47
Tabelle 10: Genauigkeit direkter Vergleich ICR-Anbieter.....	60
Tabelle 11: Box-Erkennung – Anteil richtig für die ersten 20 Schach-Formulare.....	61
Tabelle 12: Box-Erkennung Iteration 0 – Anteil richtig pro ICR für die ersten 20 Schach-Formulare.....	61
Tabelle 13: Box-Erkennung Iteration 4 – Anteil richtig pro ICR für die ersten 20 Schach-Formulare.....	61
Tabelle 14: Aktuelle Parameter für Zug-Korrektur.....	62
Tabelle 15: Zug-Korrektur - Anteil richtig ohne Benutzer-Korrektur.....	62
Tabelle 16: Zug-Korrektur - Anteil richtig mit Benutzer-Korrektur.....	63
Tabelle 17: Zeitmessungen für «game_1».....	65
Tabelle 18: Zeitmessungen für «game_40».....	65
Tabelle 19: Zeitmessungen für «game_28».....	65
Tabelle 20: Die 6 zusätzlichen False-Positives bei Threshold 0.68.....	68

11 Anhang

11.1 Projektmanagement

11.1.1 Offizielle Aufgabenstellung

Allgemeines:	
Titel:	OCR für Schach Scorecards [Web-App, Machine Learning]
Anzahl Studierende:	2
Durchführung in Englisch möglich:	Ja, die Arbeit kann vollständig in Englisch durchgeführt werden und ist auch für Incomings geeignet.
Betreuer:	Zugeteilte Studenten:
HauptbetreuerIn: Mark Cieliebak,  ciel	Diese Arbeit ist zugeteilt an: - Volkan Caglayan, caglavl (IT) - Bernt Nielsen, nielsber (IT)
Fachgebiet:	Studiengänge:
SOW Software	IT Informatik
Zuordnung der Arbeit :	Infrastruktur:
InIT Institut für angewandte Informationstechnologie	benötigt keinen zugeteilten Arbeitsplatz an der ZHAW
Interne Partner :	Industriepartner:
Es wurde kein interner Partner definiert!	Es wurden keine Industriepartner definiert!
Beschreibung:	
<p>Many serious chess players take notes of the moves during each game, which allows them to replay and analyze these games afterwards. These notes are taken by hand on a scorecard, and afterwards entered manually into a chess program.</p> <p>We are developing a mobile app that does this automatically: take a picture of a scorecard, detect the moves, and provide them in machine readable form.</p> <p>In two previous student projects, we already implemented a rudimentary web app that can detect the moves from a scorecard. Put shortly, it takes an image of a score card, runs line detection and OCR to recognize the handwritten characters in the sheet, and applies a chess rule engine to predict the most likely sequence of moves.</p> <p>Goal of the Thesis:</p> <p>The existing app should be extended and optimized such that a satisfying solution for the user evolves. This includes the following steps:</p> <p>Analyze which algorithmic steps in the image processing are responsible for errors in the detected moves Optimize the corresponding algorithms Extend and improve the existing web app, since the user interface is very basic and sometimes unintuitive.</p> <p><i>If you are interested in this challenging topic, please get in touch. We will then meet and discuss the details. Email: ciel@zhaw.ch, Phone: 058 934 72 39.</i></p>	
Voraussetzungen:	
<p>Current Technology Stack:</p> <ul style="list-style-type: none"> • Backend: Python, Flask, OpenCV • Deployment: Apache • Frontend: Javascript, no particular GUI framework) 	

11.1.2 Zeitplan

Für dieses Projekt wurde kein statischer Zeitplan eingesetzt, stattdessen wurde die Planung agil durchgeführt mit dem GitHub Project-Board und Issues. Dabei wurde bei jeder wöchentlichen Sitzung die gemachte Arbeit evaluiert, und es wurden neue Ziele für die nächste Woche diskutiert und festgelegt. Diese Arbeitsform erlaubte es, dynamische Entscheidung bezüglich des Projektverlaufes zu treffen.

11.2 Weiteres

11.2.1 Anleitungen

ICR-Anbieter aufsetzen mit Credential File

Zur Nutzung von Very Chess werden für die jeweiligen ICR-Anbieter Credentials benötigt. Diese erhält man, wenn man einen Account beim Anbieter hat und die API als Produkt auswählt. Der Prozess zur Erstellung eines Accounts ist bei jedem ICR-Anbieter unterschiedlich, und die Art und Menge der Credentials variiert ebenfalls.

ABBYY Cloud

Link für Account-Erstellung: <https://cloud.ocrsdk.com/Account/Register>

Google Vision API

Link für Account-Erstellung: <https://console.cloud.google.com/freetrial>

Microsoft Cognitive Services

Bei Microsoft Cognitive Services ist ein Azure-Account notwendig, Microsoft Cognitive Services ist dann ein Teil des Azure-Services.

Link für Account-Erstellung: <https://azure.microsoft.com/en-us/free/cognitive-services/>

Amazon Rekognition

Es wird ein allgemeiner AWS-Account benötigt, um Amazon Rekognition zu nutzen.

Link für Account-Erstellung: <https://aws.amazon.com/rekognition/>

Credential File Endstruktur

Die Struktur muss schlussendlich wie folgt aufgebaut sein («credential_store.json»):

```
{
  "ab": {
    "applicationId": "<Value here>",
    "password": "<Value here>"
  },
  "az": {
    "region": "<Value here>",
    "key": "<Value here>"
  },
  "gl": {
    "type": "<Value here>",
    "project_id": "<Value here>",
    "private_key_id": "<Value here>",
    "private_key": "<Value here>",
    "client_email": "<Value here>",
    "client_id": "<Value here>",
    "auth_uri": "<Value here>",
    "token_uri": "<Value here>",
    "auth_provider_x509_cert_url": "<Value here>",
    "client_x509_cert_url": "<Value here>"
  },
  "rk": {
    "aws_access_key_id": "<Value here>",
    "aws_secret_access_key": "<Value here>"
  }
}
```

Aufsetzen von Very Chess

Das Projekt Very Chess ist standardmässig aufgebaut nach Python-Standards mit Flask, folgend steht eine Schritt-für-Schritt-Anweisung zum Setup. Die Credential Files («credential_store.json») müssen von den verschiedenen ICR-Anbietern erhoben werden (siehe oben).

How to setup (Windows)

1. Clone Repository
2. Create venv
 - i. `python -m venv venv`
 - ii. `.\venv\Scripts\activate`
3. Install requirements.txt
 - i. `pip install -r requirements.txt`
4. Create launch configuration
5. Copy credential files
6. Start the flask server

Für die Launch Configuration kommt es auf die Entwicklungsumgebung an, wie diese erstellt wird. Jede bekannte Entwicklungsumgebung sollte Flask als ein Standard-Template anbieten. Für diese Arbeit wurde Visual Studio Code dazu verwendet.

Zuletzt muss «credential_store.json» in das Hauptverzeichnis von Very Chess kopiert werden.

ICR Preview Tools

Für die ICR Preview Tools müssen im Code ebenfalls die Credentials nachgeführt werden. Die ICR Preview Tools sind im Verzeichnis «Unversioned_Subproject_Tools» beinhaltet. Die Credentials müssen dann bei jeweils «./js/chessRecognition.js» geändert werden. Um die Projekte zu starten, muss lediglich die «index.html»-Datei direkt mit einem Browser geöffnet werden.

Zug-Korrektur Analyse Tool

Das Zug-Korrektur Analyse Tool ist ein einfaches Python-Projekt mit «app.py» als auszuführendes Hauptskript. Die anderen zwei Module «confidence.py» und «confusion.py» sind eingebunden und sollten nicht geändert werden, ausser man möchte wirklich die Funktionalität der Zug-Korrektur ändern, nicht nur evaluieren/analysieren.

Alle Parameter für die Analyse sind im Code direkt eingebaut, folgend werden diese beschrieben:

Parameter	Beschreibung
option_write_csv	Generiert eine CSV für die Confusion wenn auf True
option_is_correction_mode	Legt fest, ob manuelle Korrektur wie vorgesehen eingesetzt wird
option_print_mode	Legt fest, welche Art von Ausgabe in der Konsole ausgegeben wird <ul style="list-style-type: none"> • [0] Eigene Ausgaben (siehe unten) • [1] Spiele und Total (Leistungsanalyse) • [2] Nur Züge (Kopierfreundlich)
option_print_total	Nur für eigene Ausgabe. Legt fest ob am Schluss noch das Total aller Spieler ausgegeben wird
option_print_games	Nur für eigene Ausgabe. Legt fest ob pro Spiel eine Ausgabe erfolgt
option_print_moves	Legt fest, ob für jeden einzelnen Zug eine Ausgabe Erfolg
option_print_only_errors	Wenn auch festgelegt mit «option_print_moves», werden nur die Fehler ausgegeben
param_ocrs_direct_hit_bonus_factor	Parameter für Bonus, wenn ICR direkt legalen Zug abbildet
param_ocr_weights	Gewichte für die einzelnen ICRs
param_skip_threshold	Skip-Threshold für Skipping-Algorithmus
ocrs	Liste mit verwendeten OCRs, bei Änderungen des Ensembles anpassen
ocr_confusion_params	Liste mit verwendeten Minimum-Confusion-Value für jede ICR einzeln

Als Eingabe für das Tool muss im Verzeichnis noch die generierte CSV-Datei des CSV-Generator in das relative Verzeichnis "data/Game_data_dict.csv" eingefügt werden (Namen übernehmen).

CSV-Generator

Der CSV-Generator ist benötigt, um die Eingabe für das Zug-Korrektur Analyse Tool zu generieren. Es ist zu beachten, dass das Verzeichnis Very-Chess-Subprojects im gleichen Verzeichnis wie Very-Chess sein muss, da dieses Projekt von Very-Chess abhängig ist (relativer Pfad '../..../Very-Chess').

Folgend werden die relevanten Parameter für den CSV-Generator erläutert, die direkt im Code eingebaut sind:

Parameter	Beschreibung
RANGE	Legt fest, welche Schach-Formulare bearbeitet werden, mit inklusiven Grenzen. (Default [1, 50])
FILE_PREFIX	Sagt aus, welchen Prefix die Eingabe-Bilder haben müssen, um bearbeitet zu werden.

Als Eingabe für den CSV-Generator müssen im relativen Verzeichnis «./images» die Bilder abgelegt werden, diese folgen dem Muster «PREFIX_X», also der Prefix gefolgt von Bodenstrich und dann nummeriert (gestartet mit 1). Weiter müssen im Verzeichnis «./pgns» die entsprechenden PGN-Dateien abgelegt werden mit demselben Muster. Schlussendlich generiert das Tool die Datei «Game_data_dict.csv» im Hauptverzeichnis des Tools.

11.2.2 Deployment

Folgende Anweisungen sind eine Kopie der vorherigen Arbeit, sind aber immer noch gültig.

Follow:

- <https://apu.cloudlab.zhaw.ch>
- <https://www.youtube.com/watch?v=YFBRVJPhDGY>
- <https://stackoverflow.com/questions/39418012/my-apache-wsgi-flask-web-app-cannot-import-its-internal-python-module>

to have as a backup when configuring the mod_wsgi. You can also choose another web server and gateway of choice like Nginx or similar.

Installation

1. Set up server like documented on cloudlab.

- Forward port 80 (or 443 if needed) to the server.

2. Install Python and Pip (sudo apt-get install python)

- sudo apt update
- sudo apt install software-properties-common
- sudo add-apt-repository ppa:deadsnakes/ppa
- sudo apt install pythonX.Y (<-- enter version)
- sudo apt install python3-pip

3. run: sudo pip3 install -r requirements.txt (located in base of repository)

UBUNTU ONLY

4. Follow: <https://www.youtube.com/watch?v=YFBRVJPhDGY> tutorial on how to deploy with apache wsgi.

- NOTE: <https://stackoverflow.com/questions/39418012/my-apache-wsgi-flask-web-app-cannot-import-its-internal-python-module> answer to configure the YOURAPPNAME.wsgi file, otherwise it will not work correctly.

5. Connect to the server via the public IP-Address and use your page!

11.2.3 Dateistruktur

Es folgt der Aufbau der Verzeichnisstruktur von Very Chess.

```

|   README.md
|   requirements.txt
|   __init__.py
|
+---model
|   |   pipeline.py
|   |   recognition.py
|   |   __init__.py
|   |
|   +---helper
|   |   |   enumClasses.py
|   |   |   exceptions.py
|   |   |   geometry.py
|   |   |   jsonParser.py
|   |   |   __init__.py
|   |
|   +---ocr
|   |   |   __init__.py
|   |   |
|   |   +---abbyy
|   |   |   |   abbyyCloudProcess.py
|   |   |   |   AbbyyOnlineSdk.py
|   |   |   |   __init__.py
|   |   |
|   |   +---aws
|   |   |   |   awsRekognitionProcess.py
|   |   |
|   |   +---azure
|   |   |   |   azureVisionProcess.py
|   |   |   |   __init__.py
|   |   |
|   |   \---google
|   |   |   |   googleVisionProcess.py
|   |   |   |   __init__.py
|   |
|   +---preprocessing
|   |   |   box_detection.py
|   |   |   leadingNumberRemover.py
|   |   |   preProcessImage.py
|   |   |   removeUnusedBoxes.py
|   |   |   splitTableInBoxes.py
|   |   |   __init__.py
|   |
|   \---scanner
|   |   |   scanner.py
|   |   |   transform.py
|   |   |   __init__.py
|   |
+---static
|   +---images
|   |   +---favicon
|   |   |   |   favicon.ico
|   |
|   |
|   |

```

```
| | +---icons
| | |   anleitung.png
| | |   arrow.png
| | |   logo.png
| | |   move.svg
| | |   newlogo.png
| | +---processedUploads
| | |   aligned_template.PNG
| | |   select_last.gif
| | |   templatemove.png
| | |   unselect_boxes.gif
| | \---uploads
| | |   README.md
| +---js
| | |   confirmBoxes.js
| | |   editorPage.js
| | |   jquery.js
| | |   upload.js
| | \---Move_corrector
| | |   |   confidence.js
| | |   |   corrector.js
| | |   |
| | |   \---corrector_libs
| | |       chess.js
| \---styles
| | |   base.css
| | |   confirmBoxes.css
| | |   editorPage.css
| | |   style.css
| | |   uploadPage.css
| | \---partials
| | |   infoBox.css
| | |   loadOverlay.css
| | |   tournamentForm.css
| \---templates
| | |   base.html
| | |   confirmBoxes.html
| | |   editorPage.html
| | |   error.html
| | |   uploadPage.html
| | \---partials
| | |   infoBox.html
| | |   loadOverlay.html
| | |   tournamentForm.html
```

11.2.4 Confusion-Matrizen

ABBYY Cloud

	#	+	-	1	2	3	4	5	6	7	8	=	B	K	N	O	Q	R	a	b	c	d	e	f	g	h	x
#	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
+	0	68	0	5	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-	0	0	65	0	0	0	0	1	0	1	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	145	2	0	3	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
2	0	0	1	8	149	1	0	7	1	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	29	70	272	0	6	1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0
4	0	1	0	30	27	1	342	9	5	6	2	1	0	7	0	0	0	0	1	0	1	1	2	2	3	0	3
5	0	0	0	3	36	11	0	378	3	0	4	1	0	0	0	0	0	0	0	2	2	0	1	0	0	0	0
6	0	0	0	53	11	0	1	11	345	1	1	0	1	4	0	0	0	1	0	2	0	0	0	0	0	0	0
7	0	1	0	6	23	5	1	1	0	182	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	18	11	0	0	5	2	0	144	0	0	1	0	0	0	0	0	1	0	0	1	0	1	0	0
=	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	1	6	0	0	0	2	0	2	0	0	362	2	1	0	0	23	0	0	1	0	1	0	1	0	1
K	0	0	0	1	0	0	0	0	0	0	0	0	1	113	1	0	0	1	0	0	0	0	0	0	0	0	1
N	0	0	0	3	0	0	0	0	0	1	0	0	6	41	446	0	0	3	0	0	0	0	0	0	0	0	1
O	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	127	0	1	1	0	2	0	4	0	1	0	0
Q	0	0	0	1	9	1	0	1	0	0	1	0	38	4	1	1	237	18	4	0	0	0	1	0	2	0	0
R	0	0	0	2	0	0	0	0	0	1	0	0	48	12	0	0	0	260	0	0	0	0	0	1	0	0	0
a	0	0	0	3	3	0	0	0	0	0	0	0	0	3	0	0	0	0	167	2	2	11	3	0	6	1	1
b	0	0	0	4	0	0	0	4	14	1	0	0	0	2	0	0	0	1	0	248	1	1	0	1	0	11	0
c	0	0	0	4	1	0	0	3	1	1	0	0	2	2	1	1	0	0	6	2	423	0	15	5	6	1	1
d	0	0	0	6	1	1	0	2	0	1	0	0	0	0	0	2	0	0	17	9	1	477	3	3	2	4	0
e	0	0	0	0	3	1	0	0	1	1	2	0	2	0	0	0	0	2	5	5	20	1	416	3	5	0	0
f	0	0	0	2	1	0	0	0	3	6	1	0	2	3	0	0	0	1	0	5	8	1	1	302	1	1	0
g	0	0	1	0	4	2	0	1	0	0	0	0	1	0	0	0	0	0	21	5	5	0	4	2	164	1	0
h	0	0	0	2	0	0	0	2	0	0	0	0	0	2	0	0	0	0	0	33	1	0	0	0	0	149	0
x	0	0	0	0	4	1	2	6	1	4	4	0	0	0	0	0	0	0	2	0	5	1	0	0	0	0	465

Google Vision API

	+	-	1	2	3	4	5	6	7	8	=	B	K	N	O	Q	R	a	b	c	d	e	f	g	h	x
#	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
+	51	0	0	0	0	4	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
-	0	75	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	115	0	0	2	0	0	0	0	0	0	0	0	0	0	4	0	0	7	0	0	0	1	0	
2	0	0	0	179	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	1	0	0	0	0	
3	0	0	1	1	389	0	0	0	0	0	1	0	0	0	0	0	2	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	407	0	0	0	0	0	0	0	0	0	15	0	0	1	9	0	0	5	0		
5	0	0	0	0	3	0	366	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0		
6	0	0	0	0	0	0	0	315	0	0	0	1	0	1	0	0	58	0	2	5	1	1	0	0		
7	0	0	0	1	0	0	0	0	198	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	
8	0	0	0	0	0	0	1	0	156	0	1	0	0	1	1	0	2	0	2	5	4	2	1	0		
=	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
B	0	0	0	0	8	0	2	3	0	4	0	474	0	0	0	3	0	1	0	0	0	0	0	0	0	
K	0	0	1	0	0	0	0	0	0	0	0	96	0	0	0	0	0	1	0	0	0	0	0	0	0	
N	0	0	3	0	0	0	1	0	0	0	0	0	534	0	0	0	0	0	0	0	0	0	0	0	1	
O	0	0	2	0	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	
Q	0	0	0	2	0	0	0	4	0	0	2	0	0	7	321	0	13	0	0	0	0	0	0	0	0	
R	0	0	1	1	0	0	0	0	1	0	0	1	0	0	0	364	1	0	0	0	2	0	0	0	0	
a	0	1	0	4	0	0	0	0	0	0	0	0	0	0	0	205	0	0	5	0	0	0	0	0	0	
b	0	0	2	1	2	0	2	201	0	0	0	0	0	2	0	0	0	105	0	2	0	0	0	0	0	
c	0	0	3	1	0	0	0	9	0	0	0	0	0	1	0	0	3	0	332	0	24	0	0	0	0	
d	0	0	7	4	0	0	0	0	0	0	0	0	0	1	0	0	26	0	3	516	5	0	0	0	0	
e	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	531	0	0	0	1	
f	10	0	5	1	0	16	0	9	5	0	0	0	0	0	0	0	0	0	0	6	293	0	0	0	0	
g	0	0	0	0	7	2	0	2	0	1	0	0	0	0	1	0	14	1	0	0	8	0	156	0	0	
h	0	0	4	2	1	6	5	11	0	0	0	0	0	0	0	0	0	1	0	0	0	0	150	0	0	
x	1	0	1	0	0	1	0	5	0	1	0	0	0	0	0	1	0	3	0	0	0	3	0	0	534	

Azure Cognitive Services

	#	+	-	1	2	3	4	5	6	7	8	=	B	K	N	Q	R	a	b	c	d	e	f	g	h	x
#	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
+	1	60	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-	0	0	77	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	68	0	0	2	0	0	12	0	0	0	0	0	0	0	3	0	0	6	0	0	0	0	0
2	0	0	0	0	142	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	382	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	323	0	0	0	0	0	0	0	1	0	0	4	0	1	0	0	9	1	0	0
5	0	0	0	2	0	0	0	328	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	284	0	0	0	0	0	0	0	0	0	67	0	0	0	4	0	2	0
7	0	0	0	0	1	0	0	0	0	147	0	0	0	0	0	0	0	0	0	0	0	1	6	0	0	1
8	0	0	0	0	0	1	0	0	1	0	128	0	2	0	0	0	0	1	6	0	2	3	5	1	0	0
=	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	4	0	6	0	1	0	0	1	0	438	0	0	0	1	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	0	106	0	0	1	0	0	0	0	0	0	0	0	0
N	0	0	0	3	0	0	0	0	0	0	0	0	0	1	493	0	0	0	0	0	0	0	0	0	0	0
O	0	0	0	0	0	0	0	1	2	1	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0
Q	0	0	0	1	9	1	0	0	0	0	3	0	1	0	0	265	2	8	0	0	2	0	0	0	0	0
R	0	0	0	2	2	0	0	0	0	0	0	0	2	9	0	1	326	0	0	0	0	0	0	0	0	0
a	0	0	0	0	7	0	3	0	0	0	0	0	0	0	0	0	0	168	0	1	2	0	0	1	0	0
b	0	0	0	0	1	0	1	8	164	0	1	0	1	0	0	0	0	0	90	0	0	1	0	0	0	0
c	0	0	0	1	3	0	1	1	3	0	0	0	0	0	0	0	0	0	0	234	0	11	0	0	0	4
d	0	0	0	2	10	0	2	1	0	0	1	0	0	0	0	0	0	8	0	1	433	0	0	0	0	0
e	0	0	0	0	16	0	0	0	0	0	1	0	0	0	0	0	0	0	5	0	463	0	0	0	0	0
f	0	15	0	4	0	0	5	0	2	1	2	0	0	1	0	0	0	0	0	0	3	233	0	0	0	0
g	0	0	0	0	1	3	4	1	0	0	2	0	0	0	0	0	0	8	0	1	0	3	0	107	0	0
h	0	0	0	10	0	0	3	3	8	1	0	0	0	0	0	0	0	4	0	0	0	0	0	128	0	0
x	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	0	0	1	0	0	0	0	488

Amazon Rekognition

	+	-	1	2	3	4	5	6	7	8	=	B	K	N	O	Q	R	a	b	c	d	e	f	g	h	x
+	22	0	0	0	0	2	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
-	0	63	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	73	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
2	0	0	1	110	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0
3	0	0	0	0	303	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	294	0	0	0	0	0	0	0	0	0	0	0	5	0	1	0	0	4	3	1	0
5	0	0	0	0	0	0	278	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	1	0	0	0	228	0	0	0	0	0	0	0	0	0	0	81	0	0	0	2	0	1	0
7	0	0	0	0	0	0	0	0	180	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
8	0	0	0	0	0	0	0	0	0	109	0	1	0	0	0	0	0	0	2	0	2	0	0	3	0	1
=	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	7	0	0	2	0	0	0	349	0	0	0	0	0	0	1	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	0	0	51	0	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	1	0	0	0	0	0	0	0	0	0	405	0	0	0	0	0	0	0	0	0	0	0	0
O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	18	0	0	0	0	0	0	0	0	0	0	0
Q	0	0	0	9	1	0	0	0	0	0	0	1	0	0	3	222	7	3	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	242	0	0	0	0	1	0	0	0	0
a	0	0	1	3	0	0	0	0	0	0	0	0	0	1	0	0	0	124	0	0	2	0	0	0	0	0
b	0	0	1	0	1	1	4	119	0	1	0	1	0	0	0	0	0	0	85	0	0	0	0	0	0	0
c	0	0	2	0	0	0	1	0	0	0	0	0	0	0	2	0	0	1	0	232	0	15	0	0	0	1
d	0	0	10	1	0	1	1	0	0	0	0	0	0	0	1	0	4	0	0	392	0	0	0	0	0	0
e	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0	362	0	0	0	0	0
f	10	0	3	0	0	3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	204	0	0	0
g	0	0	0	3	1	0	1	0	0	0	0	0	0	0	0	0	0	24	0	1	0	0	0	74	0	1
h	0	0	0	0	0	3	0	1	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	128	0
x	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	379

11.2.5 Datenset

Im Folgenden sind die Informationen zum Datenset ersichtlich.

Games	Stiftfarbe	Länge (Zug-Paare)	Leserlichkeit	Layout	Autor
1	blau	32	gut	Regency chess company	Bernt
2	blau	27	mittel	Regency chess company	Bernt
3	blau	23	mittel	Regency chess company	Bernt
4	bleistift	23	mittel	Regency chess company	Bernt
5	bleistift	31	gut	Regency chess company	Bernt
6	blau	36	gut	Regency chess company	Bernt
7	blau	38	gut	Regency chess company	Bernt
8	rot	25	gut	Regency chess company	Bernt
9	blau	33	gut	Regency chess company	Bernt
10	grün	59	gut	Regency chess company	Bernt
11	blau	27	gut	Regency chess company	Volkan
12	blau	35	gut	Regency chess company	Volkan
13	rot	26	mittel	Regency chess company	Volkan
14	rot	39	mittel	Regency chess company	Volkan
15	schwarz	24	gut	Regency chess company	Volkan
16	schwarz	54	gut	Regency chess company	Volkan
17	blau	46	gut	Regency chess company	Volkan
18	blau	32	mittel	Regency chess company	Volkan
19	bleistift	28	mittel	Regency chess company	Volkan
20	bleistift	22	mittel	Regency chess company	Volkan
21	blau	40	gut	Regency chess company	Person A
22	blau	30	mittel	Regency chess company	Person A
23	rot	34	mittel	Regency chess company	Person A
24	blau	32	gut	Regency chess company	Person A
25	schwarz	29	gut	Regency chess company	Person B
26	bleistift	31	schlecht	Regency chess company	Person B
27	rot	24	gut	Regency chess company	Person B
28	rot	25	gut	Regency chess company	Person B
29	blau	29	gut	Regency chess company	Person C
30	bleistift	32	mittel	Regency chess company	Person C
31	bleistift	33	gut	Regency chess company	Person C
32	rot	35	mittel	Regency chess company	Person C
33	blau	28	schlecht	Regency chess company	Person D
34	rot	27	mittel	Regency chess company	Person D
35	blau	33	gut	Regency chess company	Person E
36	bleistift	33	gut	Regency chess company	Person E
37	bleistift	33	mittel	Regency chess company	Person E
38	blau	35	gut	Regency chess company	Person E
39	blau	30	mittel	Regency chess company	Person F
40	blau	55	mittel	Regency chess company	Person F
41	blau	37	mittel	Regency chess company	Person F
42	blau	45	mittel	Regency chess company	Person F
43	schwarz	40	mittel	Gelbes Scoresheet	Prior BA
44	blau	26	gut	Chess Scoresheet simple 1	Prior BA
45	blau	20	gut	Chess Scoresheet simple 2	Prior BA
46	blau	21	gut	Regency chess company	Prior BA
47	bleistift	23	mittel	Chess Scoresheet simple 3	Prior BA
48	blau	21	gut	Regency chess company	Prior BA
49	blau	40	gut	Chess Scoresheet colorful	Prior BA
50	blau	19	gut	Chess Scoresheet simple 4	Prior BA

Länge (Zug-Paare)	Anzahl
durchschnitt	32
median	31.5
min	19
max	59

Leserlichkeit	Anzahl
gut	28
mittel	20
schlecht	2
Total	50

Autor	Anzahl
Bernt	10
Volkan	10
Person A	4
Person B	4
Person C	4
Person D	2
Person E	4
Person F	4
Prior BA	8
Total	50
Distinct	9

Stiftfarbe	Anzahl
blau	27
bleistift	10
rot	8
schwarz	4
grün	1
Total	50

Layout	Anzahl
Regency chess company	44
Gelbes Scoresheet	1
Chess Scoresheet simple 1	1
Chess Scoresheet simple 2	1
Chess Scoresheet simple 3	1
Chess Scoresheet simple 4	1
Chess Scoresheet colorful	1
Total	50
Distinct	7