Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

CAI Centre for Artificial Intelligence

Bachelorarbeit Informatik

Aufbau einer Evaluierungspipeline für die Schachspiel-Digitalisierungssoftware ChessReader

Autoren	Nadine Moser Robin Meier
Hauptbetreuung	Prof. Dr. Mark Cieliebak

Datum	07.06.2024

Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

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.
Schliesslich sei erwähnt, dass generative KI-Systeme bzw. KI-Tools in verschiedenen Prozessphasen dieser Arbeit zum Einsatz kamen. Es wurde ChatGPT für die Verbesserung der Formulierungen des Texts und zur Unterstützung beim Schreiben von Programmcode verwendet.

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:	**Name Studierende:**
Zürich, 07.06.2024	Nadine Moser
Zürich, 07.06.2024	Robin Meier

1 Zusammenfassung

ChessReader ist eine Applikation, welche es ermöglicht, von Hand ausgefüllte Schachblätter mithilfe von optischer Zeichenerkennung (OCR) zu digitalisieren. Sie wurde im Rahmen von mehreren Projekt- und Bachelorarbeiten an der Zürcher Hochschule für angewandte Wissenschaften aufgebaut. Bisher wurden die weiterentwickelten Komponenten von ChessReader isoliert daraufhin geprüft, ob sie das Produkt verbessern. Es existiert keine genormte Methode, um den Einfluss von Änderungen auf den gesamten Verarbeitungsprozess der Schachblätter zu analysieren.

Die vorliegende Arbeit befasste sich mit der Entwicklung einer Pipeline, die eine vollständige, automatisierte Evaluierung von ChessReader anhand eines Corpus von Schachblättern ermöglicht. Für die Evaluierung wurden spezifische Metriken definiert, welche die Ausgabe von ChessReader in verschiedenen Aspekten bewerten. Darüber hinaus wurde die Schnittstelle zu ChessReader überarbeitet, um die Evaluierung zu ermöglichen. Ein bestehender Corpus an Schachblättern wurde neu strukturiert und um zusätzliche Meta-Eigenschaften ergänzt.

Der Evaluierungsprozess beginnt mit dem Einlesen des überarbeiteten Corpus. Die enthaltenen Schachblätter werden in ChessReader importiert und durchlaufen eine Zeichenerkennung. Anschliessend werden die definierten Metriken über die erhaltenen Erkennungen angewendet. Die Ergebnisse werden mit Diagrammen und Tabellen visualisiert und in einem Report im PDF Format zusammengefasst.

Die Evaluierungspipeline leistet einen wesentlichen Beitrag zur Qualitätssicherung von ChessReader. Sie bietet eine strukturierte Methode zur Evaluierung und schafft eine Grundlage für zukünftige Verbesserungen und Erweiterungen der Applikation.

2 Abstract

ChessReader is an application, which allows the digitization of handwritten chess scoresheets with the use of optical character recognition (OCR). It was developed as part of several project- and bachelor theses at the Zurich University of Applied Sciences. Up until now, newly developed components of ChessReader have been tested in isolation to see wether they improve the product. At this point in time there is no standardized method, to analyze the impact of changes on the overall processing of scoresheets.

This thesis focuses on the development of a pipeline that enables a complete, automated evaluation of ChessReader using a corpus of scoresheets. Specific metrics were defined for the evaluation, which assess the output of ChessReader in various aspects. Additionally, the interface to ChessReader was revised to enable the evaluation and an existing corpus of scoresheets was restructured and supplemented with additional meta properties.

The evaluation process begins by first loading the revised corpus. The scoresheets contained are imported into ChessReader and run through the character recognition process. The defined metrics are then applied to the obtained detections. The results are visualised using diagrams and tables and get summarized in a report in PDF formatting.

The evaluation pipeline makes a large contribution to the quality assurance of ChessReader. It provides a structured method for evaluation and lays the groundwork for future improvements and extensions of the application.

3 Vorwort

Wir, Robin Meier und Nadine Moser, absolvierten bereits mehrere gemeinsame Projekte während dem Studium. Für uns war beide schnell klar, wir würden die Bachelorarbeit gerne im Team erarbeiten. Während der Themenwahl sprang uns ChessReader ins Auge, wir konnten allerdings noch nicht genau einschätzen welche Schwerpunkte ein Projekt darin umfasst. Ein Gespräch mit Prof. Dr. Mark Cieliebak verschaffte Klarheit und unser Ziel von einer Evaluierungspipeline stand fest.

In diesem Sinne möchten wir unseren herzlichen Dank an Prof. Dr. Mark Cieliebak aussprechen. In den wöchentlichen Meetings unterstützte er uns mit seinem Fachwissen und auch ausserhalb war er schnell und zuverlässig erreichbar. Weiteren Dank gilt unserem Umfeld, welches uns in verschiedenen Arten und Formen stetig unterstützt hat.

Inhaltsverzeichnis

4 Einleitung

In einem Schachspiel werden getätigte Spielzüge handschriftlich auf einem Schachblatt (englisch "Scoresheet") festgehalten. Abbildung 1 zeigt ein Beispiel hierfür von Bobby Fischer aus dem Jahr 1970. Schachblätter gelten bei Spielen der International Chess Federation (FIDE) als offizielle Aufzeichnung [1] und werden auch verwendet, um Spiele nachträglich zu Trainingszwecken analysieren zu können [2]. Oft werden die Züge von einem Schachblatt manuell in einer Schach-Engine nachgetragen, um von dieser eine Beurteilung zu erhalten. Dieser Digitalisierungsprozess ist jedoch zeitintensiv und fehleranfällig. ChessReader löst dieses Problem. Auf der Webseite ChessReader.org können Spielenende ein Scoresheet hochladen, auf welchem die handschriftlichen Züge erkannt werden. Der Benutzer oder die Benutzerin kann die erkannten Halbzüge bestätigen und bei Bedarf anpassen. Die Schachzüge können dann jederzeit im standardisierten Portable Game Notation (PGN) Format exportiert werden. Der Erkennungsmechanismus basiert auf drei Hauptkomponenten: Ein Boxerkennungsalgorithmus, welcher die Halbzüge auf dem Scoresheet lokalisiert, ein Optical Character Recognition (OCR) Ensembling [3] aus mehreren externen OCR-APIs, welche die in den Boxen enthaltene Schrift erkennen und eine Schach-Engine, welche die Erkennungen der APIs konsolidiert.

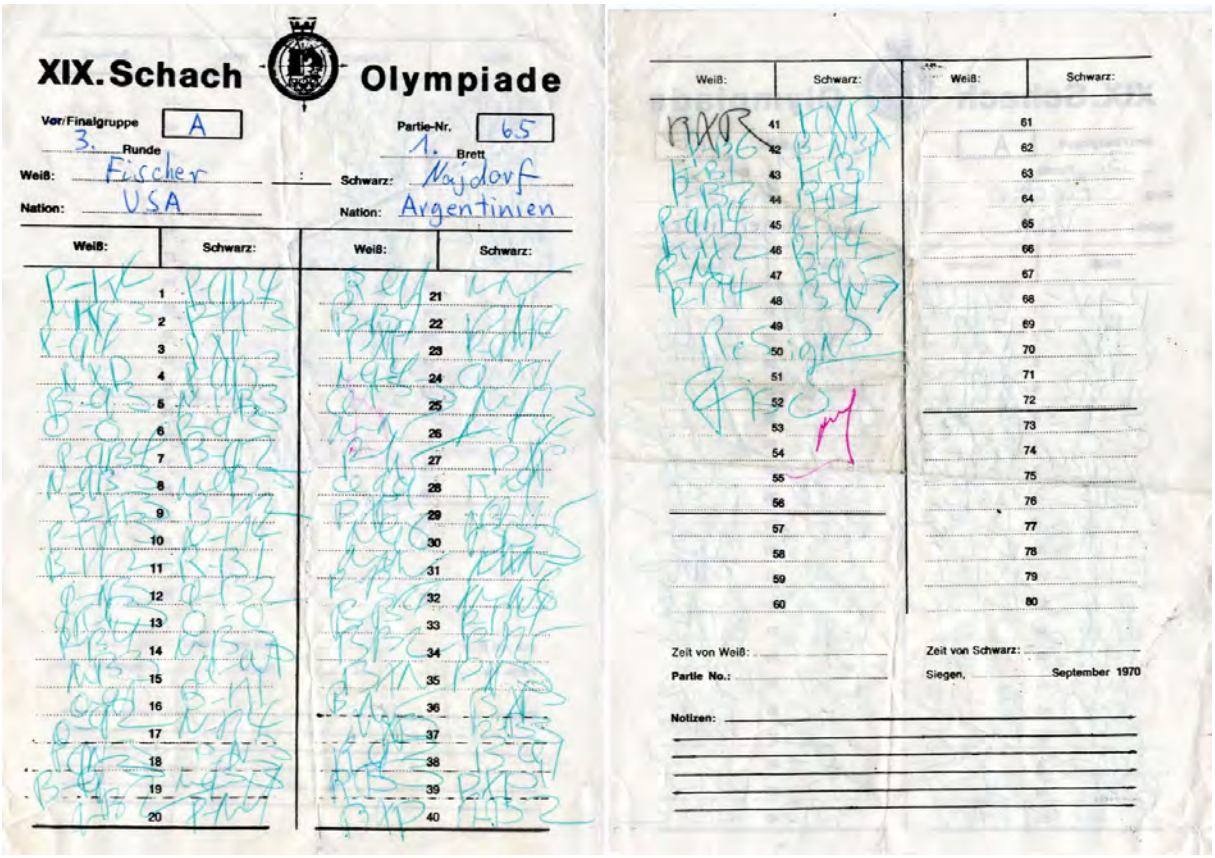


Abbildung 1: Bobby Fischers Schachblatt aus einer Partie gegen Miguel Najdorf an der Schacholympiade 1970 in Siegen. [4]

In vergangenen Projekt- und Bachelorarbeiten [5][6][7][8][9][10] wurde das Produkt in diversen Aspekten erweitert und noch besser auf die Nutzung abgestimmt. Die Arbeiten hatten den Fokus auf der User Experience, der Struktur oder der Zeichenerkennung.

4.1 Zielsetzung

Während der Verwendung von ChessReader und dem Durchlesen von vorherigen Arbeiten darüber, stellte sich vermehrt die Frage wie gut ChessReader Schachzüge erkennen kann. Lohnt sich die Verwendung des Tools oder ist es immer noch einfacher und angenehmer Schachblätter von Hand zu digitalisieren? Vorherige Bachelorarbeiten zeigten in ihrem Teilgebiet anhand diversen Metriken auf, dass ChessReader besser wie zuvor funktioniert, führten aber selten eine Bewertung des kompletten Verarbeitungsprozesses von Scoresheets bis hin zur resultierenden Zeichenerkennung mit einem standardisierten Test-Datenset durch. Dies führte dazu, dass sich ChessReader im Laufe seiner Entwicklung in einem Teilbereich verbesserte, sich jedoch als Nebenwirkung durch die Änderungen unbewusst in einem anderen verschlechterte. Die Schwierigkeit die aktuelle Genauigkeit der Scoresheetverarbeitung einzuschätzen und die Ungewissheit, ob Änderungen negative Einflüsse auf andere Teilaspekte der Verarbeitungspipeline von ChessReader haben, erschwert die Weiterentwicklung des Produkts massiv und nimmt Entwicklern und Entwicklerinnen die Sicherheit, Änderungen vorzunehmen.

In diesem Sinne setzt sich diese Arbeit mit dem Aufbau einer Evaluierungspipeline für ChessReader auseinander. Es soll ein einfach zu verwendendes Werkzeug den zukünftigen Entwicklern und Entwicklerinnen zur Verfügung gestellt werden, mit welchem von ChessReader auf Knopfdruck eine vollständige Evaluierung möglich ist. Die Evaluierung soll anhand mehrerer Metriken, eine Aussage über die Genauigkeit des Produkts liefern. Im Fokus steht dabei die technische Hauptfunktionalität, sprich der Verarbeitungsprozess von Scoresheets bis zur Ausgabe der erkannten Halbzüge. Dies beinhaltet die Erkennung der Boxen in die Halbzüge geschrieben sind, das OCR Ensembling und die verwendete Schach-Engine. Nebenfunktionalitäten wie die visuelle Darstellung oder die Benutzerverwaltung sind dabei ausserhalb des Rahmens dieser Arbeit. Folgend werden die Anforderungen an die Evaluierungspipeline genauer beschrieben.

Funktionalitätsanforderungen

- Es soll möglich sein, einen Datensatz mit einer standartisierten Struktur an mehreren Schachblättern als Input für die Evaluierungspipeline zu verwenden, sodass für jedes Schachblatt ein Resultat von ChessReader generiert wird. Die Auswertung soll dann über alle Resultate durchgeführt werden, um einen Überblick über die durchschnittliche Genauigkeit zu erhalten.
- Es sollen aussagekräftige und detaillierte Evaluierungsmetriken definiert werden, welche die generierten Resultate von ChessReader auswerten und eine Aussage über die einzelnen Subkomponenten des Schachblatt-Verarbeitungsprozesses machen können.
- Ausgewertete Metriken sollen in einem Report mit Hilfe von entsprechenden Beschreibungen und diversen Diagrammen visualisiert werden. Der Report soll informativ sein und die wichtigsten Daten auf übersichtliche Weise darstellen.
- Die ausgewerteten Metriken und alle Zahlen, welche damit verbunden sind, sollen zudem in einem vom Menschen lesbaren Textformat zur Verfügung gestellt werden, um ergänzende Analyse einzelner Daten zu ermöglichen.
- Die Auswertung von Metriken soll mittels einer Konfigurationsdatei steuerbar sein. Die Auswertung einzelner Metriken soll aktivier- und deaktivierbar sein. Dies soll ermöglichen, Schwerpunkte im Report zu setzen, um gezielt Informationen auswerten zu können.

Nebenanforderungen

- Die Evaluierungspipeline soll unabhängig von ChessReader sein und nur über die zur Verfügung gestellten API-Schnittstellen damit interagieren. Dies, um die bestehende Codebase nicht komplexer zu machen und um den Evaluierungscode einfach von Produktivcode trennen zu können. Trotzdem soll der Evaluierungscode im gleichen Git-Repository gespeichert werden, um zusätzlichen Anwendungsaufwand der Pipeline zu minimieren.
- Die Evaluierungspipeline soll einfach in der Anwendung und der Erweiterbarkeit sein. Bei Bedarf an neuen Auswertungsmetriken sollen diese schnell in die bestehende Pipeline integriert werden können.
- Die Pipeline soll in einzelne Schritte aufgeteilt sein und Änderungen an einem Schritt sollen, wenn überhaupt, nur minimale Änderungen in anderen Schritten verursachen.
- Die Erweiterung von bestehenden Funktionalitäten von ChessReader, wie die Erweiterung des OCR Ensembles, soll in der Auswertung dynamisch berücksichtigt werden und keine Anpassung der Evaluierungspipeline erfordern, solange sich die API Schnittstellen von ChessReader nicht verändern.

4.2 State of the Art

Auf dem Markt gibt es bereits diverse Tools zur Evaluierung und Überwachung von Künstlichen Intelligenz (KI) und Machine Learning (ML) Modelen. Ein Beispiel hierfür ist die Open-Source Lösung Deepchecks[1]. Diese stellt vordefinierte "Checks" (Metriken) zur Verfügung, um während der Model Entwicklung, dem Deployment über Continuous Integration und in Produktion Anforderungen und Performanz eines Modells zu überprüfen. Deepchecks bietet Outputs in Jupyter Notebooks, HTML oder JSON (JavaScript Object Notation) an und ist gesamthaft ein gut strukturiertes Framework für das Testen und Evaluieren von einem einzelnen ML Model. Ähnliche Open-Source Alternativen sind Evidently AI[2] und MLflow[3]. All diese Produkte bieten ein breites Feld an Metriken und Visualisierungen an.

ChessReader besteht nicht nur aus ML Komponenten. Für eine aussagekräftige Analyse ist eine massgeschneiderte Auswertung erforderlich. Der Einsatz von bestehenden Tools würde erheblichen Mehraufwand bedeuten, um diese für den gewünschten Nutzen anzupassen. Aus diesem Grund wurde entschieden, eine eigene Implementierung zu entwickeln.

4.3 Lösungsansatz

Um die in Kapitel 4.1 definierten Ziele zu erreichen, wurde folgendes Konzept für die Evaluierungspipeline erstellt.

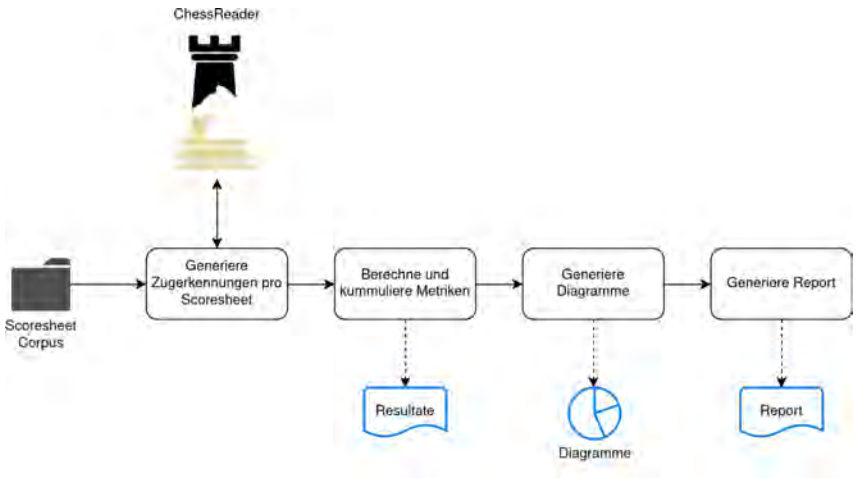


Abbildung 2: Übersicht über den Ablauf der Evaluierungspipeline

- Als Input der Pipeline wird ein Datensatz an Scoresheets mitgegeben.
- Die im Datensatz enthaltenen Scoresheets werden einzeln in ChessReader importiert, sodass dieser Erkennungen der Halbzüge für alle Scoresheets durchführt.
- Die von ChessReader generierten Erkennungen werden mit den Realwerten (Labels) aus dem Datensatz verglichen und anhand einiger Metriken bewertet. Die ausgewerteten Metriken werden in einer textbasierten Datei abgelegt.
- Basierend auf den Resultaten in der Textdatei, wird ein Report im PDF Format generiert, welcher Diagramme und Tabellen für die einzelnen Metriken beinhaltet.

[1] deepchecks.com
[2] evidentlyai.com
[3] mlflow.org

4.3.1 Technische Vorraussetzungen

Um die Pipeline wie oben beschrieben implementieren zu können, müssen folgende Vorraussetzungen erfüllt sein:

- Die Evaluierungspipeline muss in der Lage sein, ohne Benutzerintervention mehrere Scoresheets auf ChessReader hochzuladen und eine Zugerkennung dafür zu generieren. Da die Evaluierungspipeline nur von der API Schnittstelle von ChessReader abhängig sein soll, müssen hierfür die notwendigen Endpunkte zur Verfügung gestellt werden. Die API Endpunkte müssen es ermöglichen, einzelne Schritte der Scoresheet Verarbeitung in Isolation auszuführen.
- Um ergänzende Meta-Informationen von ChessReader, welche nicht bereits in der grafischen Weboberfläche angezeigt werden, auswerten zu können, müssen dafür zusätzliche API-Endpunkte implementiert werden.
- Ein Datensatz an Scoresheets muss in einem automatisch auswertbaren Format, verfügbar sein. Der Datensatz muss zusätzliche Eigenschaften über die Scoresheets und die Zugnotationen enthalten, sodass es möglich ist, anhand der Eigenschaften, analytische Schlussfolgerungen aus einer Evaluierung zu ziehen. Dies kann erreicht werden, indem der bereits existierende "Corpus v4" von Boner und Hostettler [10] überarbeitet wird.

5 Theoretische Grundlage

In diesem Kapitel werden notwendige theoretische Grundlagen erläutert. Im Laufe des Berichts werden die erwähnten Themen referenziert und angewandt.

5.1 Standard Algebraische Notation (SAN)

Die Standard Algebraic Notation oder auch kurz SAN ist eine Schreibweise, wie Schachzüge auf einem Schachblatt notiert werden können. Bei Spielen der FIDE wird die Verwendung der algebraische Notation im Handbuch "Laws of Chess" unter Artikel 8 [1] vorgeschrieben und in Appendix C [11] genauer erklärt.

SAN basiert auf einem Koordinatensystem wobei die Y-Achse aus Zahlen von 1 bis 8, beginnend auf der Seite von Weiss und die X-Achse aus Buchstaben von a bis h, von links nach rechts, besteht. Siehe hierfür Abbildung 3.

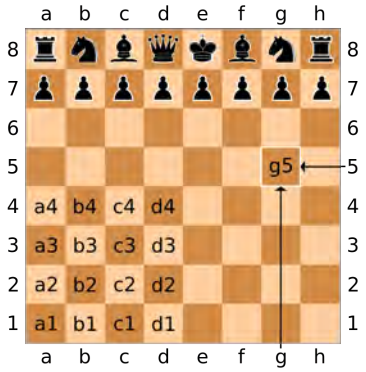


Abbildung 3: Koordinatensystem der algebraischen Notation [12]

Jede Figur, abgesehen vom Bauer, wird mit einer Abkürzung betitelt. Auf Englisch sind die Abkürzungen: K=King, Q=Queen, R=Rook, B=Bishop und N=Knight. Wichtig zu vermerken ist, dass diese Abkürzungen nicht sprachlich standartisiert sind. In der deutschen Notation werden die Figuren folgendermassen bezeichnet: K=König, D=Dame, T=Turm, L=Läufer, S=Springer.

SAN wird verwendet, um Halbzüge (in englisch "ply") zu notieren. Ein Halbzug wird hier definiert als eine einzelne Bewegung einer weissen Figur oder einer schwarzen Figur. Im Gegensatz dazu, ist ein Zug definiert als die Kombination eines Halbzuges einer weissen Figur und dem folgenden Halbzug einer schwarzen Figur. Ein Ausnahmefall ist es, wenn Weiss Schwarz Schachmatt setzt und Schwarz somit kein Halbzug mehr ausführt.

Die Notation eines Halbzuges besteht aus dem Buchstaben der Figur und dem Feld zu welchem sie sich bewegt. Für das Beispiel, dass sich ein Läufer auf das Feld "g6" bewegt, würde die dazugehörige SAN als "Bg6" geschrieben werden. Wird lediglich ein Feld vermerkt, so gilt diese Bewegung dem Bauern. Ein Bauer, welcher sich auf Feld "c3" bewegt, wird somit schlicht als "c3" notiert.

Für den Fall, dass zwei Figuren gleicher Art das selbe Zielfeld besetzen könnten, wird zusätzlich ihr Ursprungsfeld beschrieben. Wird der Turm auf dem Feld "h8" in Abbildung 4 auf das Feld "d8" verschoben, so muss dies als "Rhd8" notiert werden, um Verwechslungen zu vermeiden.

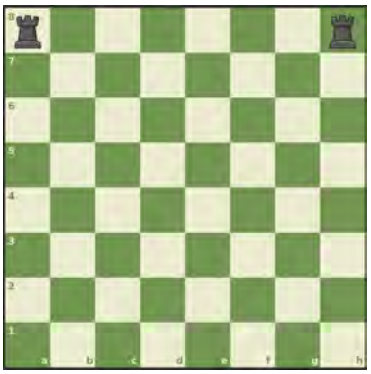


Abbildung 4: Beide Türme können sich auf das Feld "d8" bewegen

Folgend ist eine Auflistung weiterer Notationsspezifikationen:

- *Promotion*: "e8=Q"
- *Rochade, kurz und lang*: "0-0", "0-0-0"
- *Schlagen*: "Lxf4"
- *Schach*: "Rh8+"
- *Schachmatt*: "Qb2++" oder "Qb2#"
- *En Passant*: "exd6 e.p."

5.1.1 Portable Game Notation (PGN)

Um Schachspiele elektronisch in einer Chess Engine auszuwerten, gibt es diverse Dateiformate. Das Format, welches in ChessReader als Output verwendet wird, ist die sogenannte Portable Game Notation (PGN). Diese verwendet die oben erkläre algebraische Notation und fügt ein Header mit mehreren Metadaten hinzu, welche Auskunft über das Spiel beinhaltet. Ein Beispiel hierfür ist in Abbildung 5 gegeben.

```
[Event "ZHAW Chess Championships"]
[Site "Technikumstrasse 9, 8401 Winterthur"]
[Date "2024-01-06"]
[Round "1"]
[White "Nadine Moser"]
[Black "Robin Meier"]
[Result "1-0"]

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
```

Abbildung 5: Beispiel einer PGN Datei

5.2 JavaScript Object Notation (JSON)

Douglas Crockford gilt als Erfinder des JavaScript Object Notation Dateiformats [13], kurz als JSON bekannt. JSON ist ein leichtgewichtiges Dateiformat, welches von Mensch wie auch von Maschine leicht gelesen werden kann. Es ähnelt Programmiersprachen, wodurch es schnell verstanden ist, ist aber dennoch Programmiersprachen unabhängig. Auf Grund dieser Eigenschaften eignet es sich insbesondere für den Datenaustausch.

5.2.1 Struktur

Ein JSON wird mit einer geöffneten, geschweiften Klammern ({) geöffnet und mit einer schliessenden, geschweiften Klammer (}) geschlossen. Eine Datei im JSON-Format kann aus einem Objekt oder einer ungeordneten Liste bestehen.

Objekt
Ein Objekt ist eine Sammlung von ungeordneten Namen-Werte Paaren. Ein Name wird mit Anführungszeichen umschlossen, anschliessend wird ein Doppelpunkt geschrieben und zum Schluss der Wert platziert [14]. In dieser Arbeit wird der Begriff "Attribut" als Synonym für den Namen verwendet. Nach dem Google Standard [15] soll ein Name im "camelCase" Format geschrieben werden. Besteht ein Name aus einem Wort, so wird das Wort klein geschrieben. Besteht er aus mehreren Wörtern, so werden die Wörter zusammen gehängt. Bei jedem angehängten Wort wird der erste Buchstabe grossgeschrieben.

Ein Besipiel für Wort im camelCase Format ist: `camelCase`

Ein Wert kann eine Zeichenkette, eine Zahl, ein Objekt, eine ungeordnete Liste (siehe Ungeordnete Liste 5.2.1), `true`, `false` oder `null` sein. Paare werden mit einem Komma getrennt [14].

Folgende Visualisierung beschreibt die mögliche Zusammensetzung eines Objekts.

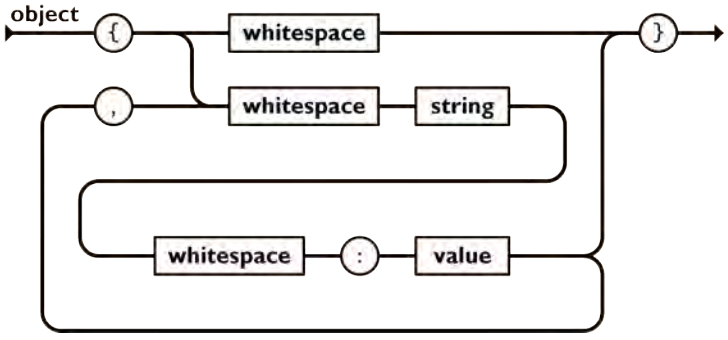


Abbildung 6: Ablauf für das Erstellen einer JSON Datei [16]

Ungeordnete Liste
Eine ungeordnete Liste, oft als Array bezeichnet, beginnt mit einer öffnenden, eckigen Klammer ([) und wird mit einer schliessenden, eckigen Klammer (]) geschlossen. Dazwischen werden die Werte geschrieben und mit einem Komma abgetrennt [14].

Ein Beispiel eines Arrays ist : `["erstens", "zweitens", "drittens"]`

5.3 Optical Character Recognition (OCR)

Optical Character Recognition beschreibt einen Prozess, um hand- oder computergeschriebene, alphanumerische Zeichen, innerhalb eines digitalen Bildes zu erkennen. In ChessReader wird OCR verwendet, um die einzelnen Halbzüge zu identifieren.

Eine einfache OCR Engine besteht zusammengefasst aus drei Schritten: Preprocessing, Text Erkennung und Post Processing. Im Preprocessing Schritt wird das Bild aufbereitet. Es werden Filter angewendet, welche die Erfolgschance der Texterkennung erhöhen sollen, wie Anpassen des Bildwinkels oder Anwendung eines Denoising Algorithmus. Die Texterkennung basiert auf einem Pattern Matching Algorithmus, bei welchem versucht wird Muster in der Erkennung bekannten Zeichen zuzuweisen. Im Post Processing Schritt werden eventuell notwendige Nachbearbeitungen der erkannten Daten durgeführt [17].

Ein performantes OCR System selber zu implementieren ist nicht trivial, weshalb ChessReader auf externe OCR Engines vertraut, welche ermöglichen ein Bild über eine Representational State Transfer (REST) API Schnittstelle hochzuladen und eine Erkennung zurück zu erhalten. Verschiedene Anbieter an OCR Engines funktionieren unterschiedlich. Bei einigen ist es möglich, spezifische Regionen innerhalb einer Datei anzugeben, in welchen eine Erkennung durchgeführt werden soll, andere generieren immer für ein komplettes Inputbild eine Erkennung. Teilweise unterscheiden die Anbieter auch zwischen regulärer Text-Erkennung und Dokument-Erkennung, welche auf das Einlesen von Dokumenten spezialisiert ist. Folgend eine kurze Auflistung, welche OCR Engines ChessReader verwendet und welche Funktionalitäten davon genutzt werden:

- *ABBY FineReader* [18] erlaubt das Angeben von Textfeldern, in welchen eine Erkennung durchgeführt werden soll. Zudem kann spezifiziert werden, welche Zeichen in der Erkennung vorkommen dürfen.
- *Amazon Rekognition* [19] führt eine Erkennung über den kompletten Input aus.
- *Azure AI Vision* [20] führt eine Erkennung über den kompletten Input aus.
- *Google Vision API* [21] bietet zwei Modi an: "Document Text Detection" und "Text Detection", wobei "Document Text Detection" auf dichte Texte und Dokumente optimiert ist. In ChessReader werden beide Modi verwendet. Beide Modi führen dabei eine Erkennung über den kompletten Input aus.

5.3.1 Evaluierung von OCR Systemen

Um die Genauigkeit von OCR Systemen messen zu können, werden im Normalfall zwei Metriken verwendet, die Character Error Rate (CER) und die Word Error Rate (WER). Sie sagen aus, welche Fehlerrate eine Erkennung im Vergleich zum Referenzwert aufweist.

Die Metriken werden über die Levenshtein Distanz [22] definiert als:

$$\text{WER} = \frac{S_w + D_w + I_w}{N_w} \tag{1}$$

$$\text{CER} = \frac{S_c + D_c + I_c}{N_c} \tag{2}$$

wobei:

- S_x die Anzahl der substituierten Wörter/Zeichen ist,
- D_x die Anzahl der gelöschten Wörter/Zeichen ist,
- I_x die Anzahl der eingefügten Wörter/Zeichen ist,
- N_x die Gesamtzahl der Wörter/Zeichen in der Referenz ist.

5.4 Corpus Design

John Sinclair veröffentlichte im Jahre 2004 Guidelines für den Aufbau eines Corpus [23]. Er geht dabei auf 10 Prinzipien ein. Die Guidelines sind für Corpora mit geschriebenen Texten als Inhalt ausgelegt, demnach werden sie nach bestem Wissen und Gewissen auf einen Datensatz für Schachblätter angewandt. Für diese Arbeit sind die folgenden vier Prinzipien 3, 4, 5 und 7 von Relevanz. Folgend wird aufgezeigt, wie diese für den Aufbau eines Schachblatt Corpus zur Evaluierung von ChessReader umgesetzt werden.

- Prinzip 3 : *Only those components of corpora which have been designed to be independently contrastive should be contrasted.*

  Dieses Prinzip wird angewandt, in dem darauf geachtet wird, dass für bestimmte Eigenschaften Beispiele als auch Gegenbeispiele vorhanden sind. Wird ein Schachblatt mit einer leserlichen Handschrift verwendet, so sollte auch ein Schachblatt, welches unleserlich ist, vorhanden sein.

- Prinzip 4 : *Criteria for determining the structure of a corpus should be small in number, clearly separate from each other, and efficient as a group in delineating a corpus that is representative of the language or variety under examination.*

  Die Scoresheets sollen klar gruppiert werden können. Die Halbzüge auf den Scoresheets werden in der SAN (siehe Kapitel 5.1) vermerkt. Diese variiert von Sprache zu Sprache, weshalb es sinnvoll ist, im Corpus sprachliche Untergruppen zu bilden. Diese Gruppierung ist beim Betrachten der Corpus-Ordnerstruktur ersichtlich. Ausserdem wird darauf geachtet, dass weitere Eigenschaften so definiert werden, dass Scoresheets einfach und eindeutig nach diesen gruppiert werden können. Eine solche Eigenschaft kann zum Beispiel die Schriftfarbe, in der die Halbzüge notiert wurden, sein.

- Prinzip 5 : *Any information about a text other than the alphanumeric string of its words and punctuation should be stored separately from the plain text and merged when required in applications.*

  Dieses Prinzip besagt, dass zusätzliche Informationen separat von den gesammelten Werten abgelegt werden sollen. Argument dafür sei eine einfachere Datenverarbeitung.

  In dieser Arbeit und im Rahmen der Corpus-Überarbeitung, wurde bewusst gegen dieses Prinzip entschieden, da mit dem Zusammenlegen die Zuweisung von Metadaten und Halbzügen intuitiver ist. Auch, dass in einem Programmcode nur eine Datei eingelesen werden muss, spricht für eine Konsolidierung.

- Prinzip 7 : *The design and composition of a corpus should be documented fully with information about the contents and arguments in justification of the decisions taken.*

  Die Struktur des Corpus und dessen zusätzliche Informationen werden im Verlaufe dieser Arbeit thematisiert und erläutert. Diese Arbeit kann als Dokumentation für den überarbeiteten Corpus genutzt werden.

5.5 Evaluierung von ML Systemen

In ihrem in 2017 veröffentlichten Paper [24] beschreiben Breck et al. ein Framework an 28 Tests, um die Production-Readiness eines Machine Learning Produkts zu prüfen. Die Tests sind in die Kategorien *Data*, *Model*, *Infra* und *Monitor* eingeteilt und grundsätzlich für kontinuierlich, online trainierte, supervised Modelle konzipiert. Aus diesem Grund sind viele der beschriebenen Tests nicht direkt auf ChessReader anwendbar. Einige bieten aber trotzdem eine Hilfestellung für den Aufbau des Corpus und in der Strukturierung der Evaluierungsmetriken. Folgend wird auf diese Tests eingegangen, was sie bedeuten und wie sie auf ChessReader angewendet werden können.

- *Data 1: Feature expectations are captured in a schema.* Im Paper wird dies als eine Plausibilitätsprüfung für die verwendete Trainings- und Testdaten beschrieben. Die Repräsentation der Daten für eine Domäne soll überprüfbar sein. Im Falle der Evaluierung von ChessReader kann dies umgesetzt werden, indem die Eigenschaften des verwendeten Corpus im generierten Report beschrieben werden. So soll erkennbar sein, ob ein schlechtes Evaluierungsresultat auf verwendete Inputdaten mit gewissen Eigenschaften zurückzuführen ist.
- *Model 5: A simpler model is not better.* Es sollen regelmässige Vergleiche mit einem einfacheren Model gemacht werden, um den Nutzen eines komplexeren Models im Vergleich sicherzustellen. Dies kann in der Evaluierung von ChessReader direkt aufgezeigt werden, indem die Predictions von ChessReader mit den Predictions einer einzelnen OCR Engine verglichen wird.
- *Model 6: Model quality is suffitient on all important data slices.* Ein Model soll anhand einem Subset der Daten mit einer bestimmten gemeinsamen Eigenschaft validiert werden, um mögliche Schwachstellen erkennen zu können. Dies kann ebenfalls direkt für die ChessReader Evaluierung übernommen werden, setzt aber voraus, dass der verwendete Corpus nach Eigenschaften gefiltert werden kann.
- *Infra 3: The full ML pipeline is integration tested.* Im Paper wird beschrieben, dass die Interaktion der einzelnen Schritte innerhalb der ML Pipeline in einem End-zu-End Test validiert werden soll, um mögliche Fehlerfortpflanzungen erkennen zu können. Für die Evaluierung von ChessReader bedeutet dies, dass die Einflüsse der einzelnen Scoresheet-Verabeitungsschritte auf das Endresultat aufzuzeigen sind.
- *Infra 5: The model allows debugging by observing the step-by-step computation of training or inference on a single example.* Es soll möglich sein, falls ein Model schlecht auf spezifische Inputdaten reagiert, diese einzeln zu testen. Umgesetzt in der Evaluierungspipeline bedeutet dies, dass es möglich sein soll, Scoresheets, für welche eine Evaluierung viel schlechter ist, zu erkennen und gezielt separiert evaluieren zu können.

6 Ausgangslage

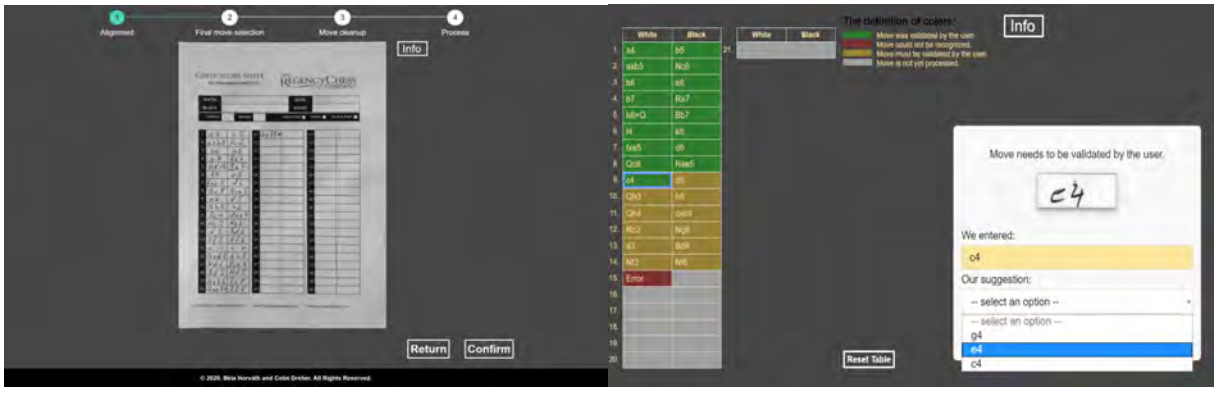
Im folgenden Kapitel wird genauer auf das Produkt ChessReader und den bereits existierenden Scoresheet Corpus v4 eingegangen. Dies dient als Grundlage, um die in Kapitel 7 und Kapitel 8 vorgenommenen Änderungen in Kontext zu setzen.

6.1 ChessReader

ChessReader wurde an der Zürcher Hochschule für Angewandte Wissenschaften (ZHAW) von mehreren, unabhängigen, studentischen Gruppen und Mitarbeitern der SpinningBytes AG entwickelt. Erzielte Fortschritte der ChessReader Software sind über mehreren wissenschaftliche Arbeiten dokumentiert, was es schwierig macht, schnell einen Überblick über die Umsetzung zu erhalten. Deshalb wird hier eine Zusammenfassung der vorherigen Semester-, und Bachelorarbeiten gegeben und die technische Implementation von ChessReader vor Beginn dieses Projekts genauer beschrieben.

6.1.1 Versionsgeschichte

Die Proof of Concept Version von ChessReader [5], damals noch VeryChess genannt, wurde von Dreher und Horvath als monolithische Python Applikation, mithilfe des Web Frameworks Flask[4] aufgebaut. Frontend Visualisierungen wurden mit der Templating Engine von Flask[5] entwickelt. Der Benutzer oder die Benutzerin kann ein Schachblatt hochladen, welches anhand der Ecken des Blattes ausgerichtet wird. Anschliessend werden die Tabellenzellen an ihrer horizontalen und vertikalen Linien erkannt. Auf die erkannten Zellen wird dann die OCR Engine ABBYY angewendet [18], welche für jede Zelle eine Liste an möglichen Zeichenkombinationen zurück gibt. In einem Post-Processing Schritt wird mithilfe eines stochastischem Entscheidungsbaumes die wahrscheinlichste Abfolge von Zügen eruiert. Die daraus resultierende Zugkombination wird dem Benutzer oder der Benutzerin im Frontend angezeigt, wo dann einzelne Züge bestätigt oder korrigiert werden können. Es kann jederzeit das Resultat im PGN Format heruntergeladen werden.



(a) Hochgeladenes Scoresheet (b) Benutzeransicht zur Zugbestätigung

Abbildung 7: Proof of Concept Version VeryChess

[4] palletsprojects.com/p/flask/
[5] flask.palletsprojects.com/en/2.3.x/templating/

Im gleichen Jahr wurde in einer Folgearbeit von Abduli [6] ein zusätzlicher Pre-Processing Schritt implementiert. Dieser erkennt Züge, welche mit horizontalen Tabellenlinien überlappen. Er entfernt die Linien und füllt entstehende Lücken in der Handschrift auf. Dieser Schritt verbesserte die Fähigkeit der ABBYY Engine betroffene Zeichen zu erkennen. Abbildung 8 zeigt einen Zug mit dem überlappenden Buchstaben "g" vor und nach dem Pre-Processing Schritt.

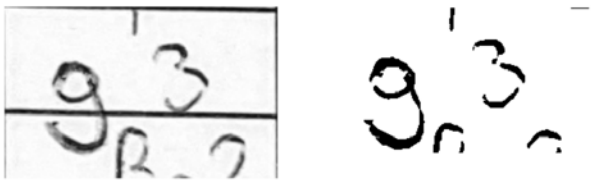


Abbildung 8: Buchstabe mit Überlappung vor dem Pre-Processing Schritt (links) und nach dem Pre-Processing Schritt (rechts) [6].

Die nächste Erweiterung folgte ein Jahr später durch Caglayan und Nielsen [7]. In dieser Bachelorarbeit wurden diverse technische und algorithmische Verbesserungen vorgenommen. Primär wurde ABBYY zu einem OCR Ensemble ergänzt, welches zusätzlich Engines von Google [21], Microsoft [20] und Amazon [19] beinhaltet. Jede OCR Engine generiert Predictions für Halbzüge, welche dann mithilfe einer eigens implementierten Schach-Engine, der "Simple Chess Engine", konsolidiert werden. Die bestehende Methode, mittels des stochastischen Entscheidungsbaumes, wird hiervon abgelöst. Zusätzlich wurde der bisherige Mechanismus zur Erkennung der Tabellenzellen ersetzt. Der originale Mechanismus funktionierte gut für Tabellen mit durchgezogenen Tabellenlinien, war jedoch nicht verwendbar für gestrichelte Tabellen oder Tabellen, welche nicht über perfekte horizontale und vertikale Linien verfügen. Der neue Algorithmus verwendet ebenfalls das OCR Ensemble, welches auf das ganze Schachblatt eine Texterkennung ausführt. Aus den Erkennungen werden die Tabellennummerierungen gefiltert und davon die einzelnen Zellen extrapoliert. Der Linienentfernungs-Pre-Processing Schritt wurde, basierend auf unzureichender Implementation wieder entfernt, da dieser Bildartefakte generiert, welche die Texterkennung verschlechterte.

Ambrosini und Hellinger [8] fokussierten sich in ihrer Arbeit primär auf die grafische Benutzeroberfläche, Multiuserfähigkeit und die Möglichkeit die Applikation zu veröffentlichen. Zudem unternahmen sie den Namenswechsel zu ChessReader. Das GUI wurde visuell mit dem CSS Framework Bulma[6] überarbeitet und durch ein animiertes Schachbrett ergänzt. Zudem wurde die Applikation mithilfe von Docker[7] containerisiert.

Boner und Hostettler [9] überarbeiteten in ihrer Projektarbeit die Software Architektur von ChessReader. Front- und Backend wurden voneinander getrennt und das Frontend wurde mit dem Web Framework Angular[8] von Grund auf neu gebaut. In ihrer Bachelorarbeit [10] wurde das Backend überarbeitet, sodass dieses REST-API [25] Endpoints zur Verfügung stellt. Zusätzlich wurde auch ein Schachblatt Corpus aufgebaut, mit dem Gedanken, Schachapplikationen wie ChessReader evaluieren zu können. Der Corpus, welche einfach "Corpus v4" genannt wird, beinhaltet zu diesem Zeitpunkt 62 Spiele mit etwas 4000 Halbzügen.

Zuletzt wurden im Sommer 2023 von einem Mitarbeiter der SpinningBytes AG[9] Verbesserungen am Produktiv-Deployment mit Docker gemacht. Datenbankdaten wurden persistiert, die Webseite hinter einen Reverse Proxy mit Verschlüsselung positioniert und die Trennung von

[6] bulma.io
[7] docker.com
[8] angular.dev
[9] spinningbytes.com

Frontend und Backend verbessert. Zudem wurde das Frontend weiter ausgearbeitet [26].

6.1.2 Ablauf der Scoresheet Verabeitung

Die aktuelle Implementation von ChessReader basiert primär auf der algorithmischen Arbeit von Caglayan und Nielsen [7], der Software-Architektur von Boner und Hostettler [10] und dem überarbeiteten Frontend von Muletta [26]. Abbildung 9 zeigt eine Übersicht des Scoresheet Verarbeitungsprozesses. Die Schritte, welche im Backend automatisch ausgeführt werden, sind in Schwarz dargestellt und Benutzerinteraktionen in Blau. Folgend wird auf diesen Ablauf genauer eingegangen. Erforderte Benutzereingaben werden zusätzlich durch Bildschirmaufnahmen unterstützt.

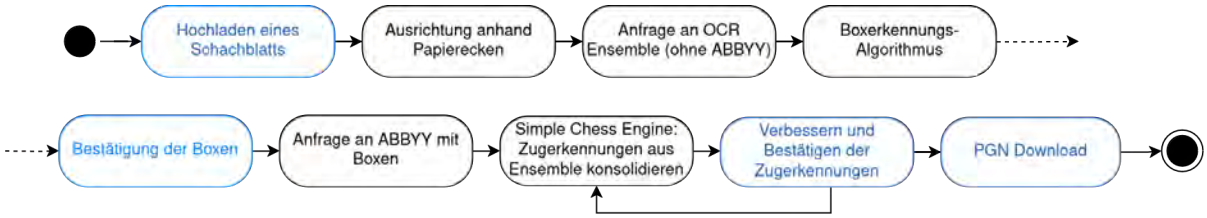


Abbildung 9: Ablauf Diagramm des ChessReader Scoresheet Verarbeitungsprozesses

Hochladen eines Schachblatts und Ausrichtung anhand Papierecken

Der Benutzer oder die Benutzerin lädt ein Schachblatt auf ChessReader hoch. Das Bild wird als Original in der Datenbank gespeichert. Zusätzlich wird es anhand der Papierecken ausgerichtet und ebenfalls abgespeichert.

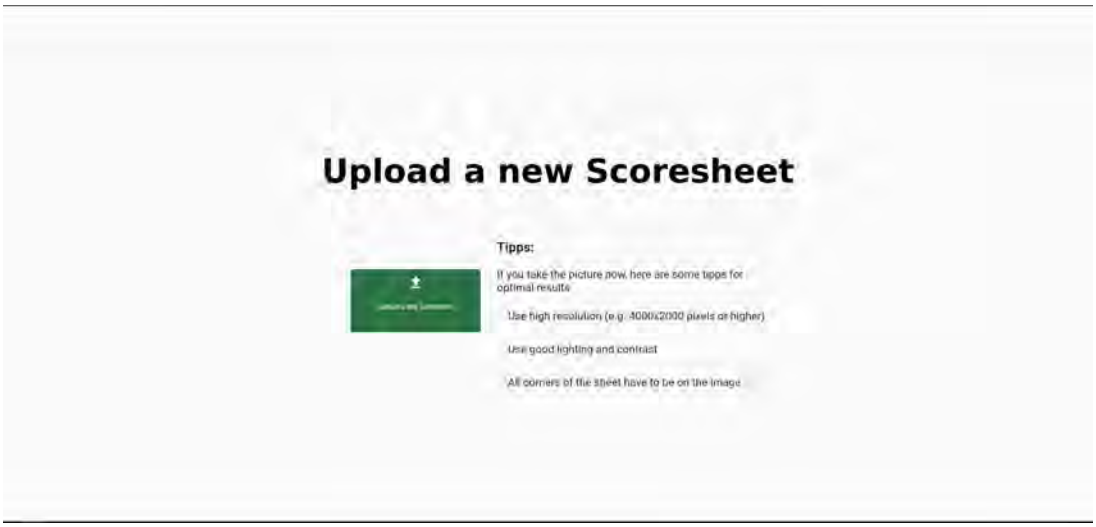


Abbildung 10: Der Benutzer oder die Benutzerin lädt ein Scoresheet mithilfe des grünen Knopfs auf ChessReader hoch

Anfrage an OCR Ensemble und Boxerkennungsalgorithmus

Im selben Schritt wird sogleich die Erkennung der Boxen gestartet. Hierfür wird eine Texterkennung durch die OCR Engines von Google, Microsoft und Amazon durchgeführt. Der erkannte Text wird nach den Zellennummerierungen gefiltert und die zugehörigen Zellen werden daraus extrapoliert. Die erhaltenen Boxen werden in der Datenbank gespeichert.

Bestätigung der Boxen

Dem Benutzer oder der Benutzerin werden die erkannten Boxen nun im Frontend angezeigt. Dieser soll die letzte Box markieren, welche einen Halbzug beinhaltet. Alle Boxen welche der Benutzer oder die Benutzerin markiert hat, werden wieder dem Backend gesendet und die nicht benötigten Boxen werden aus der Datenbank gelöscht.

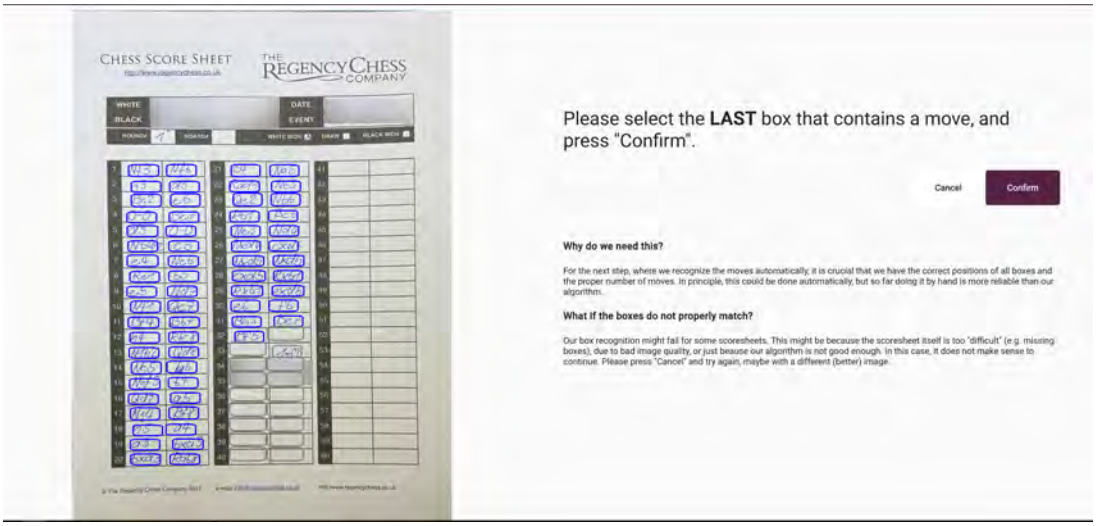
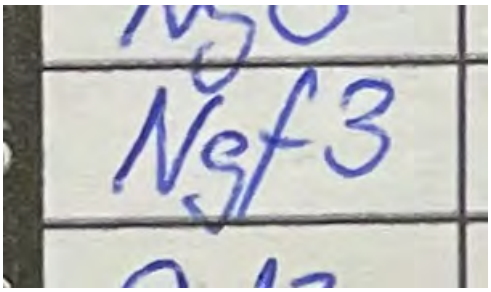


Abbildung 11: Der Benutzer oder die Benutzerin markiert die letzte Box, welche erkannt werden soll

Anfrage an Abbyy und Simple Chess Engine

Anschliessend wird die Zugerkennung gestartet. Für die OCR Engines von Google, Microsoft und Amazon werden die zuvor erhaltenen Resultate, den erkannten Boxen zugewiesen, um eine Prediction für die entsprechende Box zu erhalten. Zusätzlich wird die ABBYY OCR Engine auf die Bereiche der Boxen angewendet. Auf die Predictions aller OCR Engines werden in der Simple Chess Engine im sogennanten "Confusion Modul" mögliche Zeichenverwechslungen berechnet wie "a" zu "d" oder "b" zu "6". Die berechneten, möglichen Verwechslungen werden als Input für das "Confidence Modul" verwendet. Im Confidence Modul wird pro OCR Engine anhand der möglichen Verwechslungen überprüft, wie visuell ähnlich die Erkennung einer OCR Engine zu einer gültigen SAN Notation ist. Daraus entsteht für jeden erkannten Halbzug jeder Engine ein Confidence Wert zwischen null und eins. Um aus den Engine Antworten und deren Confidence-Werten ein Resultat zu erhalten, werden die Confidence-Werte der Engines pro Halbzug-Kandidat miteinander multipliziert. Der Halbzug-Kandidat mit der höchsten Gesamt-Confidence wird weiterverwendet.



(a) Handgeschriebener Schachzug

```
1 'Ngf3', 0.6961012348727155
2 'Nh3', 0.015662277784636098
3 'Ne4', 0.015662277784636098
4 'Nxe6', 0.0023493416676954146
5 'Nxf7', 0.015662277784636098
6 'Nh7', 0.0023493416676954146
```

(b) Zugehörige Confidencewerte für verschiedene Möglichkeiten

Abbildung 12: Confidence Modul Ausgabe [7]

Der Corrector-Algorithmus der Simple Chess Engine nimmt die Liste der erkannten Züge und versucht das Schachspiel nachzuspielen. Hierfür wird für jeden Spielzustand das Confidence-Modul mit den zugehörigen Halbzug-Kandidaten ausgeführt und der Halbzug mit der höchsten Confidence verwendet. Dies wird für jeden Spielzustand rekursiv durchgeführt, bis das Spiel endet, oder der letzte zu erkennende Halbzug erreicht ist.

Verbessern und Bestätigen der Zugkorrektur

Die resultierende Liste an Halbzügen wird in der Datenbank zusammen mit den Antworten der Engines abgespeichert und dem Benutzer oder der Benutzerin im Frontend angezeigt. Der Benutzer oder die Benutzerin muss nun die Züge, welche eine Confidence unter einem bestimmten Schwellwert (0.9 in der aktuellen Implementierung von ChessReader) von Hand korrigieren. Züge mit einer Confidence von über 0.9 können übersprungen werden. Nimmt der Benutzer oder die Benutzerin eine Verbesserung eines Zuges vor, wird im Hintergrund der Corrector-Algorithmus ab diesem Punkt erneut ausgeführt, um einen neuen validen Pfad durch den Spielbaum zu finden. Bestätigte Züge werden in der Datenbank als "valid" markiert.

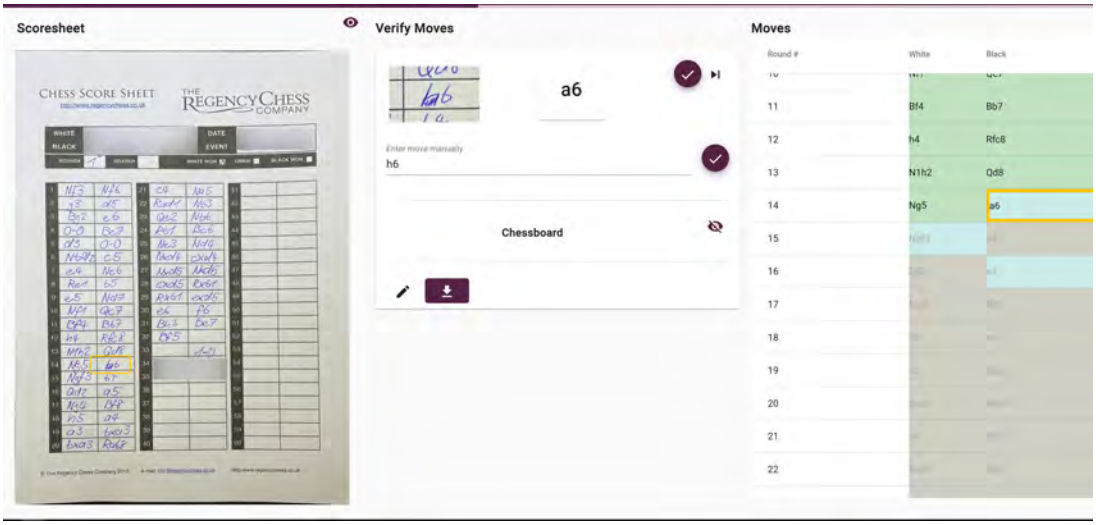


Abbildung 13: Der Benutzer oder die Benutzerin bestätigt oder korrigiert die hellblau markierten Züge von Hand, alle anderen können übersprungen werden

PGN Download

Der Benutzer oder die Benutzerin kann zu jeder Zeit eine Textdatei im PGN Format herunterladen.

```
[Event "ZHAW Chess Championships"]
[Site "Technikumstrasse 9, 8401 Winterthur"]
[Date "2024-01-06"]
[Round "1"]
[White "Nadine Moser"]
[Black "Robin Meier"]
[Result "1-0"]

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
```

Abbildung 14: PGN Outputformat nach einer vollständigen Digitalisierung eines Scoresheets

6.1.3 REST-API Endpunkte

Die REST-API Endpunkte stellen die Schnittstelle zwischen Front- und Backend dar. Alle Aktionen im Frontend lösen eine Reaktion im Backend aus. Die, für die Verarbeitung von Scoresheets, wichtigsten Endpunkte sehen folgendermassen aus:

- `POST http://chessreader:5000/api/matches` wird verwendet, um ein neues Scoresheet hochzuladen und führt die Ausrichtung des Scoresheets anhand der Ecken aus. Dieser Endpoint startet auch den Boxerkennungsalgorithmus. Nach erfolgreichem Hochladen wird eine `match_id` für das Scoresheet zurückgegeben.
- `GET http://chessreader:5000/api/matches/<match_id>/boxes` gibt eine Liste von gespeicherten Boxen für eine `match_id` zurück.
- `PUT http://chessreader:5000/api/matches/<match_id>/boxes` entfernt alle Boxen für eine gegebene `match_id`, welche nicht zu den, im Body mitgegebenen, Boxen gehören.
- `GET http://chessreader:5000/api/matches/<match_id>/scan` führt die Zugerkennung für eine gegebene `match_id` aus und gibt eine Liste der erkannten Halbzüge zurück.
- `POST http://chessreader:5000/api/matches/<match_id>/play` wird verwendet, um im Frontend einzelne Halbzüge zu bestätigen. Falls der Halbzug korrigiert wurde, wird der Corrector-Algorithmus der Simple Chess Engine erneut aufgerufen.
- `POST http://chessreader:5000/api/matches/<match_id>/playmoves` funktioniert identisch wie der `play` Endpunkt, aber für eine Liste an Halbzügen, anstelle eines einzelnen Halbzuges.

6.2 Aufbau Corpus v4

Im Juni 2023 setzten Maurice Hofstettler und Lukas Boner den Fokus ihrer Bachelorarbeit auf den Aufbau eines Corpus an Schachblättern [10], um eine Grundlage für das Testen und Evaluieren von ChessReader zu schaffen. Das Endprodukt beinhaltet Datensätze von verschiedene Schachblättern und nennt sich Corpus v4.

6.2.1 Struktur des Corpus v4

Die Struktur des Corpus v4 wurde so gewählt, dass neue Sprachen und Scoresheets einfach hinzugefügt werden können. Annotierte Schachblätter sind in Englisch, Deutsch und Französisch vorhanden. Unter den einzelnen Sprachordnern befinden sich die dazugehörigen, annotierten Schachspiele. Die einzelnen Spiele sind dabei wieder in Ordner abgelegt, welche alle Informationen zu den notierten Halbzügen und Metadaten beinhalten.

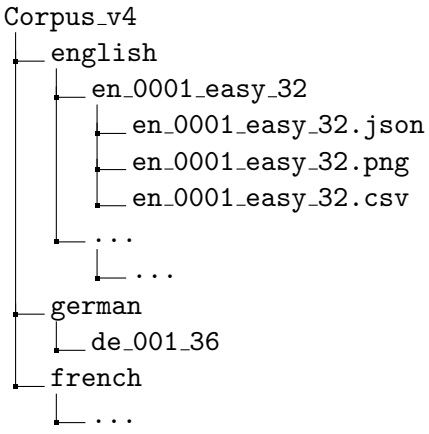


Abbildung 15: Ordnerstruktur des Corpus v4

6.2.2 Zug Anmerkungen

Die Informationen zu einem Zug wurden in einer CSV Datei abgelegt. Der Aufbau dieser Datei wurde dabei stark an ein Schachblatt angelehnt. Eine Linie im CSV enthält zwei Halbzüge und einige zusätzliche Informationen.

	A	B	C	D	E	F	G	H
1	corrected move white	uncorrected move white	is hard to read	has notation error	corrected move black	uncorrected move black	is hard to read	has notation error
2	e4		0	0	d6		0	0
3	d4		0	0	Nf6		0	0
4	Nc3		0	0	g6		1	0
5	Be3		0	0	a6		0	0
6	a4		1	0	Bg7		0	0

Abbildung 16: Ausschnitt aus en_0031_easy_33.csv

Für die beiden Spielenden "white" und "black" werden die Halbzüge notiert. Das Feld "uncorrected move" entspricht exakt dem notierten Text auf dem Scoresheet. Im "corrected move" Feld wird ein Zug in korrekter SAN notiert, falls das Original einen Fehler beinhaltet. Ist das "uncorrected" Feld leer, so ist der Wert derselbe wie im "corrected" Feld.

"has notation error" wird auf 1 gesetzt, wenn die Notierung auf dem Scoresheet nicht der SAN entspricht. Dies kann zum Beispiel eintreffen, wenn eine Kurzform notiert wird, die inoffiziell aber dennoch eindeutig ist.

Ob der Halbzug einfach oder schwer zu lesen ist, wird im "is hard to read" Feld festgehalten. Ist der Wert auf 1 so gilt der notierte Halbzug als schwer leserlich. Der Wert 0 verspricht einen einfach lesbaren Halbzug. Ob ein Halbzug als schwer leserlich markiert wird, hängt von verschiedenen Faktoren ab, ein Beispiel wäre unlesbare Handschrift. Die Einschätzung liegt dabei beim Annotierenden. Weitere Faktoren sind das Überschneiden von Buchstaben und das Überschreiten der Box.

Im CSV Ausschnitt in Abbildung 16 ist der Halbzug "g6" als schwer zu lesen eingestuft worden. Dies liegt daran, dass das "g" über die Box hinaus ragt. Wird nur der Boxinhalt analysiert, kann das "g" mit einem "a" verwechselt werden.

Die Überschreitung der Box sieht folgendermassen aus:

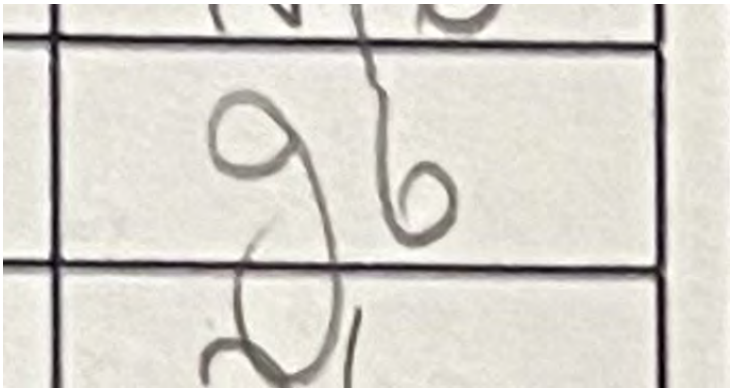


Abbildung 17: Beispiel für ein schwer zu lesender Halbzug

6.2.3 Metadaten

Die Metadaten werden in einer separaten JSON Datei abgespeichert. Hier werden verschiedene Zusatzinformationen zum Scoresheet abgelegt.

- `file_name` beschreibt das dazugehörige CSV mit den Zuginformationen.
- `game_length` sagt aus wie viele Züge im Spiel getätigt wurden. Dies entspricht der Anzahl Reihen im CSV.
- `won` beschreibt den Gewinner und ist entweder "white" oder "black". Ist es ein Untentschieden wird "draw" vermerkt.
- `language` ist der ISO-Code[10] der verwendeten Sprache, um die Züge aufzuschreiben.

[10] https://www.loc.gov/standards/iso639-2/php/code_list.php

- `difficulty` beschreibt die Schwierigkeit des gesamten Scoresheets. Bewertet werden die folgenden sechs Eigenschaften: Tiefe Helligkeit oder tiefer Kontrast, dunkle Schatten auf einem wichtigen Teil des Blattes, Verzerrte Prespektive, mehr als 33 % unleserliche Halbzüge und gepunktete oder gestrichelte Boxränder. Trifft keine der Eigenschaften zu, so ist das Scoresheet als "easy" klassiert, trifft genau eine Eigenschaft zu als "medium" und wenn mehrere zutreffen, wird es als "hard" markiert.
- `keywords` enthalten Eigenschaften des Scoresheets. Trifft eine der Eigenschaften zu, welche das Scoresheet als "medium" oder als "hard" klassifizieren lässt, so sind die entsprechenden Eigenschaften hier beschrieben.
- `box_postitions` ist eine Liste mit allen Boxen auf dem Scoresheet. Eine Box besteht aus ihrem Index, ihrer Breite und Höhe und den X- und Y-Koordinaten in Pixel, welche aussagen, wo der Mittelpunkt der Box liegt. Ob die Boxen einen Halbzug enthalten, ist nicht relevant.
- `table_position` ist eine Liste von Positionen, wo sich die Tabellen auf dem Scoresheet befinden.
- `table_size` beschreibt die Breite und Höhe der Tabelle in Pixel.

Folglich ein Beispiel mit den beschriebenen Informationen zum Scoresheet en_0001_easy_32.png. Für die Übersichtlichkeit wurden nicht alle Box-Positionen dargestellt.

```
{
    "file_name": "en_0001_easy_32.csv",
    "game_length": 32,
    "won": "white",
    "language": "en",
    "difficulty": "easy",
    "keywords": [],
    "box_positions": [
        [
            {
                "box_index": 0,
                "height": 76,
                "pos_x": 358,
                "pos_y": 980,
                "width": 196
            },
            ...
        ]
    ],
    "table_position": [
        {
            "x": 352,
            "y": 943
        }
    ],
    "table_size": [
        {
            "height": 1961,
            "width": 1656
        }
    ]
}
```

Listing 1: Ausschnitt aus Metadaten JSON Datei von en_0001_easy_32.json

7 Überarbeitung von ChessReader

Die von ChessReader zur Verfügung gestellten REST-API Endpunkte wurden überarbeitet, damit die Evaluierungspipeline automatisiert Scoresheets hochladen kann. Während der Bereitstellung der Schnittstellen wurden zudem weitere Anpassungen vorgenommen. Im folgenden Kapitel wird die Überarbeitung der Schnittstellen aufgezeigt. Die zusätzlichen Änderungen werden unter dem Kapitel 7.2 aufgelistet, in der Hoffnung, dass zukünftige Entwickler und Entwicklerinnen die Qualität und Dokumentation des Codes weiterführen und weiter verbessern werden.

7.1 Erfüllen der technischen Voraussetzungen

Bei der Überarbeitung der REST-API Endpunkte handelt es sich in den meisten Fällen um eine Verbesserung im logischen Ablauf. Folglich werden die wichtigsten Schnittstellen und deren Änderungen aufgelistet und erläutert. Wie die wichtigsten Endpunkt vor der Überarbeitung aufgebaut waren, ist im Kapitel 6.1.3 beschrieben. Eine detaillierte Übersicht aller überarbeiteten Endpoints ist im Anhang C ersichtlich.

- `POST http://chessreader:5000/api/matches`

  Dieser Endpoint wurde unterteilt. Vor der Überarbeitung wurde bei dieser Schnittstelle zuerst ein Schachspiel erstellt und anschliessend gleich die Boxerkennung gestartet. Die Boxerkennung wurde aus diesem Endpoint herausgelöst und in die Schnittstelle `GET http://chessreader:5000/api/matches/<match_id>/boxes` verlegt. Dies führt zu einer klaren Trennung der Funktionalitäten und unterstützt eine übersichtliche Strukturierung der Evaluierungspipeline.

- `PUT http://chessreader:5000/api/matches/<match_id>/boxes`

  Vor der Überarbeitung mussten alle Boxen, welche nicht gelöscht werden sollten, mit all ihren Daten, wie ihr `boxIndex` und ihre Koordinaten, mitgesendet werden. Dieser Ablauf ist fehleranfällig, da so Boxen dazwischen manipuliert werden könnten. Auch für eine Automatisierung ist es eine grosse Erleichterung, nicht alle Boxen bereitstellen zu müssen. Daher wurde dieser Endpoint umgeschrieben, dass nur ein `boxIndex` mitgesendet werden muss, um die letzte Box anzuwählen. Alle nachfolgenden Boxen werden aus der Datenbank entfernt.

- `GET http://chessreader:5000/api/matches/<match_id>/moves`

  `GET http://chessreader:5000/api/matches/<match_id>/scan`

  Vor der Überarbeitung wurde beim Aufruf des `/scan` Endpoints die Zugerkennung für das Scoresheet durchgeführt. Durch diese Implementierung kam es vor, dass die Erkennung vermehrt über dasselbe Scoresheet laufen gelassen wurden.

  Mit dem Refactoring wurde die Logik im `/moves` Endpunkt überarbeitet, sodass zuerst eine Datenbankabfrage gemacht wird, und mögliche Predictions lädt. Sind bereits Vorhersagen vorhanden, so werden diese gesendet. Sind noch keine erstellt worden, so wird die Vorhersage gestartet und anschliessend das Resultat zurück gegeben.

7.2 Weitere Änderungen

Neben dem überabreiteten Code, welcher einen direkten Zusammenhang mit der Evaluierungspipeline hat, wurden während dem Refactoring weitere Punkte verbessert. Es folgt eine Auflistung der erarbeiteten Änderungen.

- *Service Auslagerung* Um Redundanzen zu vermeiden und eine begünstigte Struktur für Testing zu schaffen, wurde ein Grossteil der Programmlogik in Services ausgelagert. Hierbei wurde sich am Design Prinzip Separation of Concern [27] orientiert.
- *Namensschema* Variablen wurden nach unterschiedlichen Schemen benannt. Hier wurde sich an Standards orientiert und Variablen entsprechend umbenannt. Variablen im Python-Codeteil wurden nach dem PEP 8 Standard [28] zu "snake_case" umbenannt. Für Attribute im JSON Format wird der Google JSON Style Guide verwendet [15]. Dieser besagt, dass Attribute im "camelCase" geschrieben werden sollen.
- *Dependency Aktualisierungen* Veraltete Abhängigkeiten wurden aktualisiert, um allfällige Sicherheitslücken zu schliessen. Pakete, welche nicht mehr verwendet werden, wurden gelöscht und aus der `requirements.txt` Datei entfernt.
- *Swagger* Um die Endpoints des Backends zu dokumentieren, wurde Swagger eingesetzt. Dies macht die einzelnen Endpoints übersichtlicher und ermöglicht, dass diese direkt interaktiv ausprobiert werden können (Siehe Anhang C).
- *Semantic Versioning* ChessReader ist im Verlaufe der Zeit gewachsen und viele Änderungen wurden vorgenommen. Zudem besteht die gesamte Applikation aus mehreren Teilen: Das Frontend, das Backend und mit dieser Arbeit, die Evaluierungspipeline. Um die Kompatibilität der Komponenten sicherzustellen, wurde die Versionierung anhand des Semantic Versionings eingeführt. Somit ist einfach dokumentiert, welche Versionen miteinander kompatibel sind.
- *Datenbank Überarbeitung* Bei der Datenbanken wurde eine einheitliche Struktur in den Spaltennamen umgesetzt. Eine Tabelle und diverse Spalten, die nicht verwendet wurden, sind entfernt worden. Fremdschlüssel, die auf dieselbe Tabelle verwiesen, waren teilweise unterschiedlich benannt. Auch dies wurde vereinheitlicht. Ein ERD ist im Anhang D zu finden.
- *CNN OCR Engine* Basierend auf der Implementierung eines Semester Projekts von Amaury et al. [29] durch Cieliebak [30] wurde eine zusätzliche OCR Engine in der Form eines Convolutional Neural Network (CNN) dem OCR Ensemble hinzugefügt. Die Implementierung wurde mit Pytorch[11] entwickelt und im ONNX Format gespeichert[12]. Die OCR Engine nimmt als Input nicht ein ganzes Scoresheet, sondern einzelne, ausgeschnittenen Bilder der zuvor erkannten Boxen, um eine Erkennung darauf auszuführen.
- *Evaluation Modus* Da bei einer Evaluierung sehr viele Scoresheets auf einmal im ChessReader hochgeladen werden können, wurde mithilfe der Python Library `load_dotenv`[13] ein Evaluierungsmodus hinzugefügt. Dieser ermöglicht, dass ChessReader zu einer separaten Datenbank, wie der regulären Development Datenbank verbindet (Siehe Anhang B). Der Modus kann mit `python app.py evaluate` gestartet werden.

[11] pytorch.org
[12] onnx.ai
[13] pypi.org/project/python-dotenv/

8 Überarbeitung des Chess-Scoresheet Corpus

Für eine aussagekräftige Evaluation von ChessReader ist eine gute Datengrundlage essenziell. In diesem Sinne wurde in dieser Arbeit Zeit in die Überarbeitung des Corpus v4 [10] investiert und der "Chess-Scoresheet Corpus" erstellt. Die Daten wurden dabei aus dem Corpus v4 entnommen. Folglich wird auf das Verbesserungspotenzial des Corpus v4, die Anforderungen an den überarbeiteten Corpus und dessen Umsetzung eingegangen.

8.1 Verbesserungspotenzial des Corpus v4

Der Corpus v4 bietet bereits eine gute Grundlage, ist allerdings noch ausbaufähig. Es folgt eine Auflistung von Punkten mit Verbesserungspotenzial. Auf die Bedeutung der Eigenschaften wird in Kapitel 6.2 eingegangen.

- Metadaten und Informationen zu den Zügen sind in zwei unterschiedlichen Dateien mit unterschiedlichem Dateiformat abgelegt. Dadurch muss beim Einlesen in einem Programm zwei verschiedene Vorangehensweisen entwickelt werden. Der Code wird dadurch komplizierter, als wenn nur eine Datei mit allen Informationen genutzt wird.
- Im CSV wird die Eigenschaft "is hard to read" abgelegt. Dieser Wert hat viele Einflussfaktoren und wie stark diese gewichtet werden, entscheidet die annotierende Person. Dies führt zu einer sehr subjektiven Bewertung, was inkonsistente Einschätzungen zur Folge hat.
- Die Informationen wie schwer ein Scoresheet zu lesen ist, setzt sich aus unterschiedlichen Eigenschaften zusammen. Ein Scoresheet kann als schwer leserlich eingestuft werden, weil es Schatten aufweist, während ein anderes auf Grund von verzerrter Perspektive so eingestuft wird. Ein wiederum anderes Scoresheet kann wegen einer unsauberen Handschirft als schwer eingestuft werden. Äusserungen, wie gut solche Scoresheets eingelesen werden können, können aus dieser Eigenschaft kaum getroffen werden, da diese viel zu unpräzise sind.
- Die "keywords": Einzelne Keywords sind wichtig für die Evaluierung, allerdings, kann das Format besser gewählt werden. Beim Hinzufügen von Eigenschaften in einer Liste, können Tippfehler und unterschiedliche Bezeichnungen eine Herausforderung werden.
- Es sind keine Informationen über die spezifischen Eigenschaften zu einzelnen Boxen vorhanden. Einzig die Keywords geben einene kleinen Einblick, lassen sich allerdings nicht einer einzelnen Box zuweisen. Eine Aussage über die Schwierigkeit beschreibt immer das ganze Scoresheet. Grundsätzlich wäre es aber möglich, dass nur eine Box schwer zu lesen ist, während andere Boxen gut erkennbar sind.
- Die Boxpositionen von allen Boxen werden im JSON abgespeichert, egal ob sie einen Halbzug beinhalten oder nicht. Für die Evaluierung sind nur die beschriebenen Boxen relevant. Die überschüssigen Boxinformationen blähen das JSON unnötig auf und machen es unleserlich.

8.2 Anforderungen

Der Corpus soll gut strukturiert und von Menschenauge wie auch für Programmiersprachen einfach zu lesen sein. Zudem sollen die wichtigsten Eigenschaften der Scoresheets und insbesondere der Handschrift festgehalten werden, um so mögliche Zusammenhänge zwischen den Eigenschaften und fehlerhaften Vorhersagen von ChessReader feststellen zu können. Für hinzukommende Scoresheets und Eigenschaften soll der Corpus einfach erweiterbar sein, gleichzeitig aber nicht unnötig aufgebläht werden. Durch klare Berwertungsvorgaben, soll es möglich sein, Scoresheets von verschiedenen Erfassern annotieren zu lassen und dennoch auf gleiche Resultate zu kommen.

8.3 Umsetzung

Im Designentscheid des Chess-Scoresheet Corpus kristallisierten sich drei zentrale Themengebiete heraus. Die drei Themen sind die Dateistruktur, die Namensgebung und der Aufbau der Annotationen. Auf jeden dieser Teile wird in den folgenden Unterkapitel eingegangen und die wichtigsten Entscheidungen festgehalten.

8.3.1 Dateistruktur

Für jedes Scoresheet wurde ein Ordner erstellt. Im jeweiligen Ordner ist eine Datei im JSON-Format abgelegt. Dieses Format wurde aufgrund der Lesbarkeit für Mensch und Computer gewählt.

Wie ein solches JSON aufgebaut ist und welche Eigenschaften nach welchen Bedingungen gesetzt werden, wird im Kapitel 8.3.3 erläutert.

Die von Hand ausgefüllten Scoresheets werden digitalisiert und ebenfalls im Ordner abgelegt. Die Digitalisierung kann dabei mittels einer Kamera oder eines Scanners geschehen. Im Corpus gibt es gescannte, wie auch abfotografierte Dokumente.

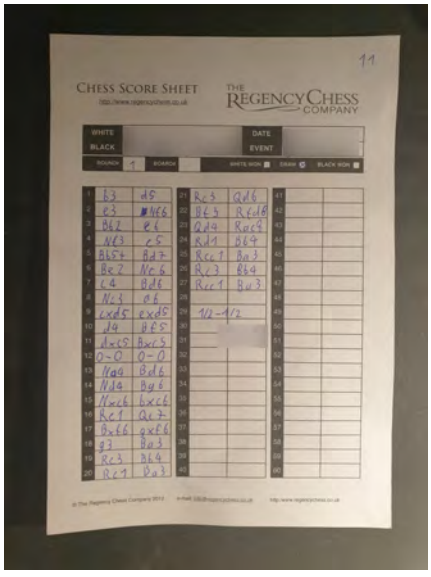


Abbildung 18: Scoresheet en_011_27.png wurde fotografiert

Abbildung 19: Scoresheet de_006_36.png wurde gescannt

Die Züge auf den Scoresheets werden in SAN geschrieben. Diese Notation variiert leicht von Sprache zu Sprache. Bis Anhin unterstützt ChessReader nur die englische SAN. Demnach wurde die Struktur so aufgebaut, dass die Sprachen abgetrennt sind. Sprachordner wurden erstellt und die entsprechenden Scoresheet-Ordner darunter abgelegt. Dies entspricht dem vierten Prinzip von Sinclair (siehe Kapitel 5.4).

Folglich wird eine Übersicht der Struktur des Chess-Scoresheet Corpus aufgezeigt.

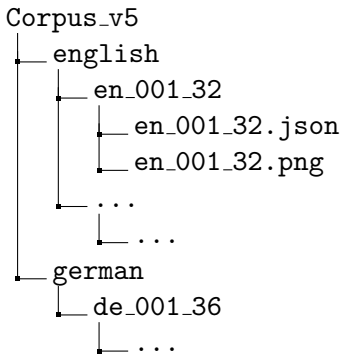


Abbildung 20: Ordnerstruktur des Chess-Scoresheet Corpus

8.3.2 Namensgebung

Der Ordner, die Annotations-Datei und das Bild sind jeweils unter dem selben Namen abgelegt, lediglich die Endung unterscheidet sie. Ein Dateiname setzt sich aus den beiden Eigenschaften `language` im ISO 639-1 Standard [31], `game_length` und einem `index` zusammen. Die Sprache und die Spiellänge werden in einem späteren Kapitel 8.3.3 erläutert. Der Index wird nur im Namen verwendet. Beim Erfassen eines neuen Scoresheets muss lediglich darauf geachtet werden, dass der Index noch nicht verwendet wurde. Vorzugsweise wird der höchste bestehende Index innerhalb des Sprachordners, um eins hochgezählt und diese Zahl als neuen Index verwendet. Durch den Index ist es einfacher ein bestimmtes Scoresheet zu finden, zudem ist es so möglich zwei unterschiedliche Scoresheet in der selben Sprache und der gleichen Länge abzulegen. Eine ähnliche Namensgebung wurde bereits im Corpus v4 genutzt, und dient als Inspirationsgrundlage für die Namensgebung im Chess-Scoresheet Corpus.

Aufbau des Dateinamen: [`language`]_[`index`]_[`game_length`].[Dateiformat]

Beispiel Dateiname: `en_011_27.png`

8.3.3 Aufbau der Annotations-Datei

Eine einzelne Annotations-Datei im JSON Format enthält die Metadaten zum Scoresheet und Informationen über die Halbzug-Boxen. Hier wurde bewusst gegen das fünfte Prinzip von Sinclair entschieden, da eine konsolidierte Datei, eine einfachere Handhabung und Einlesung verspricht.

Ein Beispiel für eine Annotations-Datei ist in der unten abgebildeten Auflistung ersichtlich. Im Anschluss wird genauer auf die Struktur des JSONs eingegangen und die Bedeutung der verschiedenen Attribute erläutert.

```
{
    "game_length": 19,
    "winner": "draw",
    "language": "en",
    "handwriting_color": "blue",
    "table_positions": [
        {
            "height": 1334,
            "x": 379,
            "y": 639,
            "width": 1059
        }
    ],
    "sheet_properties": {
        "is_perspective_asymmetric": true
    },
    "ply_boxes": [
        {
            "corrected_ply": "e4",
            "original_ply": "e4",
            "box_position": {
                "height": 34,
                "x": 383,
                "y": 667,
                "width": 185
            },
            "handwriting_properties": {
                "legibility_level": 3
            }
        }, ...
    ]
}
```


Listing 2: Beispiel eines annotierten JSONs en_050_19.json

Metadaten

Bei den Metadaten werden Informationen, die auf das gesamte Bild des Scoresheets zutreffen, festgehalten. Der unten abgebildete JSON-Ausschnitt zeigt ein Beispiel der Metadaten. Deren Bedeutung wird im Anschluss genauer erläutert.


```
{
    "game_length": 40,
    "winner": "white",
    "language": "en",
    "handwriting_color": "black",
    "table_positions": [
        {
            "height": 1147,
            "x": 794,
            "y": 740,
            "width": 895
        }
    ], ...
}
```


Listing 3: Ausschnitt der Metadaten aus en_043_40.json

`game_length` beschreibt die Anzahl Züge.

`winner` beschreibt den Gewinner, entweder "black" oder "white". Bei einem Unentschieden wird "draw" festgehalten.

`language` enthält die Information in welcher Sprache die Schachzüge notiert wurden. Dafür wird der ISO 639-1 Standard verwendet.

`handwriting_color` beschreibt die Schriftfarbe der notierten Spielzüge.

`table_positions` ist eine Liste von Angaben zu Tabellenpositionen. Schachblätter sind nicht normiert und unterscheiden sich in der Strukturierung. Um mehrere Tabellen abspeichern zu können, wird hier eine Liste verwendet. Die Informationen beschreiben die Höhe und Breite der Tabelle sowie der Mittelpunkt davon mittels einer X- und Y-Koordinate in Pixel.

ChessReader v2.0.0 ist in der Lage Schachspiele zu digitalisieren, die auf einem einseitigen Scoresheet festgehalten wurden. Spiele, welche sich über mehrere Seiten ziehen, werden nicht unterstützt. Dementsprechend wurde der Chess-Scoresheet Corpus für einblättrige Scoresheets ausgelegt.

Die `sheet_properties` werden erstellt, um bestimmte Merkmale eines Bildes des Scoresheets festzuhalten. Dabei handelt es sich um Eigenschaften die entweder zutreffen oder nicht. Zutreffende können festgehalten werden, indem die entsprechenden Attribute explizit auf `true` gesetzt werden. Um den Corpus möglichst kurz zu halten, werden Eigenschaften die nicht zutreffend sind, weggelassen. Dadurch lassen sich Scoresheets mit und ohne bestimmten Eigenschaften festhalten, wodurch das dritte Prinzip von Sinclair befolgt wird. Der Standardwert jeder Eigenschaften ist `false`.

In der unten aufgezeigten Auflistung ist ein Ausschnitt eines ausgefüllten JSONs zu sehen. Das Scoresheet weist dabei einzig die Eigenschaft `low_contrast` auf.

```
{
    ...
    "sheet_properties": {
        "is_low_contrast": true
    }
    ...
}
```

Listing 4: Ausschnitt von Metadaten aus en_043_40.json

Die Struktur lässt es zu, dass neue Scoresheet-Eigenschaften unter den `sheet_properties` hinzugefügt und somit die Beschreibung der Eigenschaften erweitert und verfeinert werden kann.

Folglich werden verwendete Eigenschaften und wie sie einzuschätzen sind, aufgelistet.

- `is_low_contrast`

  Um die Qualität einer Aufnahme festzuhalten, wird `is_low_contrast` verwendet. Dabei soll beim Erfassen von Auge eingeschätzt werden, wie gut die Aufnahme ist. Tiefer Kontrast, eine verdunkelte Aufnahme oder verschwommene Zeichen sind Gründe, für das Setzen dieser Eigenschaft. Das Verwenden von bereits existierenden Algorithmen, zum Setzen eines Kontrastwertes, wurden verworfen, da dabei auch zusätzliche Faktoren wie der Untergrund, auf welchem das Scoresheet fotografiert wurde, ausgewertet werden.

- `is_sheet_incomplete`

  Dieses Merkmal beschreibt, ob das Blatt unvollständig ist oder nicht.

- `has_shadows`

  Beschreibt, ob Schatten Halbzug-Informationen auf dem Scoresheet verdecken.

- `has_stains`

  Verfärbungen auf dem Blatt werden mit dieser Eigenschaft festgehalten. Dabei kann es sich beispielsweise um einen Kaffeefleck handeln.

- `is_perspective_asymmetric`

  Scoresheets werden meistens auf A4 Papier erfasst, ein Bild davon sollte ein Rechteck darstellen. Je nach Winkel der Aufnahme verläuft das Bild auf einen Fluchtpunkt zu, was die Schriften verziehen kann.

- `has_borderless_table_cells`

  Diese Eigenschaft beschreibt Scoresheets mit Tabellen, bei welchen die einzelnen Zellen nicht auf allen vier Seiten mit einem Rahmen umfasst sind. Dabei genügt es, wenn eine Seite nicht vorhanden ist.

Die folgenden Bilder zeigen Beispiel-Scoresheets mit den oben beschriebenen Eigenschaften. Im Chess-Scoresheet Corpus ist kein Scoresheet mit der Eigenschaft `has_stains` vorhanden, dementsprechend wird auch kein Beispiel aufgezeigt.

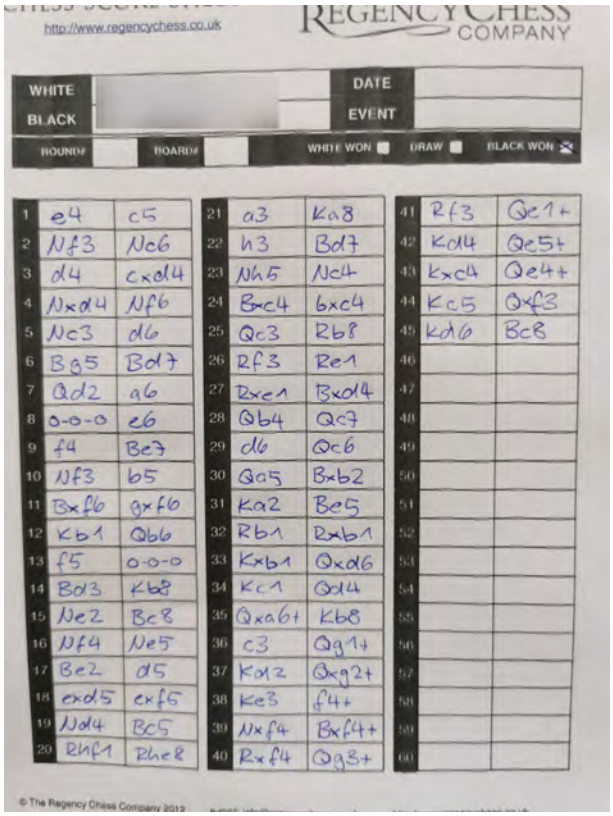


Abbildung 21: Scoresheet en_042_45.png mit den Eigenschaften `is_low_contrast`, `is_sheet_incomplete` und `is_perspective_asymmetric`

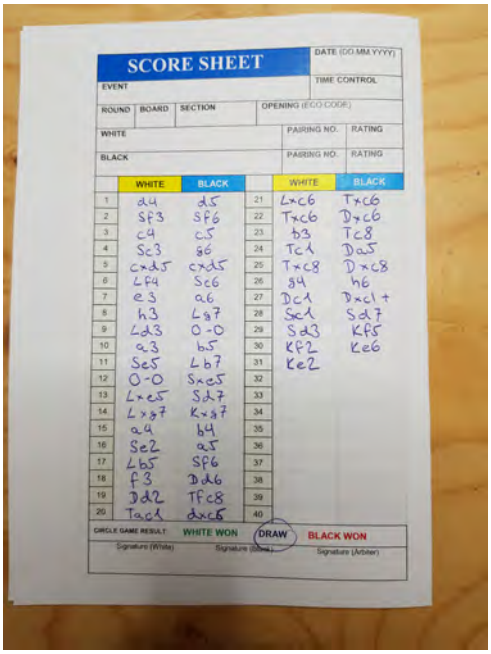


Abbildung 22: Scoresheet de_008_31.png mit der Eigenschaft `has_shadows`

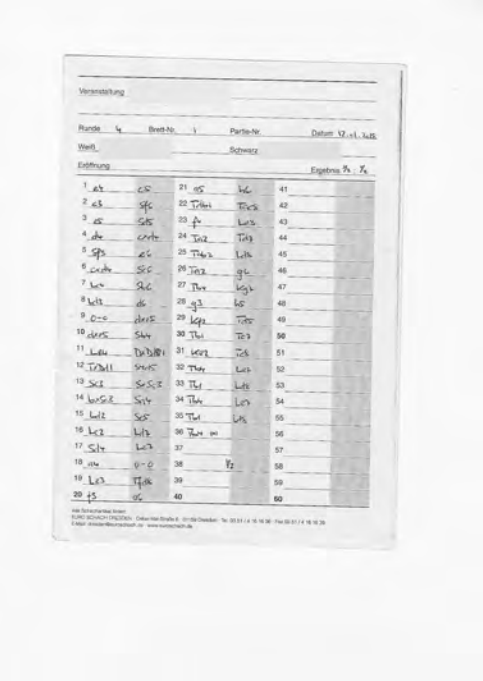


Abbildung 23: Scoresheet de_001_36.png mit der Eigenschaft `has_borderless_table_cells`

Halbzug-Boxen

In den `ply_boxes` werden die Halbzüge festgehalten. Dabei wird der effektiv auf dem Scoresheet aufgeschriebene Halbzug in `original_ply` notiert. Unter `corrected_ply` wird der korrigierte Halbzug notiert. Diese beiden Werte unterscheiden sich dann, wenn beim Erfassen eine inoffizielle Notation verwendet oder ein fehlerhaften Halbzug aufgeschrieben wurde. Der korrekte Halbzug wird in offizieller SAN geschrieben.

`box_position` enthält die Informationen, wo sich die Box um den Halbzug befindet. Die Höhe und Breite beschreiben die Grösse in Pixel, während die X- und die Y-Koordinaten den Mittelpunkt der Box in Pixel beschreiben.

Die `handwriting_properties` beschreiben Eigenschaften der Handschrift des erfassten Halbzugs. Trifft eine Eigenschaft auf die Halbzug-Box zu, so wird der Wert auf `true` gesetzt. Ist der Wert `false`, so besitzt die Box diese Eigenschaft nicht. Dasselbe gilt, wenn die Eigenschaft nicht in der Annotations-Datei vorhanden ist. Folgend werden die möglichen Eigenschaften beschrieben.

- `has_correction`

  Dieses Attribut beschreibt, ob bei einer Halbzug-Box eine Korrektur beim Ausfüllen vorgenommen wurde. Ein Buchstabe, der mit einem anderen Buchstaben überschrieben wurde, oder ein ganz oder teilweise durchgestrichener Buchstabe gilt als Korrektur.

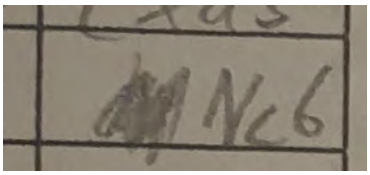


Abbildung 24: Beispiel für `has_correction` aus en_019_28.png

- `has_additional_text`

  Diese Eigenschaft beschreibt zusätzlichen Text innerhalb der Box. Dabei kann es sich um Buchstabenteile von angrenzenden Boxen handeln oder zusätzliche Informationen wie ein Zeitstempel.

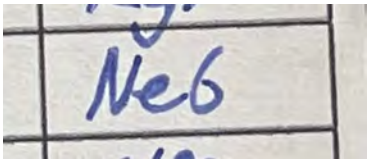


Abbildung 25: Beispiel für `has_additional_text` aus en_006_36.png

- `is_crossing_boxes`

  Ragen die Buchstaben aus der Box heraus, so wird diese Eigenschaft gesetzt. Dies tritt häufiger bei Buchstaben wie "g" und "f" auf.

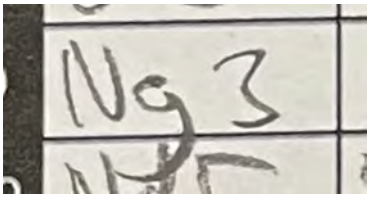


Abbildung 26: Beispiel für `is_crossing_boxes` aus en_037_33.png

- `legibility_level`

  Das Legibility Level beschreibt die Leserlichkeit der Handschrift. Dabei werden drei Stufen definiert: easy, medium und hard. Im JSON werden stellvertretend Zahlenwerte von 3, 2 und 1 verwendet.

  - `easy` - 3: Die Zeichenfolge ist auf den ersten Blick lesbar.
  - `medium` - 2: Die Zeichenfolge wird mit einem genaueren zweiten Blick erkannt.
  - `hard` - 1: Die Zeichenfolge ist nicht oder nur mit viel Mühe lesbar.

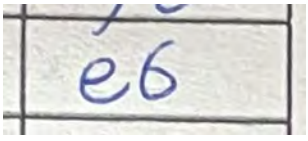


Abbildung 27: legibility_level : easy - 3

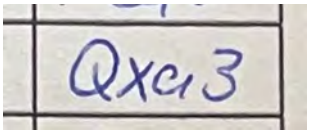


Abbildung 28: legibility_level : medium - 2

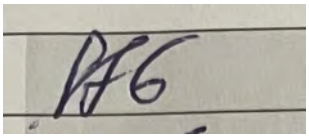


Abbildung 29: legibility_level : hard - 1

Das folgende Beispiel weist alle vordefinierten Handschrift-Eigenschaften auf, die im Chess-Scoresheet Corpus vertreten sind. Es beinhaltet zusätzlichen Text von der Box oberhalb, beinhaltet Korrekturen und ragt selber über den unteren Rand der Box hinaus. Der Halbzug ist nicht eindeutig identifizierbar und wurde demnach als schwer lesbar eingestuft.

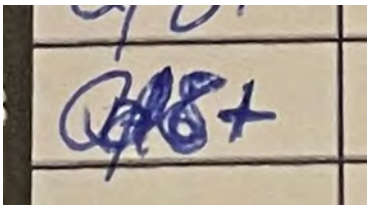


Abbildung 30: Halbzug "Qd8+"

```
{ ...
  {
    "corrected_ply": "Qd8+",
    "original_ply": "Qd8+",
    ...
    "handwriting_properties": {
        "has_correction": true,
        "has_additional_text":  true,
        "is_crossing_boxes": true,
        "legibility_level": 1
    }
  },
  ...
}
```

Listing 5: Handwriting Properties von Halbzug "Qd8+"

8.4 Vergleich Chess-Scoresheet Corpus und Corpus v4

In der Erarbeitung des Chess-Scoresheet Corpus wurde der Corpus v4 als Grundlage genutzt. Die positiven Aspekte wurden übernommen oder versucht zu optimieren und Schwachstellen wurden ausgemerzt. Es folgt eine Gegenüberstellung der beiden Corpora mit zusätzlichen Erklärungen zu den Chess-Scorsheet Corpus Designentscheidungen.

8.4.1 Zusammenführung von Scoresheet- und Zug-Informationen

Wird die Struktur der beiden Corpora verglichen, so ähnelt sich diese stark.

```

{
  "game_length": 32,
  "winner": "white",
  "language": "en",
  "handwriting_color": "blue",
  "table_positions": [ ... ],
  "sheet_properties": { ...},
  "ply_boxes": [ ... ]
}
```

Listing 6: Chess-Scoresheet Corpus

```
{
  "file_name": "en_0001_easy_32.
    csv",
  "game_length": 32,
  "won": "white",
  "language": "en",
  "difficulty": "easy",
  "keywords": [],
  "box_positions": [ ... ],
  "table_position": [ ...],
  "table_size": [ ... ]
}
```

Listing 7: Corpus v4

Im JSON Ausschnitt des Corpus v4 auf Zeile 26 befindet sich der Dateiname der Datei, welche die Zug-Informationen beinhaltet. Dieser ist im Chess-Scoresheet Corpus nicht zu finden. Bei der Überarbeitung wurde darauf Wert gelegt, dass alle Annotationen in einer Datei abgespeichert werden. Dies dient der Übersichtlichkeit und bringt eine einfachere Handhabung der Informationen mit sich.

Die Informationen der Spiellänge, des Gewinners, der Sprache und der Tabellen Positionen sind bereits im Corpus v4 vorhanden. Lediglich "won" wurde zu "winner" umbenannt, da dies aussagekräftiger ist. Die Werte der Attribute zeigen im Chess-Scoresheet Corpus die gleiche Struktur, wie im Corpus v4 auf.

Zusätzlich wurde das Attribut `handwriting_color` hinzugefügt. Dies kann für eine detailliertere Analyse genutzt werden. Es kann beispielsweise geprüft werden, ob ein OCR Engine für blaue Schriftfarbe schlechtere Predictions liefert, als für eine Schwarze.

Zusammenführung in Halbzug-Boxen

Die `ply_boxes` umfassen Informationen zum Halbzug, der Box-Position und zusätzliche Informationen zur Handschrift. Im Corpus v4 sind diese Informationen innerhalb des Corpus verteilter und weniger detailliert.

Die Spalte `has notation error` aus dem CSV im Corpus v4 wird im Chess-Scoresheet Corpus mit dem `corrected_ply` und `original_ply` abgedeckt.

	A	B	C	D	E	F	G	H
1	corrected move white	uncorrected move white	is hard to read	has notation error	corrected move black	uncorrected move black	is hard to read	has notation error
2	e4			0	0 d6			0 0
3	d4			0	0 Nf6			0 0
4	Nc3			0	0 g6			1 0
5	Be3			0	0 a6			0 0
6	a4			1	0 Bg7			0 0

Abbildung 31: Ausschnitt aus en_0031_easy_33.csv

Die Zuordnung der Boxen zu den Halbzügen musste über den Index berechnet werden. Im Chess-Scoresheet Corpus wurde die Handhabung benutzerfreundlicher gestaltet: Die Box-Positionen sind pro Halbzug-Box abgelegt. Eine zusätzliche Erweiterung sind die `handwriting_properties`. Diese umfassen Eigenschaften der Halbzüge. Im Corpus v4 war es nicht möglich, Eigenschaften für einen bestimmte Box festzulegen. Ausserdem wurden teilweise subjektive Einschätzungen gefordert. `is_hard_to_read` beispielsweise erwartet vom Erfasser, eine persönliche Einschätzung. Im Chess-Scoresheet Corpus wurden für die `handwiritng_properties` klare Vorgaben definiert, wie der Inhalt einzustufen ist (siehe Kapitel 8.3.3).

Scoresheet Eigenschaften

Die Möglichkeit Eigenschaften für das gesamte Blatt zu setzen, sollte weiterhin bestehen. Daher wurden das Attribut `sheet_properties` definiert. Diese Form ist eine verbesserte Version der `keywords` aus Corpus v4, da nun für beispielsweise `has_shadows` ein klares Attribut definiert wurde. Auf eine Liste wurde verzichtet, da aus der Sicht der Programmierung ein JSON Objekt einfacher und performanter zu verarbeiten ist.

Zusammenführung der Tabellen-Informationen

Die Tabellen Position und Tabellen Grösse aus dem Corpus v4 wurden in der Überarbeitung zusammengelegt. `table_positions` umfasst die Grösse und Positionen der Tabellen. Diese beiden Informationen stehen in einem direkten Zusammenhang und sind demnach sinnvollerweise miteinander konsolidiert.

9 Definition der Evaluierungsmetriken

Im folgenden Kapitel wird auf die Definition der Metriken eingegangen, welche in der Evaluierungspipeline implementiert wurden.

Um bestimmen zu können, welche Metriken für die Evaluierung von ChessReader notwendig sind, müssen zuerst die zu prüfenden und wertenden Komponenten identifiziert werden. Aus einer Liste der Komponenten lässt sich dann eine Sammlung an Metriken definieren, welche für diese Komponenten eine Aussage treffen können. Im Scoresheet Verarbeitungsprozess werden folgende Komponenten verwendet:

- *Der Boxerkennungsalgorithmus*, welcher benötigt wird, um die Erkennung der OCR Engines auf bestimmte Bereiche auf dem Scoresheet zu fokussieren. Erkannte Halbzüge werden immer einer Box zugewiesen. Den Boxerkennungsalgorithmus zu evaluieren ist notwendig, da alle folgenden Schritte darauf beruhen.
- *Mehrere OCR Engines*, welche eine Halbzug-Erkennung durchführen. Die Erkennungen der OCR Engines werden als Input für die Simple Chess Engine verwendet und tragen somit einen ausschlaggebenden Teil zum Endresultat bei, welches ein Benutzer oder eine Benutzerin im Frontend sieht. Funktioniert eine der OCR Engines aus dem Ensemble wesentlich schlechter, würde das einen klaren Einfluss auf das Endresultat haben. Aus diesem Grund muss die Genauigkeit der einzelnen OCR Engines evaluiert werden.
- *Die Simple Chess Engine*, welche aus den Antworten der OCR Engines mithilfe des Corrector-Algorithmus das endgültige, im Frontend dargestellte, Resultat generiert. Die Antworten der OCR-Engines können noch so gut sein, wenn die Simple Chess Engine aber fehlerhaft funktioniert und somit schlechtere Endresultate als ihre Inputwerte generiert, wird ChessReader als Ganzes unbrauchbar sein.
- *Der Skipping Algorithmus* ermöglicht Erkennungen, welche mit einer hohen Confidence als "richtig erkannt" markiert wurden, zu überspringen. Ist der Benutzer oder die Benutzerin in der Lage, Züge zu überspringen, welche eine hohe Confidence haben, aber trotzdem falsch sind, wird dies zu generell schlechteren Output-Resultaten führen.

Zusätzlich muss bestimmt werden, mit welchen "Ground-Truth" Daten eine Metrik berechnet wird. Ground-Truth Daten oder auch "Labels" werden in diesem Kontext als Referenz-Informationen über ein Scoresheet und dessen Halbzüge definiert. Ein Beispiel für ein Label ist die Information, dass der handgeschriebenen Halbzug "Ng4" wirklich "Ng4" ist. Diese Informationen sind notwendig, um erhaltene Predictions von ChessReader vergleichen zu können und um somit eine Aussage über deren Richtigkeit treffen zu können. Für die folgenden Metriken wird der in Kapitel 8 beschriebene Chess-Scoresheet Corpus verwendet. Alle Metriken werden mithilfe der Labels aus dem Corpus berechnet.

Die definierten Metriken sind:

- Die Accuracy der erkannten Boxen.
- Die Accuracy der erkannten Halbzüge pro Provider.
- Die Accuracy der erkannten Zeichen pro Provider.
- Der Einfluss von Handschrift-Eigenschaften auf die Provider Accuracy.
- Confusion Matrix zur Überprüfung von Zeichenverwechslungen pro Provider.
- Confusion Matrix zur Überprüfung des im "Skipping Algorithmus" definierten Confidence Thresholds.
- Übersicht über verwendeten Corpus.

In den folgenden Unterkapitel wird darauf eingegangen, wie die einzelnen Metriken berechnet werden, was sie aussagen und wie sie schlussendlich in der Evaluierungspipeline visualisiert werden. Zudem wird festgehalten, wenn eine Metrik einen der Tests aus dem ML Testing Framework umsetzt, welches in Kapitel 5.5 beschrieben wurde.

Folgend wird öfters der Begriff "Provider" verwendet. Ein Provider ist definiert als eine Engine, welche Halbzug-Erkennungen zurückgibt. Dies beinhaltet sowohl alle OCR Engines, welche Teil des Ensembles sind, als auch die Simple Chess Engine, welche das finale Resultat generiert. Diese namentliche Zusammenfassung wurde gemacht, da viele Metriken identisch auf eine OCR Engine des Ensembles, als auch auf den Output der Simple Chess Engine angewendet werden können. In den Diagrammen werden Abkürzungen für die einzelnen Provider verwendet, diese sind wie folgt:

- *ABBYY* - ABBYY FineReader
- *AWS* - Amazon Rekognition
- *Azure* - Azure AI Vision
- *CNN* - eigene Implementation eines CNNs
- *Google* - Google Vision API Document Text Detection
- *Google2* - Google Vision API Text Detection
- *ChessReader* - Output der Simple Chess Engine

9.1 Accuracy der erkannten Boxen

Die Accuracy der erkannten Boxen wird mithilfe der Intersection over Union (IoU) berechnet, welche einen Wert zwischen 0 und 1 ist und beschreibt wie sehr sich zwei Boxen überlappen. Die IoU für eine Box Prediction b_p und ein Box Label b_l ist definiert als:

$$IoU_b = \frac{|b_l \cap b_p|}{|b_l \cup b_p|} \tag{3}$$

wobei $|b_l \cap b_p|$ die Überlappung der Box Prediction und dem Box Label und $|b_l \cup b_p|$ die Vereinigung der Box Prediction und dem Box Label ist.

In Abbildung 32 wird ein Beispiel für die IoU gegeben. Das Box Label wird mit einem blauen und die Box Prediction mit einem roten Rahmen dargestellt. Die Überlappung (Intersection) (hier in hellblau eingefärbt) ist die Schnittmenge der beiden Boxen und die Vereinigung (Union) ist die Summe der drei Bereiche. IoU ist eine wichtige Metrik für den Boxerkennungsalgorithmus, da sie eine Erklärung bieten kann, wieso ein OCR System eine falsche Erkennung wiedergibt. In diesem Fall wurde das obere Ende der Buchstaben "N" und "d" abgeschnitten, was dazu führen könnte, dass eine OCR Engine, welche innerhalb dieser Box eine Prediction macht, anstelle von "Nd7" "Na7" erkennen könnte. Somit wird Test *Infra 3* des ML Test Frameworks umgesetzt, da Folgefehler, basierend auf unzureichender Boxerkennung, durch diese Metrik ersichtlich werden.

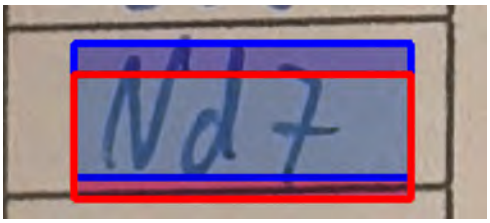


Abbildung 32: Intersection over Union Beispiel mit $IoU = 0.663$

Um die Box Accuracy Metrik für die Box Erkennungen über ein komplettes Scoresheet zu erhalten, kann der Mittelwert der IoU aller Box Predictions und den zugehörigen Box Labels innerhalb des Scoresheets berechnet werden. Die Box Accuracy für eine Box Erkennungen auf einem Scoresheet s ist somit:

$$IoU_s = \frac{1}{N_s} \sum_{b \in B} IoU_b \tag{4}$$

wobei N_s die Anzahl der Halbzüge im Scoresheet s ist.

Um nun die IoU Metrik über alle Scoresheets in einem Corpus zu erhalten, kann der Mittelwert aller IoU_s berechnet werden:

$$IoU = \frac{1}{n} \sum_{s \in S} IoU_s \tag{5}$$

wobei n die Anzahl der Scoresheets im Corpus ist.

Die IoU wird mithilfe von zwei Boxplot Diagrammen visualisiert. Die erste Visualisierung, dargestellt in Abbildung 33, fasst die IoU Metrik für die Erkennungen über alle Scoresheets in einem Corpus zusammen. Mithilfe dieser Darstellung ist es möglich, Outliers zu erkennen, für welche die Box Predictions entweder besonders schlecht, oder besonders gut funktioniert haben. Die Outlier werden im Diagram als weisse Punkte mit schwarzem Rand dargestellt. Die Ausreisser werden zudem separat aufgelistet, um auf die Scoresheets zurückzuführen. Dies entspricht Test *Infra 5* des ML Test Frameworks.

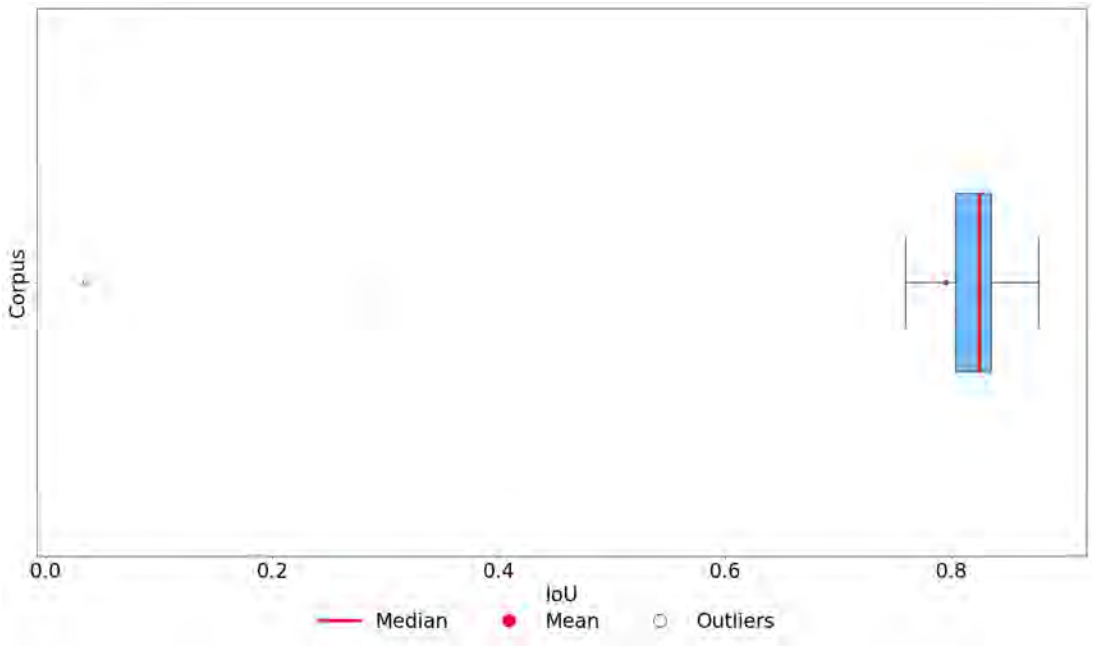


Abbildung 33: IoU Boxplot Diagram über kompletten Chess-Scoresheet Corpus

Die zweite Visualisierung, dargestellt in Abbildung 34, stellt die IoU der Boxerkennungen für jedes Scoresheet als separaten Boxplot dar. Mit dieser Darstellung ist es ebenfalls möglich festzustellen, für welche Scoresheets der Boxerkennungsalgorithmus schlecht funktioniert hat. Sie kann aber auch für eine genauere Aussage über die Erkennung innerhalb einzelner Scoresheet genutzt werden.

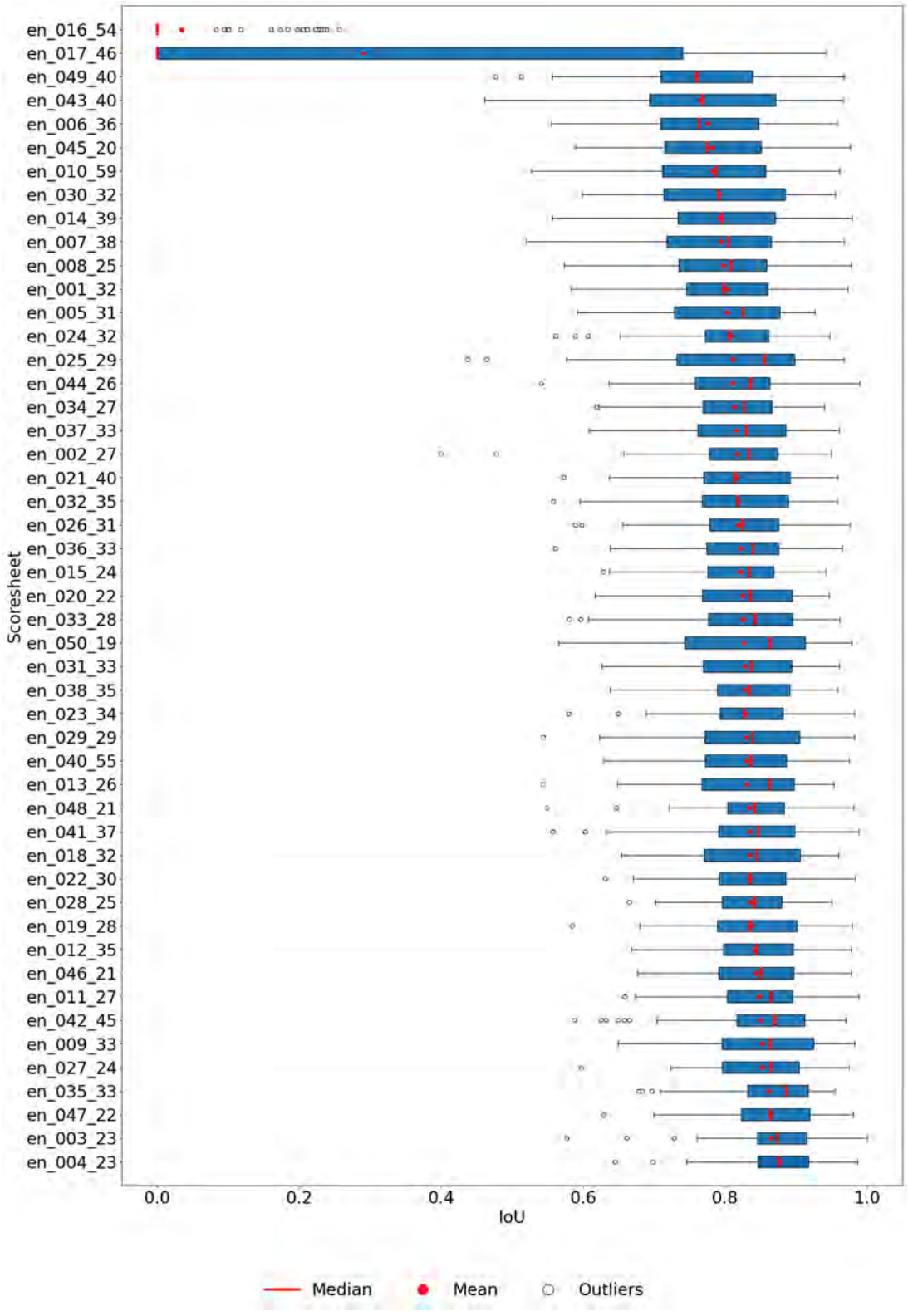


Abbildung 34: IoU Boxplot Diagram pro Schachblatt

9.2 Accuracy der Halbzug und Zeichen Erkennung

Um die Genauigkeit der erhaltenen Antworten der OCR Engines und dem Output der Simple Chess Engine zu evaluieren, wird eine Accuracy Metrik verwendet. Diese wird zu einem auf ganze Halbzüge angewendet, aber auch auf die darin enthaltenen Zeichen. Dies wird gemacht, da eventuell ein Provider sehr viele Zeichen innerhalb eines Halbzuges richtig erkennen könnte, aber für spezifischen Zeichen generell falsche Erkennungen generiert. Würde nur die Accuracy über die Halbzüge berechnet werden, würde dies zu einer 0% Trefferquote für Halbzüge mit den schwierigen Zeichen führen, was die Fähigkeiten des Providers nur unvollständig wiederspiegelt.

Bereits implementierte OCR Engines in ChessReader führen die Halbzug Erkennung auf zwei unterschiedliche Arten aus. Entweder wird die Erkennung auf ein ganzes Scoresheet ausgeführt, was eine Liste an erkannten Wörtern zurück gibt, wie bei den AWS [19], Azure [20] und Google [21] Provider, oder es wird mithilfe der zuvor erkannten Boxen, innerhalb jeder Box einzeln eine Erkennung ausgeführt. Dies ist für ABBYY [18] und beim selber implementierten CNN (siehe Kapitel 7.2) der Fall. Die WER, beschrieben in Kapitel 5.3.1, würde nur für Engines, welche eine Liste an Wörtern zurückgibt, ein sinnvolles Resultat liefern. Aus diesem Grund wird für die Genauigkeit der Halbzug-Erkennungen pro Scoresheet eine simplere Metrik $plyaccuracy_s$ implementiert, welche definiert ist für jedes Scoresheet s als:

$$plyaccuracy_s = 1 - \frac{E_s}{N_s} \tag{6}$$

wobei E_s die Anzahl falscher Erkennungen auf dem Scoresheet s und N_s die totale Anzahl and Halbzügen im Scoresheet s ist.

Um eine Accuracy Metrik für die Erkennung über den gesamten Corpus zu erhalten, wird der Mittelwert aller $plyaccuracy_s$ berechnet:

$$plyaccuracy = \frac{1}{n} \sum_{s \in S} plyaccuracy_s \tag{7}$$

Die CER ist grundsätzlich auf ChessReader anwendbar, wurde aber, um mit anderen Metriken einheitlich zu sein, von 1 abgezogen ($1 - CER$). Die CER kann grösser als 1 sein, wodurch die Character Accuracy kleiner als 0 sein könnte. Deshalb wurde im Falle, dass die Levenshtein Distanz zwischen dem Halbzug Label und der Halbzug Prediction grösser als die Länge des Labels ist, die Accuracy für diese Erkennung auf 0 gesetzt. Für eine Halbzug Prediction m_p und ein Halbzug Label m_l ist die Character Accuracy somit definiert als:

$$characcuracy_m = \begin{cases} 0 & \text{falls } lev(m_l, m_p) > len(m_p) \\ 1 - CER(m_l, m_p) & \text{sonst} \end{cases} \tag{8}$$

Für die Erkennungen über ein Scoresheet ist somit die Character Accuracy $characcuracy_s$

$$characcuracy_s = \frac{1}{N_s} \sum_{m \in M} characcuracy_p \tag{9}$$

und über den gesamten Corpus

$$characcuracy = \frac{1}{n} \sum_{s \in S} characcuracy_s \tag{10}$$

Visualisiert werden die beiden Accuracy Metriken identisch. Zu einem wird ein Boxplot Diagram generiert (siehe Abbildung 35), welches einen Überblick für die Accuracy über alle Scoresheets für die einzelnen Provider bietet. So ist erkennbar, ob ein Provider wesentlich besser, oder wesentlich schlechter als die anderen funktioniert. Dies implementiert Test *Model 5* aus dem ML Test Framework, da der Output der technisch komplexeren Simple Chess Engine direkt mit den Outputs der OCR Engines verglichen werden kann. Die Ausreisser des ChessReader Providers werden aufgelistet. Für das Diagramm wird zudem, in tabellarischer Form, der Accuracy Mittelwert (Roter Punkt im Diagramm), Accuracy Median (Rote Linie im Diagramm) und die Standardabweichung als Zahlenwerte für jeden Provider aufgelistet. Die Standardabweichung beschreibt die Streuung der Resultate um den Mittelwert.

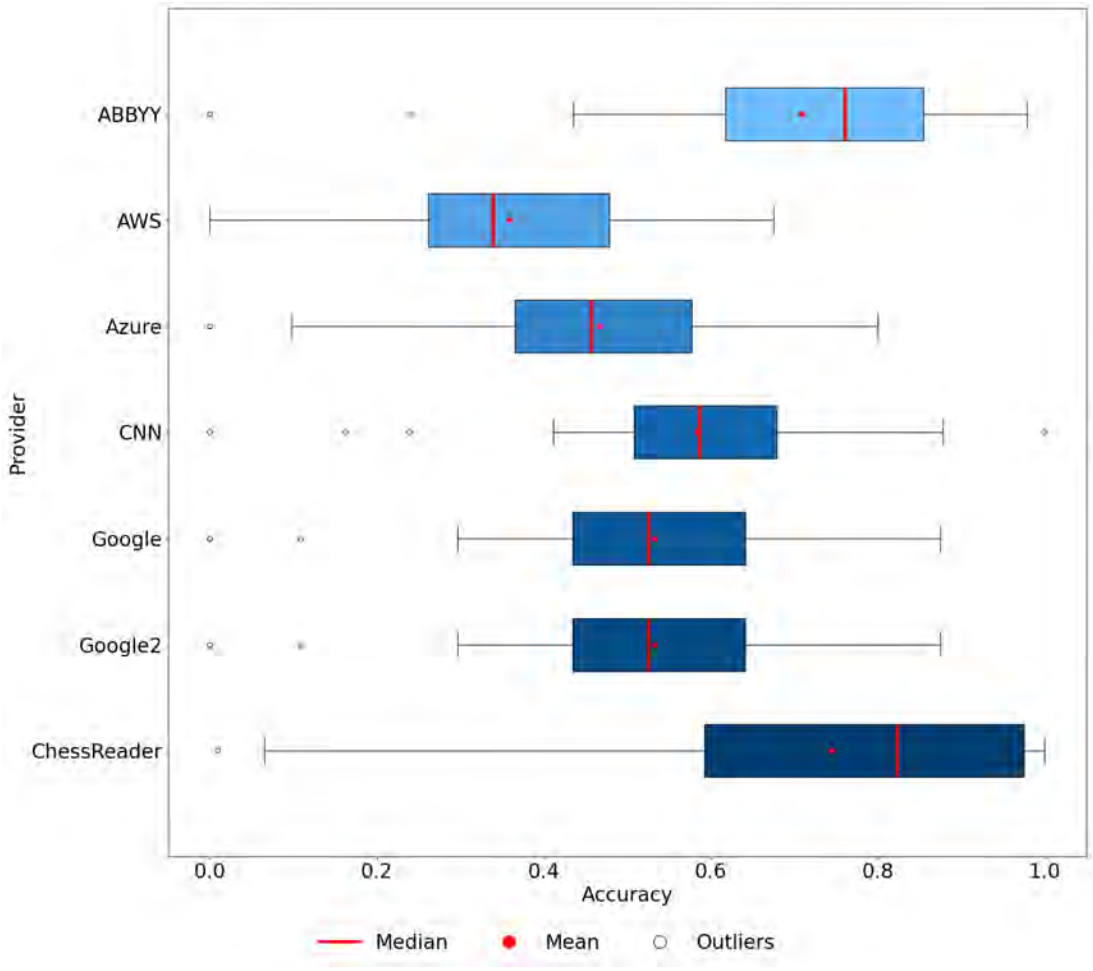


Abbildung 35: Halbzug Boxplots pro Provider

provider	total ply accuracy	total ply accuracy median	total ply accuracy std
ABBYY	70.9%	76.1%	19.7%
AWS	35.9%	33.9%	13.9%
Azure	46.6%	45.7%	15.8%
CNN	58.4%	58.7%	17.1%
Google	53.2%	52.5%	17.2%
Google2	53.2%	52.5%	17.2%
ChessReader	74.5%	82.3%	26.3%

Abbildung 36: Tabellarische Darstellung der im Boxplot enthaltenen Messwerte

Um einen genaueren Einblick über die Accuracy für einzelne Schachblätter zu erhalten, wird zudem ein sortiertes Balkendiagram verwendet, welches auf der Y-Achse die analysierten Schachblätter und auf der X-Achse die erzielte Accuracy aufzeigt (Siehe Abbildung 37). In dieser Darstellung wäre auch ersichtlich, wenn einzelne Provider bessere/schlechtere Erkennungen für spezifische Schachblätter bieten. Dies kann ein Zeichen dafür sein, dass diese Provider besondere Stärken oder Schwächen im Erkennen von Text unter bestimmten Umständen haben. Beispiele hierfür könnte die Erkennung von Text mit niedrigem Kontrast, oder verschwommene Aufnahmen sein.

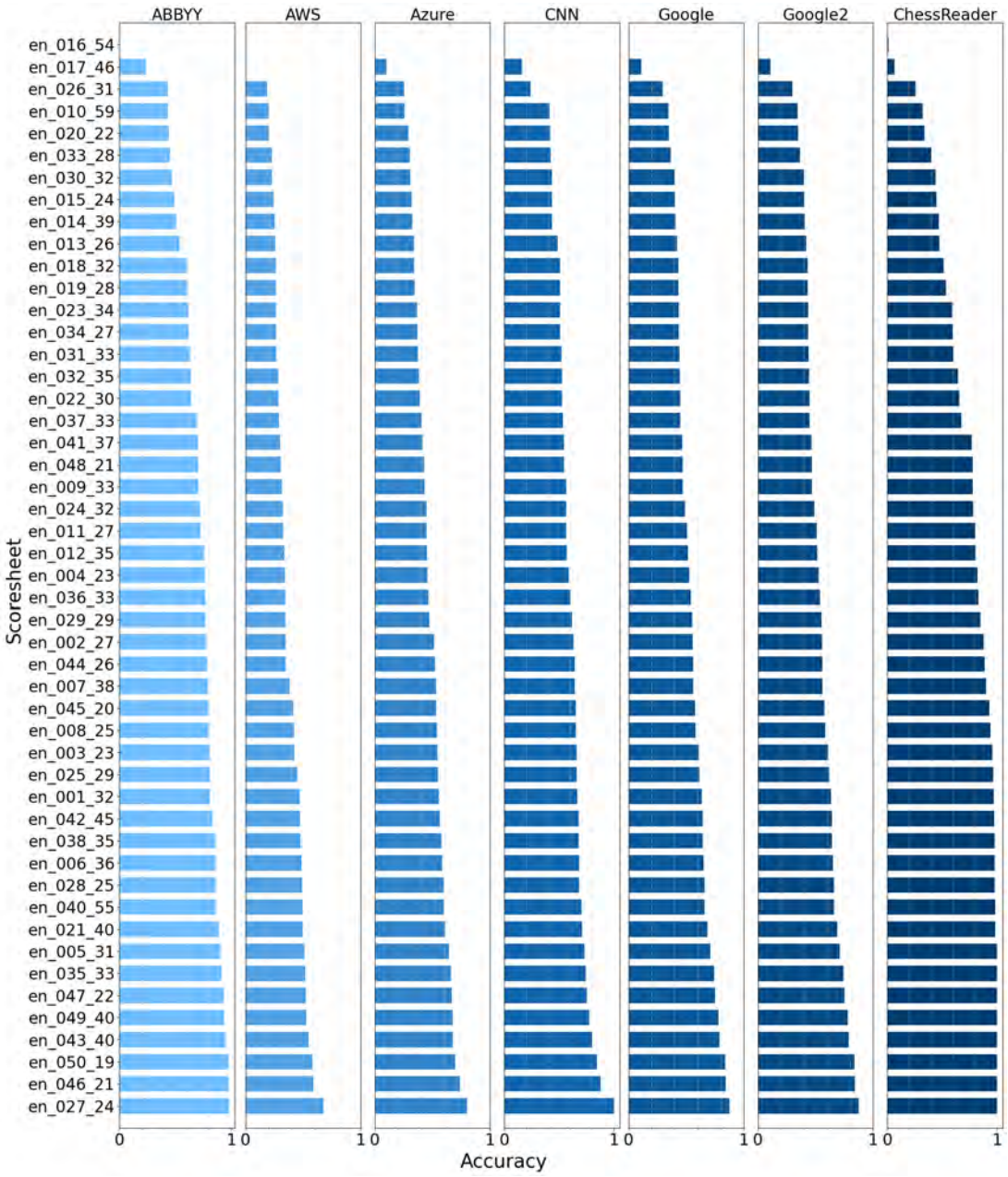


Abbildung 37: Halbzug Barchart pro Schachblatt

9.3 Einfluss von Handschrift-Eigenschaften

Mithilfe der zusätzlichen Handschrift-Eigenschaften, welche dem Chess-Scoresheet Corpus hinzugefügt wurden (siehe Kapitel 8), kann eine Aussage getroffen werden, wie stark einzelne Provider von verschiedenen Faktoren beinflusst werden. Hierfür wird die zuvor definierte *plyaccuracy* nach den Handschrift-Eigenschaften der Labels aufgeteilt. Es wird also für die Menge an Halbzug Labels, für welche eine Eigenschaft gesetzt ist, überprüft, wie viele davon richtig und falsch erkannt wurden. Dies wird für die Eigenschaften `additional_text`, `correction`, `crossing_boxes` und die legibility level gemacht. Visualisiert wird die Auswertung mithilfe von Kuchendiagrammen für jeden Provider. Ein Beispiel hierfür ist in Abbildung 38 für den ChessReader Provider dargestellt. Diese Metrik implementiert Test *Model 6* des ML Test Frameworks.

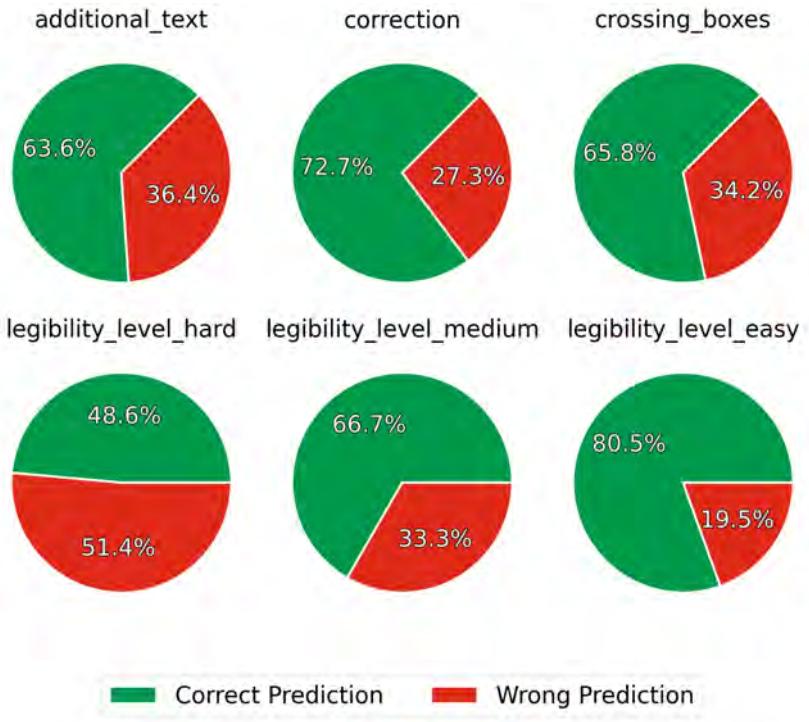


Abbildung 38: Handschrift-Eigenschaften Kuchendiagramm für ChessReader Provider

Um eine Gesamtübersicht über alle Provider zu erhalten, wird ein zusätzliches Punktediagramm für die drei Eigenschaften `additional_text`, `correction`, `crossing_boxes` generiert, wobei die X- und die Y-Achse die Prozentwerte von jeweils `additional_text` und `crossing_boxes` sind und die dritte Dimension `correction` als Farbwert dargestellt wird.

In Abbildung 39 ist die Tendenz ersichtlich, dass die Punkte von unten links nach oben rechts gehen und das die Farben ebenfalls in die selbe Richtung zeigen. Dies macht Sinn, wenn bedacht wird, dass eine OCR Engine, welche generell besser darin ist, Text zu erkennen, voraussichtlich auch generell besser für verschiedene Text Eigenschaften funktioniert. Dies muss allerdings nicht zwingend der Fall sein. Falls ein Provider mit Daten trainiert wurde, in welchen eine der Eigenschaften besonders oft vorkommt, dann wird dies in dieser Darstellung auch ersichtlich sein. In diesem Beispiel ist dies bei "Provider3" der Fall, welcher eine höhere `correction_accuracy` aufweist im Vergleich zu seiner `crossing_boxes_accuracy` und `additional_text_accuracy`.

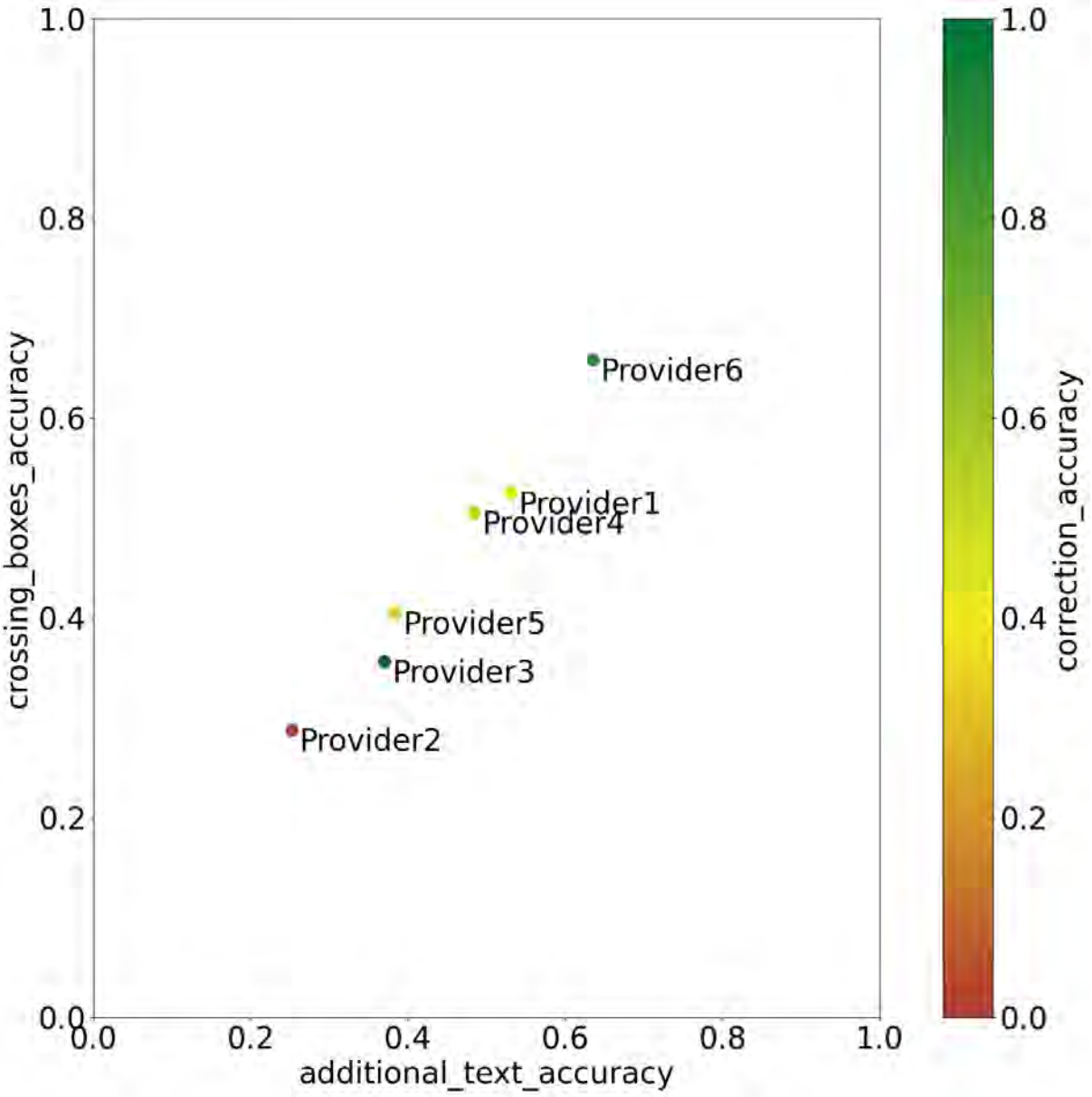


Abbildung 39: Handschrift-Eigenschaften Punktediagramm mit einem Ausreisser

9.4 Confusionmatrix einzelner Zeichen

Um einen Überblick über häufig verwechselte Zeichen wie "g" mit "a" und "b" mit "h" zu erhalten, wird pro Provider eine Confusion Matrix von den erkannten Zeichen im Vergleich zu den Label Zeichen berechnet. Dies hilft dabei, eine Aussage zu machen, ob ein Provider häufig über ähnlich aussehende Buchstaben stolpert. Da ChessReader in der Simple Chess Engine versucht, solche häufigen Verwechslungen zu minimieren, kann dessen Confusion Matrix zudem als Evaluierung des implementierten Confusion Moduls interpretiert werden (siehe Kapitel 6.1.2).

Die Berechnung der Verwechslungen wird anhand eines Beispiels gezeigt. Gegeben ist eine Liste an Halbzug Erkennungen und die zugehörigen Labels:

```
labels = ['Nf3','Nf6','Bg3','g6']

predictions = ['Nf3','Nfb','B3','bx6']
```

Es werden drei Situationen unterschieden: Die Prediction ist gleich lang als das Label, die Prediction ist kürzer als das Label und die Prediction ist länger als das Label. Für den Fall, dass die Prediction gleich lang wie das Label ist, können die Zeichen einfach einzeln miteinander verglichen werden. Es wird ein Dictionary aufgebaut, bei welchem sich der Schlüssel aus dem Label und Prediction Zeichen zusammensetzt und der Wert eine Zahl ist. Falls der Schlüssel im Dictionary noch nicht existiert, wird der Wert mit `1` initialisiert, andernfalls wird er inkrementiert. Für die erste Erkennung werden drei Werte in das Dictionary eingetragen. Da alle Zeichen korrekt erkannt wurden, sieht das Dictionary folgendermassen aus:

```
confusions = {'NN': 1, 'ff': 1, '33': 1}
```

Nach der zweiten Prediction "Nfb" werden die Schlüssel "NN" und "ff" inkrementiert und ein neuer Schlüssel "6b" für die Verwechslung vom Zeichen "6" mit dem Zeichen "b" wird dem Dictionary ergänzt:

```
confusions = {'NN': 2, 'ff': 2, '33': 1, '6b': 1}
```

Für die dritte Prediction tritt der Fall ein, dass die Prediction "B3" kürzer ist als das Label "Bg3". Es ist erkennbar, dass "b" nicht erkannt wurde, die beiden Zeichen "B" und "3" aber voraussichtlich stimmen. Um die beiden korrekt erkannten Zeichen in die Confusion Matrix einberechnen zu können, muss die Länge der Prediction angepasst werden, sodass sie identisch mit dem Label ist. Hierfür wird ein Füllzeichen an jeder möglichen Position eingefügt. Anschliessend wird zeichenweise überprüft, welche Kombination an Zeichen die kleinste Levenshtein Distanz zum Label hat.

Kombination	**Levenshtein Distanz**	**Notwendige Operationen**
$B3	2	$ löschen g einfügen
B$3	1	$ mit g ersetzen
B3$	2	g einfügen $ löschen

Tabelle 1: Mögliche Erweiterungen von "B3" auf drei Zeichen

Wie in Tabelle 1 ersichtlich hat die Kombination "B$3" die kleinste Levenshtein Distanz zum Label "Bg3". Somit wird für diese Kombination die Verwechslung berechnet:

```
confusions = {'NN': 2, 'ff': 2, '33': 2, '6b': 1, 'BB': 1, 'g$': 1}
```

Der Wert "g$" beschreibt keine wirkliche Verwechslung, weshalb solche Werte in den resultierenden Confusion Matrizen in einer separaten Zeile als "Not a Character" (NaC) notiert werden.

Für die Prediction "bx6" tritt der letzte Fall ein, dass die Prediction länger ist, wie das Label. In diesem Fall werden einzelne Zeichen entfernt. Tabelle 2 zeigt dies für das Label "g6" und Prediction "bx6".

Kombination	**Levenshtein Distanz**	**Notwendige Operationen**
b6	1	b mit g ersetzen
x6	1	x mit g ersetzen
bx	2	b mit g ersetzen x mit 6 ersetzen

Tabelle 2: Mögliche Kürzungen von "b13" auf zwei Zeichen

Sowohl die Kombination "b6" als auch "x6" haben eine Levenshtein Distanz von 1. In diesem Fall wird die erstbeste Kombination verwendet, was zu folgendem Confusion Dictionary führt:

```
confusions = {'NN': 2, 'ff': 2, '33': 2, '6b': 1, 'BB': 1, 'g$': 1, 'gb':1, '66':1}
```

Diese Berechnung wird für alle Predictions von jedem Provider durchgeführt.

Visualisiert wird das Dictionary schlussendlich als Confusion Matrix. Hierfür ist ein Beispiel in Abbildung 40 dargestellt. Dafür verwendet wurde eine Liste an Predictions vom Azure Provider über alle Scoresheets im Chess-Scoresheet Corpus. Wie bereits oben erwähnt, stellt die Zeile "NaC" nicht erkannte Buchstaben dar. Die Zeile "NaSAN" (Not a SAN) stellt Zeichen dar, welche nicht Teil des regulären SAN Zeichensatzes sind. Beispiele hierfür sind die Verwechslung von "c" mit einer Klammer "(" oder skuriller, die Verwechslungen von "g" mit dem Unicode Character U+00BA "º". Für den Azure Provider in der Matrix in Abbildung 40 ist diese Verwechslung einmal aufgetreten ist.

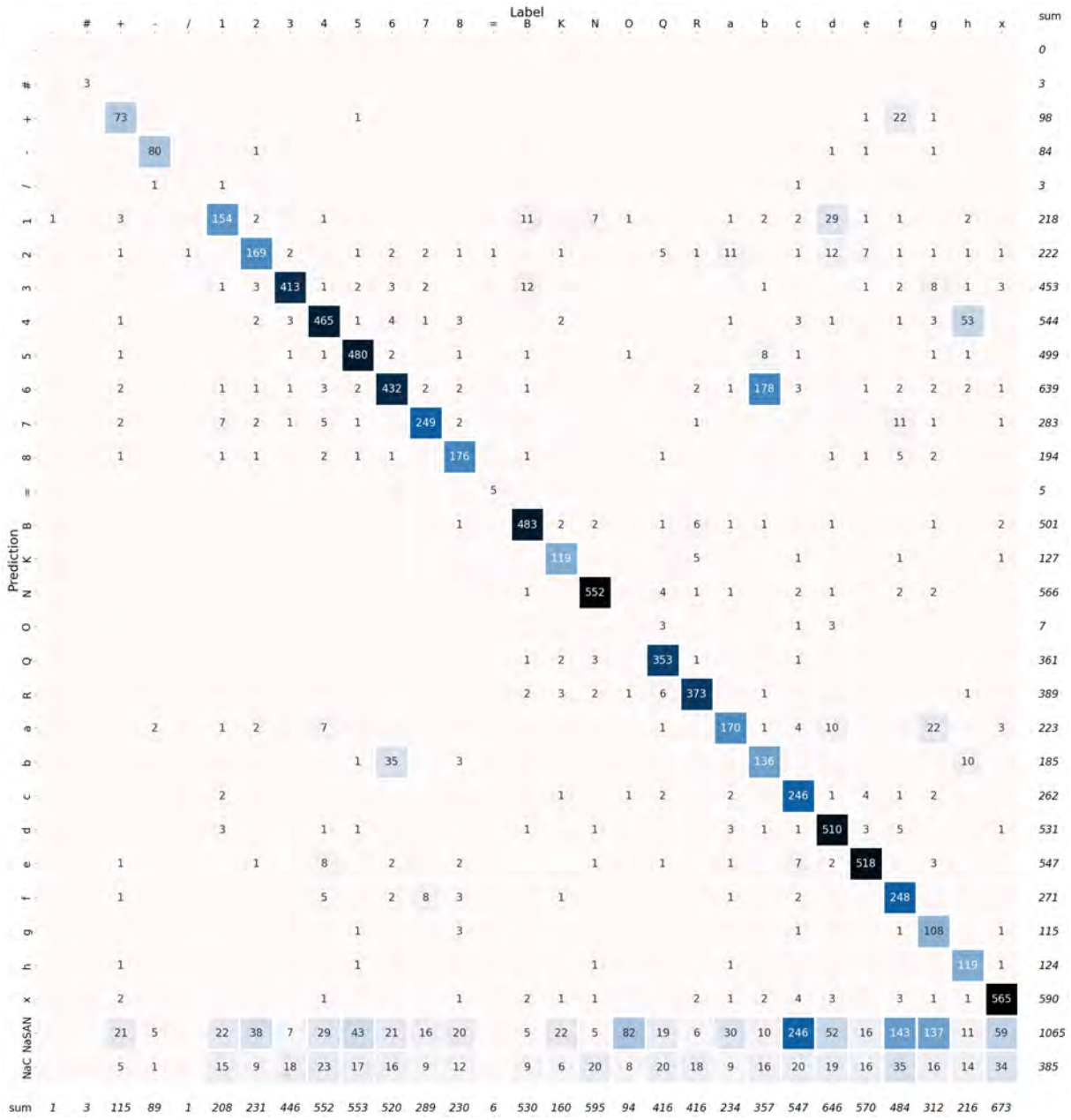


Abbildung 40: Character Confusion des Azure Providers über gesamten Chess-Scoresheet Corpus

9.5 Confusionmatrix der Zug Confidence

Der von ChessReader verwendete Skipping Algorithmus in der Simple Chess Engine ermöglicht es Zugerkennungen, welche mit hoher Zuversichtlichkeit als richtig eingeschätzt wurden, überspringen zu können. Hierfür wird ein Confidence Wert zwischen null und eins für jede Prediction berechnet. Befindet sich der Confidence Wert über einem bestimmten Schwellwert, kann der Zug während der Überprüfung übersprungen werden. Der Schwellwert, welcher aktuell in ChessReader verwendet wird, wurde mittels Trial-and-Error manuell evaluiert [7] und ist in der aktuellen Version von ChessReader auf $t = 0.9$ gesetzt. Der Skipping Algorithmus erhöht die Benutzerfreundlichkeit von ChessReader massiv, indem der manuelle Validierungsaufwand auf einem Minimum reduziert. Situationen in welchen eine Zugerkennung eine Confidence $> t$ hat, die Erkennung aber falsch ist, stellen allerdings auch ein Risiko dar. Solche Fehler werden meist erst auffallen, sobald Folgezüge keinen Sinn mehr ergeben. Aus diesem Grund ist es wichtig, den Skipping Algorithmus stetig auf seine Funktionalität zu prüfen. Es werden vier Fälle unterschieden:

- True Positive (TP): Die Confidence eines Zuges ist $> t$ und die Zugerkennung ist korrekt.
- False Positive (FP): Die Confidence eines Zuges ist $> t$ und die Zugerkennung ist falsch.
- True Negative (TN): Die Confidence eines Zuges ist $< t$ und die Zugerkennung ist falsch.
- False Negative (FN): Die Confidence eines Zuges ist $< t$ und die Zugerkennung ist korrekt.

Die Fälle True Positive und True Negative sind die erwarteten Resultate. False Negative ist zwar mit einer nicht notwendigen manuellen Kontrolle verbunden, ist aber nicht weiter tragisch. False Positives müssen auf ein Minimum gebracht werden.

In der Confidence Confusion Metrik wird für jede Erkennung des ChessReader Providers bestimmt, zu welchem der vier Fälle sie gehört. Das Resultat wird anschliessend in einer Confusion Matrix abgebildet. Ein Beispiel dafür ist in Abbildung 41 für eine Auswertung über den kompletten Chess-Scoresheet Corpus ersichtlich.

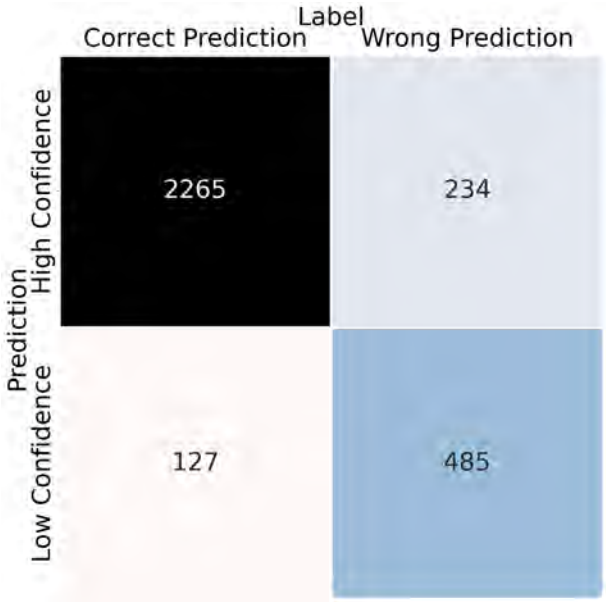


Abbildung 41: Confidence Confusion Matrix über kompletten Chess Scoresheet Corpus

Zusätzlich zur Confusion Matrix werden vier Messwerte berechnet: Accuracy, Precision, Recall und der F1-Score [32]. Diese sind definiert als

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{11}$$

$$precision = \frac{TP}{TP + FP} \tag{12}$$

$$recall = \frac{TP}{TP + FN} \tag{13}$$

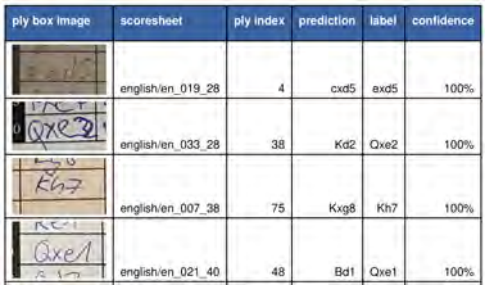
$$f1 = 2 * \frac{precision \cdot recall}{precision + recall} \tag{14}$$

- *accuracy* ist das Verhältnis aller korrekten Klassifizierungen ($TP + TN$) zur Summe aller Klassifizierungen.
- *precision* oder auch "positive predictive value" gibt das Verhältnis der Halbzüge mit hoher Confidence an. Precision ist hier eine sinnvolle Metrik, da ein False Positive grossen negativen Einfluss hat.
- *recall* oder auch "sensitivity" gibt das Verhältnis der Halbzüge an, welche eine hohe Confidence erhielten. Diese Metrik hilft, eine Aussage über die False Negatives zu machen, welche zu unnötigen Validierungsaufwand führen.
- $f1$ macht eine Aussage über die Balance zwischen Recall und Precision. Dieser Wert ist vorallem relevant, wenn eine grosse Menge an True Negatives vorhanden ist.

Für das Beispiel in Abbildung 41 sehen die Werte folgendermassen aus:

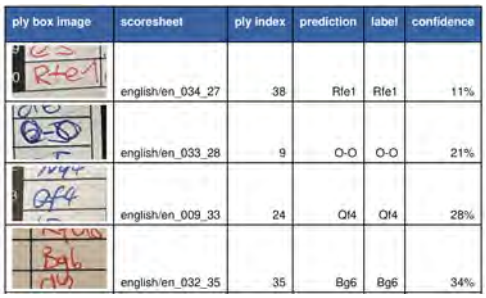
Accuracy	Precision	Recall	F1-Score
88.4%	90.6%	94.7%	92.6%

Um einen genaueren Einblick in die False Positives und False Negatives zu erhalten, werden Predictions mit der höchsten Confidence, welche False Positives sind in einer Tabelle visualisiert. Das gleiche wird auch für Predictions mit der tiefsten Confidence, welche False Negatives sind, gemacht. Dies erlaubt es eventuelle Muster oder Lücken in der Verarbeitung zu erkennen. Ein Beispiel hierfür ist in Abbildung 42 gegeben.



ply box image	scoresheet	ply index	prediction	label	confidence
	english/en_019_28	4	cxd5	exd5	100%
	english/en_033_28	38	Kd2	Qxe2	100%
	english/en_007_38	75	Kxg8	Kh7	100%
	english/en_021_40	48	Bd1	Qxe1	100%

(a) False Positives



ply box image	scoresheet	ply index	prediction	label	confidence
	english/en_034_27	38	Rfe1	Rfe1	11%
	english/en_033_28	9	O-O	O-O	21%
	english/en_009_33	24	Qf4	Qf4	28%
	english/en_032_35	35	Bg6	Bg6	34%

(b) Negatives

Abbildung 42: Confidence Confusion Tabellenausschnitt

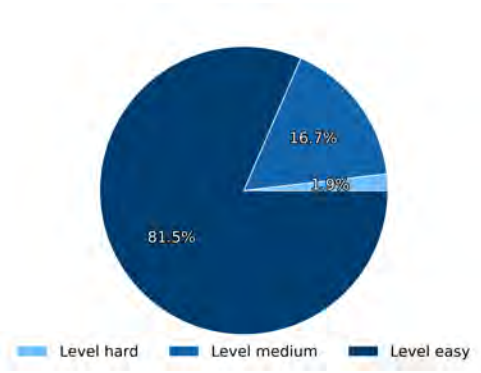
9.5.1 Übersicht über verwendeten Corpus

Ein Übersicht über den Corpus zu geben, welcher für eine Evaluierung verwendet wurde, ist nicht direkt eine Metrik. Es erlaubt allerdings die anderen Metriken in den Kontext zu setzen, für welche Daten sie generiert wurden. Dies ist entspricht dem *Data 1* Test aus dem ML Test Framework.

Für den verwendeten Corpus wird in tabellarischer Form aufgeführt, wie viele Scoresheets und Halbzüge darin enthalten sind. Es werden auch die verschiedenen vorhandenen Scoresheet Eigenschaften und Halbzug Eigenschaften aufsummiert. Zusätzlich wird in einem Kuchendiagramm die Aufteilung der verschiedenen legibility level visualisiert.

property	count
legibility level hard	3
legibility level medium	27
legibility level easy	132
has additional text	22
has correction	0
is crossing boxes	30

(a) Aufsummierung der vorhandenen Halbzug Eigenschaften



(b) Verhätnisdiagramm der drei legibility level

10 Implementierung der Evaluierungspipeline

Basierend auf dem überarbeiteten ChessReader Backend (Kapitel 7), dem verbesserten Chess-Scoresheet Corpus (Kapitel 8) und den definierten Evaluierungsmetriken (Kapitel 9), wurde die Evaluierungspipeline, welche in Kapitel 4.3 konzeptionell beschrieben wurde, implementiert.

In den folgenden Unterkapitel wird auf die Implementierung der Pipeline eingegangen. Kapitel 10.1 gibt einen Überblick über den technischen Ablauf der Pipeline und Kapitel 10.2 beschreibt die Software Architektur.

10.1 Pipeline Ablauf

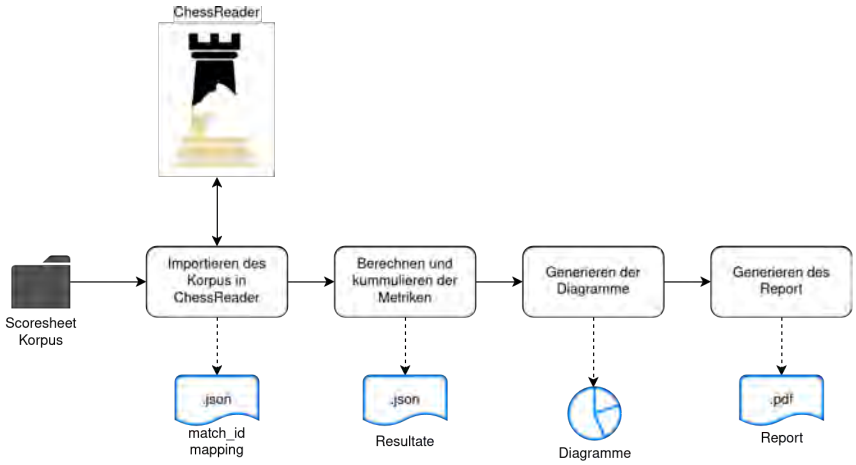


Abbildung 44: Ablauf der Evaluierungspipeline

Die Pipeline wird in mehrere Schritte aufgeteilt. Diese werden in Abbildung 44 visualisiert und in den folgendenden Unterkapitel genauer beschrieben.

10.1.1 Importieren der Scoresheets in ChessReader

In einem ersten Schritt wird ein ausgewählter Corpus importiert. Dieser muss die Struktur aufweisen, welche in Kapitel 8 beschrieben wurde. Der Import wird direkt über die API-Schnittstellen von ChessReader gemacht, um nicht direkt in den Code von ChessReader eingebunden zu sein. Jedes Scoresheet wird einzeln hochgeladen und die Box- und Zugerkennung darauf ausgeführt. Die dafür verwendeten API Endpunkte sind

- `POST http://chessreader:5000/api/matches` um ein Scoresheet hochzuladen.
- `GET http://chessreader:5000/api/matches/match_id/boxes` um die Boxerkennung auf dem Scoresheet auszuführen.
- `PUT http://chessreader:5000/api/mathces/match_id/boxes` um die letzte Box, welche ein Halbzug beinhaltet zu setzen.
- `GET http://chessreader:5000/api/matches/match_id/moves` um die Halbzug Erkennung auf dem Scoresheet auszuführen.

Für die Interaktion mit einem hochgeladenen Schachblatt über die API, verwendet ChessReader eine eindeutige `match_id`. Da innerhalb von ChessReader keine Dateinamen gespeichert werden, muss die Zuweisung des hochgeladenen Scoresheets und der erhaltenen `match_id`, durch die Evaluierungspipeline verwaltet werden. Damit im Falle eines Absturzes oder Fehlers, der Corpus nicht neu importiert werden muss, wird die Zuteilung in einer Datei namens `file_matchid_mapping.json` abgelegt. Falls die Evaluierung zu einem späterem Zeitpunkt erneut ausgeführt wird und ein Eintrag für ein Scoresheet bereits in der Zuteilungsdatei existiert, wird der Corpus Import Prozess für dieses Scoresheet übersprungen. Auflistung 8 zeigt den Aufbau der Zuteilungsdatei. Der `base` Schlüssel wird verwendet, um die Datei einem Corpus zuweisen zu können.

```
{
"base": "ChessReader_Corpus/Corpus_v5_mini",
"/english/en_001_32": 111,
"/english/en_003_23": 112,
"/english/en_002_27": 113
}
```

Listing 8: Aufbau von `file_matchid_mapping.json`

10.1.2 Berechnen und Kummulieren der Metriken

Nach dem Import des Corpus in ChessReader, wird die Auswertung der Metriken gestartet. Die Metrikauswertung findet in zwei Schritten statt: Kalkulation und Kummulation.

Bei der Kalkulation werden alle Metriken, welche in Kapitel 9 beschrieben sind, für jedes Scoresheet berechnet. Die enstehenden Resultate werden in einer JSON Datei abgespeichert. Eine Metrik wird darin als Objekt dargestellt, welches für jedes Scoresheet das Resultat der zugehörigen Metrik beinhaltet. Für Fälle in denen ein Resultat für verschiedene Provider ausgewertet wird, werden innerhalb des Scoresheet Objekts die Resultate mit dem jeweiligen Provider-Namen abgespeichert. Dies ist in Auflistung 9 für die $plyaccuracy_s$ von zwei Scoresheets dargestellt.

```
...
"ply_accuracies": {
    "/english/en_001_32": {
        "ABBYY": 0.8253968253968254,
        "AWS": 0.38095238095238093,
        "Azure": 0.36507936507936506,
        "Google": 0.5079365079365079,
        "Google2": 0.5079365079365079,
        "ChessReader": 0.9841269841269841
    },
    "/english/en_003_23": {
        "ABBYY": 0.8478260869565217,
        "AWS": 0.4782608695652174,
        "Azure": 0.45652173913043476,
        "Google": 0.6521739130434783,
        "Google2": 0.6521739130434783,
        "ChessReader": 0.9130434782608695
    }
}
...
```

Listing 9: Aufbau eines Kalkulations-Metrik Objekts in `results.json`

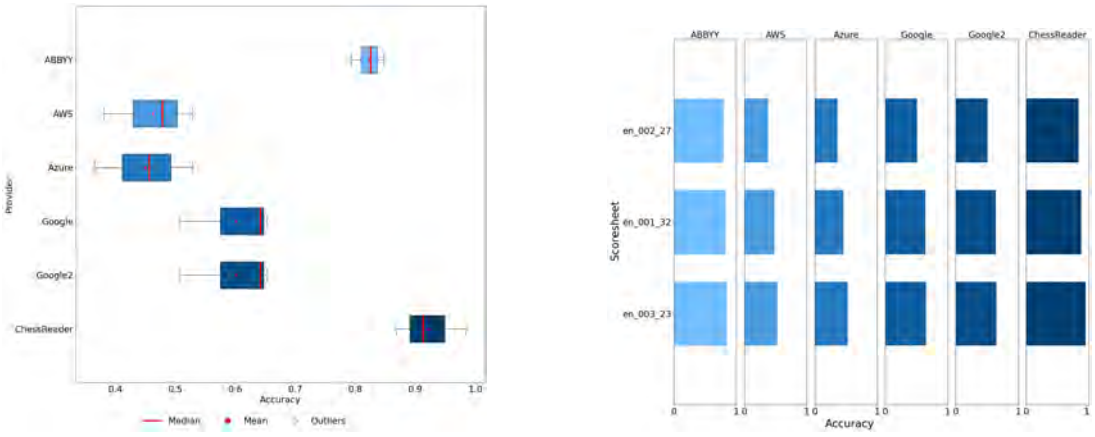
Nachdem alle Metriken für die einzelnen Schachblätter berechnet wurden, werden diese zu einem Gesamtresultat zusammengefasst. Die kummulierten Resultate werden ebenfalls im `results.json` mit dem Prefix `total_` festgehalten. Auflistung 10 zeigt ein Beispiel für die kummulierte Ply Accuracy.

```
    "total_ply_accuracy": {
        "ABBYY": 0.8218919141806754,
        "AWS": 0.46250504577001705,
        "Azure": 0.44996766366741753,
        "Google": 0.6005399516474168,
        "Google2": 0.6005399516474168,
        "ChessReader": 0.9216983302299134
    },
```

Listing 10: Aufbau eines Kummulations-Metrik Objekts in `results.json`

10.1.3 Generieren der Diagramme

Aus der zuvor befüllten `results.json` Datei, werden mithilfe der Visualisierungs-Library Matplotlib[14] die Diagramme generiert. Für das Beispiel der *plyaccuracy* wird zu einem das Per-Provider Boxplot Diagram, sowie das Per-Scoresheet Balkendiagram generiert (Siehe Abbildung 45).



(a) Per-Provider Boxplot

(b) Per-Scoresheet Balkendiagram

Abbildung 45: Generierte Diagramme anhand von Metrik Resultaten aus `results.json`

Die Diagramme werden dynamisch generiert. Falls ein zusätzlicher OCR Provider in ChessReader implementiert wird und für diesen ebenfalls ein Resultat zu einer Metrik vorhanden ist, wird dieses Resultat automatisch im Diagramm einfliessen. Die Grösse der Diagramme wird anhand der Inputdaten bestimmt. Ein Beispiel hierfür, im Vergleich zu Abbildung 45a, wird in Abbildung 46 für den zusätzlichen Provider "CNN" gezeigt.

[14] matplotlib.org

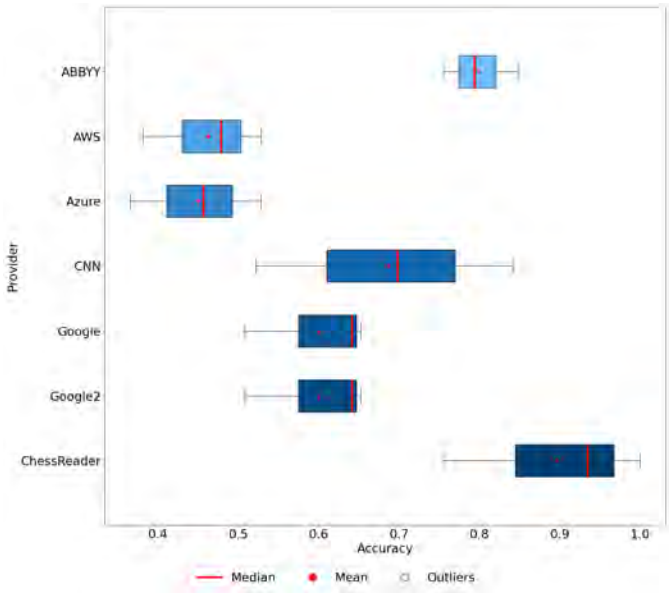


Abbildung 46: Per-Provider Boxplot mit dynamisch angepasster Grösse und zusätzlichen "CNN" Provider Resultat

Alle erstellten Diagramme werden im Ordner `output/diagrams` im Dateisystem abgelegt. Die Diagramme werden nummeriert, sodass sie immer in der gleichen Reihenfolge sind.

```
diagrams
├── 1_corpus
│   └── 1_corpus_legibility_levels.png
├── 2_accuracy
│   ├── 1_ply_accuracy_per_provider.png
│   ├── 2_ply_accuracy_per_scoresheet_per_provider.png
│   ├── 3_character_accuracy_per_provider.png
│   └── 4_character_accuracy_per_scoresheet_per_provider.png
├── 3_box_accuracy
│   ├── 1_box_accuracy_per_scoresheet.png
│   └── 2_box_accuracy_over_corpus.png
├── 4_character_confusion
│   ├── 1_[provider]_character_confusions.png
│   ├── ...
│   └── 6_chessreader_character_confusions.png
├── 5_confidence_confusion
│   └── 1_confidence_confusion.png
└── 6_handwriting
    ├── 1_handwriting_property_recognition_for_[provider]_provider.png
    ├── ...
    ├── 6_handwriting_property_recognition_for_chessreader_provider.png
    └── 7_handwriting_property_recognition_provider_comparison.png
```

Abbildung 47: Diagram Ordnerstruktur

10.1.4 Generieren des Report

Im letzten Schritt der Pipeline wird der Report mithilfe der Python Library ReportLab[15] im PDF Format generiert. Auf der Titelseite wird eine Zusammenfassung der wichtigsten drei Metriken *IoU*, *plyaccuracy* und *characcuracy* aufgezeigt. Auf der zweiten Seite wird eine Beschreibung der ausgewerteten Provider generiert. Dies dient zur Zusammenfassung der beteiligten Provider und den verwendeten Abkürzungen. Auf der dritten und vierten Seite wird ein Überblick über den Corpus gegeben, welcher für die Evaluierung verwendet wurde. Alle weiteren Kapitel werden anhand der in Abbildung 47 gezeigten Ordnerstruktur generiert. Zu den jeweiligen Diagrammen werden statische Texte ergänzt, welche die Visualisierungen erklären. Zudem werden, falls relevant, zusätzliche Tabellen eingefügt, welche die Metriken als Zahlenwerte darstellen. Ein Beispiel eines generierten Reports kann in Anhang A gefunden werden.

[15] reportlab.com

10.2 Architektur

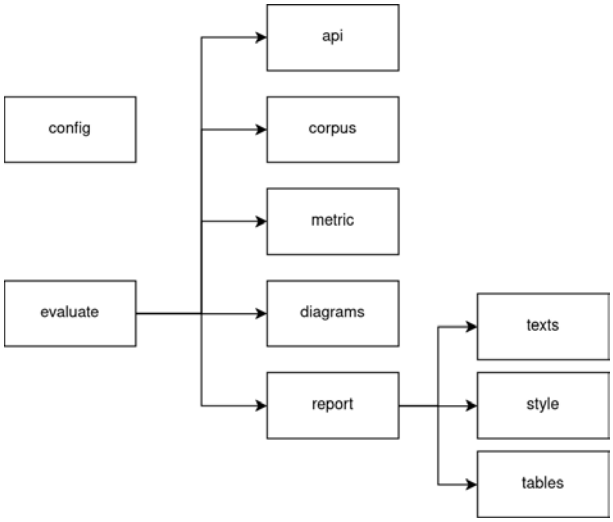


Abbildung 48: Strukturierung der Python Files

Die Architektur der Evaluierungspipeline ist über mehrere Python Dateien verteilt, wobei `evaluate.py` den in Kapitel 10.1 beschriebenen Ablauf abbildet. Grundgedanke der Architektur war es, diese nach dem "Separation of Concern" Prinzip [27] aufzubauen. Dies macht es möglich, dass einzelne Komponenten austauschbar sind. So ist es zum Beispiel möglich, die Generierung des Report-PDFs mit einem Output auf einer Webseite zu ersetzen, ohne dass dabei die anderen Schritte der Evaluierungspipeline überarbeitet werden müssen.

`config.py` beinhaltet alle Konfigurationsmöglichkeiten der Pipeline und der Visualisierungen. Hier wird zudem definiert, welcher Corpus für die Evaluierung verwendet werden soll. In Kapitel 10.2.1 wird genauer auf die einzelnen Konfigurationsmöglichkeiten eingegangen.

`api.py` beinhaltet die Schnittstelle zu ChessReader. Hier befinden sich die Funktionen für die notwendigen API Aufrufe, um ein Scoresheet hochzuladen und durchgeführte Erkennungen aus ChessReader auszulesen.

`corpus.py` beinhaltet den Corpus-Import Prozess.

`metric.py` beinhaltet die Berechnung der Evaluierungsmetriken.

`diagrams.py` beinhaltet die Funktionen zur Generierung der Diagramme.

`report.py` beinhaltet die Logik zur Report Generierung. Unterstützend hierzu beinhaltet `texts.py` alle statischen Texte, welche im Report angezeigt werden, `style.py` stellt das Dokument Styling zur Verfügung und `tables.py` bietet Hilfsfunktionen an, um Daten in tabellarischer Form aus `results.json` zu beziehen.

10.2.1 Konfigurationsmöglichkeiten

Die Datei `config.py` beinhaltet mehrere Konstanten, über welche die Evaluierungspipeline gesteuert werden kann. Folgend wird auf die wichtigsten eingegangen.

Korpusauswahl

Die Option `CORPUS_PATH` bestimmt den Corpus, welcher für die Evaluierung verwendet werden soll. `CORPUS_ALLOWED_LANGUAGES` beinhaltet eine Liste die aussagt, welche Sprachen an Scoresheets ausgewertet werden sollen. Dies wurde implementiert, da zum Stand dieser Arbeit ChessReader nur die englische SAN Notation unterstützt und deshalb Auswertungen von Scoresheets mit anderen Sprachen sehr schlechte Resultate erzielen.

```
    CORPUS_PATH = '/ChessReader_Corpus/Corpus_v5'
    CORPUS_ALLOWED_LANGUAGES = ["english", "german"]
```

Listing 11: Konfiguration des Corpus

Metrikauswahl

Eine in Kapitel 4.1 definierte Anforderungen war, Metriken aktivieren und deaktivieren zu können. Umgesetzt wurde dies mithilfe eines Enums, welcher alle definierten Metriken beinhaltet und einem Dictionary, welches für eine Metrik einen Boolean auf `True` oder `False` setzt. Nur wenn eine Metrik auf `True` gesetzt ist, werden die notwendigen Resultate berechnet, Diagramme generiert und ein Kapitel im Report dafür erstellt.

```
    METRIC_EVALUATION = {
    METRICS.PLY_ACCURACY: True,
    METRICS.CHARACTER_ACCURACY: True,
    METRICS.CONFIDENCE_CONFUSION: True,
    METRICS.CORPUS_PROPERTIES: True,
    METRICS.HANDWRITING_ACCURACY: True,
    METRICS.CHARACTER_CONFUSION: True,
    METRICS.BOX_ACCURACY: True,
    METRICS.PLY_STD: True,
    METRICS.CHARACTER_STD: True,
    METRICS.PLY_MEDIAN : True,
    METRICS.CHARACTER_MEDIAN : True,
}
```

Listing 12: Steuerung der Metrik-Aktivierung

Für bestimmte Metriken ist es möglich, dass sie Teil einer Tabelle in Kombination mit anderen Auswertungen sind. Falls eine der betroffenen Metriken deaktiviert ist, werden im finalen Report die entsprechenden Tabellenzellen auf "Metric Not Evaluated" gesetzt. Abbildung 49 zeigt die "Summary" Tabelle auf der ersten Seite des Reports, wenn `METRICS.CHARACTER_ACCURACY: False` gesetz ist.

provider	total character accuracy	total ply accuracy
ABBYY	Metric Not Evaluated	61.6%
AWS	Metric Not Evaluated	36.1%
Azure	Metric Not Evaluated	38.3%
Google	Metric Not Evaluated	48.6%
Google2	Metric Not Evaluated	48.6%
ChessReader	Metric Not Evaluated	71.6%

Abbildung 49: Summary Tabelle in einem Report, wenn Character Accuracy nicht evaluiert wird

11 Resultate

Im folgenden Kapitel wird auf die Resultate der Arbeit eingegangen. In Kapitel 11.1 wird der neu annotierten Chess-Scoresheet Corpus dargestellt und Kapitel 11.2 zeigt, wie die Evaluierungspipeline angewendet werden kann.

11.1 Corpus Annotierung

Als Grundlage für die Evaluierungspipeline dient der Chess-Scoresheet Corpus. Für die Annotierung des Corpus wurden die Informationen aus dem Corpus v4 verwendet und zusätzliche Eigenschaften definiert.

Der Chess-Scoresheet Corpus umfasst 57 Scoresheets mit 3608 Halbzügen.

Die folgende Tabelle bietet eine Übersicht, wie viele Schachblätter den Scoresheet-Eigenschaften zugeteilt wurden.

property	count
is perspective asymmetric	18
is low contrast	11
is sheet incomplete	6
has shadows	3
has borderless table cells	3

Abbildung 50: Übersicht der Anzahl vorhandenen Scoresheet mit entsprechenden Scoresheet-Eigenschaften

Die folgende Tabelle zeigt die Anzahl Boxen mit den entsprechenden Eigenschaften. Das Kuchendiagramm zeigt den prozentualen Anteil der legibility levels auf.

property	count
legibility level hard	120
legibility level medium	887
legibility level easy	2601
has additional text	783
has correction	46
is crossing boxes	1261

Abbildung 51: Übersicht der Anzahl Handwriting-Eigenschaften

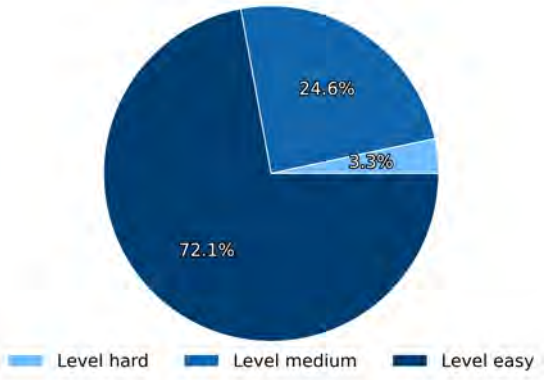


Abbildung 52: Übersicht der legibility level Anteile

ChessReader ist für die englische SAN ausgelegt. Für die Evaluierung und Generierung des Reports werden demnach nur die englischen Scoresheets verwendet, um das Ergebnis nicht zu verfälschen. Folgende Daten gelten für den englisch annotierten Corpusteil.

Der englische Corpusteil umfasst 49 Scoresheets mit 3111 Halbzügen.

property	count
legibility level hard	35
legibility level medium	729
legibility level easy	2347
has additional text	629
has correction	44
is crossing boxes	1023

Abbildung 53: Übersicht der Anzahl Handwriting-Eigenschaften der englischen Scoresheets

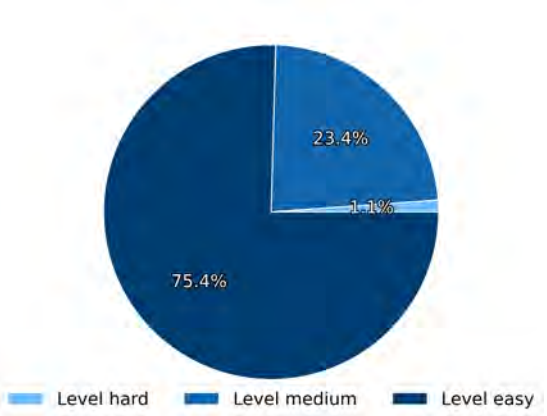


Abbildung 54: Übersicht legibility-level Anteile der englischen Scoresheets

11.2 Evaluierungspipeline

Eine Evaluierung von ChessReader mithilfe der Evaluierungspipeline kann folgendermassen durchgeführt werden. Eine detailliertere Anleitung hierzu ist in Anhang B ersichtlich.

1. Zuerst wird zum ChessReader_API Repository navigiert und das Backend im Evaluierungsmodus gestartet.

```
$ cd ChessReader_API/
$ python app.py evaluate

RUNNING EVALUATION DONT USE FOR PRODUCTION
 * Serving Flask app 'app'
 * Debug mode: on
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://192.168.1.174:5000
Press CTRL+C to quit
```

2. In der Konfiguration `evaluate/config.py` wird der Pfad des Corpus definiert, welcher für die Evaluierung verwendet werden soll. Zudem wird festgelegt, welche Sprache an Scoresheets verwendet wird.

```
1 CORPUS_PATH = '/BA/Corpus'
2 CORPUS_ALLOWED_LANGUAGES = ["english"]
```

3. Anschliessend kann die Evaluierung gestartet werden. Auf der Konsole wird dargestellt, welcher Schritt aktuell ausgeführt wird.

```
$ python evaluate/evaluate.py

Importing /BA/Corpus/english/en_019_28/en_019_28.png...
Importing /BA/Corpus/english/en_031_33/en_031_33.png...
.
.
Calculating results for /BA/Corpus/english/en_019_28...
Calculating results for /BA/Corpus/english/en_031_33...
.
.
Cummulating results for METRICS.CORPUS_PROPERTIES...
Cummulating results for METRICS.PLY_ACCURACY...
.
.
Generating diagramm for METRICS.CHARACTER_CONFUSION...
Generating diagramm for METRICS.PLY_ACCURACY...
.
.
Generating output report...
DONE!
Result JSON file is stored at evaluate/output/results.json
Report is stored at evaluate/output/
   chessreader_evaluation_report_20240606_2138.pdf
```

Der Report, welcher in der oben demonstrierten Ausführung generiert wurde, ist in Anhang A ersichtlich. Die Generierung des Reports dauerte gesamthaft 42 Minuten und 7 Sekunden, wobei das Importieren des verwendeten Corpus am meisten Zeit beanspruchte mit 41 Minuten und 4 Sekunden. Das Berechnen aller Metriken dauerte 7 Sekunden, das Erstellen der Diagramme 35 Sekunden und das Generieren des Reports 20 Sekunden.

Es sei betont, dass sich diese Arbeit auf den Aufbau der Evaluierungspipeline und nicht auf eine Auswertung von ChessReader mithilfe der Pipeline fokussiert. Aus diesem Grund werden, die im Report, erkennbaren Informationen nicht detailliert analysiert. Um allerdings den Nutzen des Reports zu demonstrieren, wird exemplarisch an ein paar Beispielen gezeigt, wie dieser analysiert werden könnte. Es werden dabei absichtlich keine vollständigen Schlussfolgerungen gezogen.

11.2.1 Zusammenhang Ply Accuracy und Box Accuracy

Wenn das "Ply Accuracy Per Scoresheet Per Provider" Diagramm betrachtet wird, ist ersichtlich, dass diese beiden Scoresheets en_016_54 und en_017_46 wesentlich schlechter abschneiden, als alle anderen im Corpus enthaltenen Scoresheets:

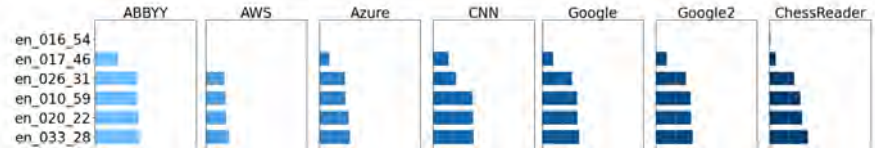


Abbildung 55: Ausschnitt aus "Ply Accuracy Per Scoresheet Per Provider" Diagramm aus Report in Anhang A

Bei Betrachtung der Box Accuracy wird auch ersichtlich, weshalb dies der Fall ist (siehe Abbildung 56). Die Box Accuracy über den kompletten Corpus ist mit 79.5% gut. Im Boxplot Diagram "Box Accuracy Over Corpus" sind jedoch zwei klare Ausreisser erkennbar. In der Auflistung unter dem Diagramm ist ersichtlich, dass diese Outlier ebenfalls die beiden Scoresheets en_016_54 und en_017_46 sind. Die Boxen auf diesen beiden Scoresheets werden nicht korrekt erkannt, was zu Folgefehlern in der Halbzug-Erkennung führt.

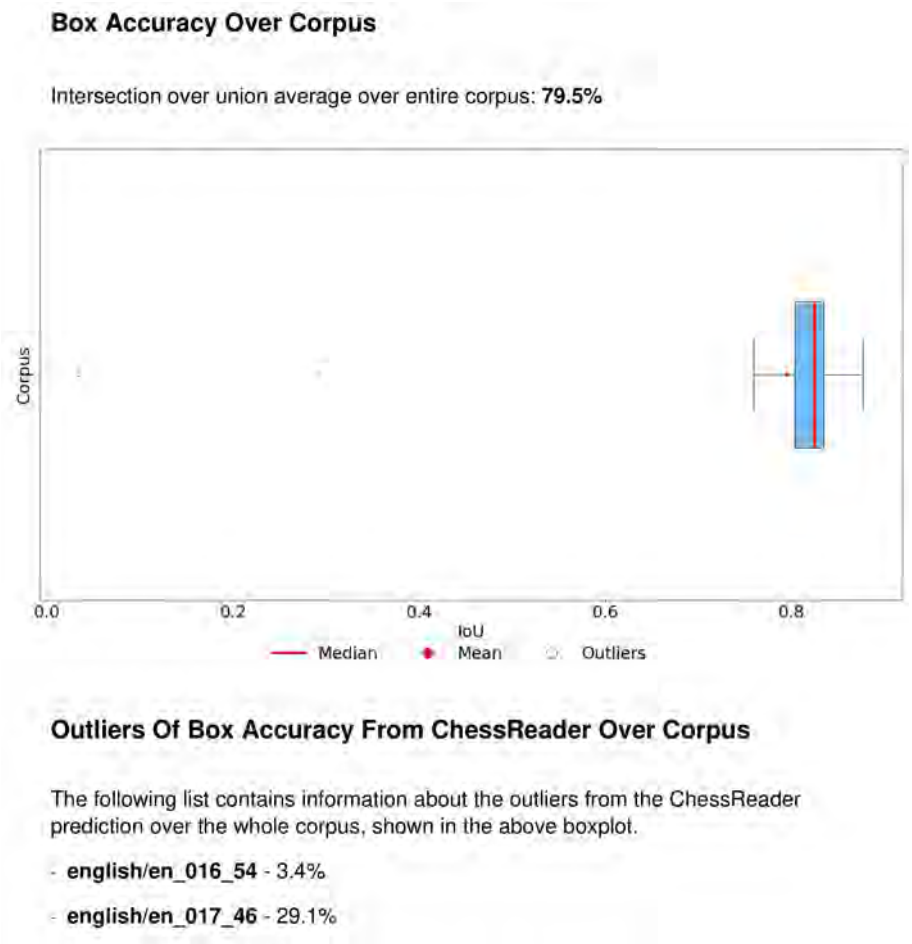


Abbildung 56: Ausschnitt aus Kapitel "Box Accuracy" aus Report in Anhang A

11.2.2 Bewertung der verwendeten OCR Engines

Mit Hilfe der Übersicht auf der ersten Seite, kann die Genauigkeit der verschiedenen OCR Engines und dem Output von ChessReader auf einen Blick oberflächlich verglichen werden. Hier ist erkennbar, dass ChessReader die Erkennungen aus dem Ensemble weiter verbessert. Dies bildet sich in der "total character accuracy" und "total ply accuracy" ab.

provider	total character accuracy	total ply accuracy
ABBYY	86.9%	70.9%
AWS	63.0%	35.9%
Azure	80.4%	46.6%
CNN	85.0%	58.4%
Google	81.9%	53.2%
Google2	81.9%	53.2%
ChessReader	89.2%	74.5%

Abbildung 57: Summary aus Report in Anhang A

Interessanter ist die Darstellung im letzten Kapitel "Handwriting Property Recognition Provider Comparison". Darin wird auf die Fähigkeit der einzelnen Provider eingegangen, mit verschiedenen Handschrift-Eigenschaften und den verschieden legibility level umzugehen. Hier ist ersichtlich, dass der AWS Provider generell die schlechtesten Erkennungen generiert. Der Azure Provider scheint ähnlich schlecht mit schwer erkennbaren Zügen (legibility level hard) umgehen zu können wie AWS, generiert aber wesentlich bessere Resultate für leicht erkennbare Züge (legibility level easy). Es ist ebenfalls erkennbar, dass die beiden Provider Google und Google2, welche unterschiedliche Erkennungsmodi verwenden (siehe Kapitel 5.3) identische Resultate liefern. Dies lässt darauf schliessen, dass diese beiden Modi in diesem Fall keinen Unterschied auf die Erkennung haben.

index	has additional text	has correction	is crossing boxes	legibility level hard	legibility level medium	legibility level easy
ABBYY	53.3%	47.7%	52.4%	28.6%	55.3%	74.1%
AWS	24.3%	15.9%	28.1%	0.0%	18.0%	39.2%
Azure	37.0%	20.5%	35.6%	2.9%	29.8%	51.4%
CNN	51.0%	36.4%	51.7%	22.9%	50.5%	58.8%
Google	38.5%	36.4%	40.6%	11.4%	29.6%	59.0%
Google2	38.5%	36.4%	40.6%	11.4%	29.6%	59.0%
ChessReader	57.1%	68.2%	60.6%	31.4%	59.9%	75.6%

Abbildung 58: Ausschnitt aus Kapitel "Handwriting Property Recognition Provider Comparison" aus Report in Anhang A

11.2.3 Analyse von Zeichenverwechslungen

Wenn die Matrizen für die verschiedenen Provider in Kapitel "Character Confusion" genauer betrachtet werden, ist ersichtlich, dass eine der häufigsten Verwechslungen bei allen OCR Engines ausser dem CNN, "b" zu "6" ist. In der Confusion Matrix von ChessReader ist diese Verwechslung praktisch komplett behoben. Eine Verwechslung, welche im ChessReader Output relativ häufig auftritt, aber nicht bei den OCR Engines, ist "e" mit "c". Im Kapitel "Top False Positive Results", ist ebenfalls ersichtlich, dass unter den False Positives mit der höchsten Confidence drei Züge sind, bei welchen ein "e" mit einem "c" verwechselt wurde:

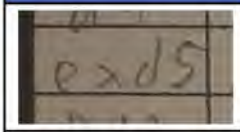
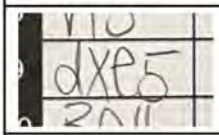
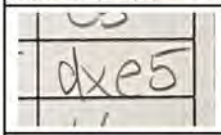
ply box image	scoresheet	ply index	prediction	label	confidence
	english/en_019_28	4	cxd5	exd5	100%
	english/en_031_33	16	dxc5	dxe5	100%
	english/en_031_33	17	dxc5	dxe5	100%

Abbildung 59: Ausschnitt aus Kapitel "Top False Positive Results" aus Report in Anhang A

Dies könnte auf das implementierte Confusion und Confidence Modul in der Simple Chess Engine zurückgeführt werden. Zeichen welche sehr häufig verwechselt werden, werden auch besser korrigiert, da sie im Confusion Modul eine höhere relative Verwechslungswahrscheinlichkeit erhalten (siehe Kapitel 6.1.2). Die Zeichen-Verwechslungen, welche dem CNN Provider unterlaufen, scheinen von blossem Auge näher an den Verwechslungen des ChessReader Outputs zu liegen.

12 Diskussion und Ausblick

Im folgenden Kapitel wird darauf eingegangen, ob die, im Kapitel 4.1, definierten Ziele an die Evaluierungspipeline und den Chess-Scoresheet Corpus erreicht wurden. Zudem wird ein Ausblick über mögliche zukünftige Arbeiten gegeben.

12.1 Erreichung der Zielsetzung

12.1.1 Chess-Scoresheet Corpus

Der Aufbau des Chess-Scoresheet Corpus war nicht von Beginn an Teil der Ziele. Es wurde jedoch schnell ersichtlich, dass für eine vollautomatisierte Auswertung von Scoresheets eine solide Datengrundlage verfügbar sein muss.

Der aufbereitete Chess-Scoresheet Corpus ist dank des verwendeten JSON Formats, einfach digital auswertbar. Die ergänzten Meta-Informationen über die Scoresheets und die Halbzüge darin, können als Basis für eine detailierte Analyse verwendet werden. Zudem ist die Annotation, dank des offenen Designs, einfach um weitere Meta-Eigenschaften erweiterbar. Das Hinzufügen von komplett neuen Schachblättern ist jedoch nicht trivial, da JSON zwar vom Menschen lesbar ist, das Generieren der spezifischen Notation aber normalerweise von einem Computer gemacht wird. Dies ist eine Problematik, welche bereits beim Corpus v4 besteht und nun durch die Entfernung der CSV Datei noch prominenter ist. Auf diese Arbeit hatte dieses Problem keinen direkten Einfluss, muss jedoch, wenn der Corpus in Zukunft weiterverwendet und ausgebaut werden soll, gelöst werden. Im Ausblick wird auf einen möglichen Lösungsansatz eingegangen.

12.1.2 Evalierungspipeline

Folgend wird auf die, in der Zielsetzung definierten, Funktionalitäts- und Nebenanforderungen an die Pipeline eingegangen.

Funktionalitätsanforderungen

Die Evaluierungspipeline ist in der Lage eine vollautomatisierte Evaluierung über den standartisierten Chess-Scoresheet Corpus auszuführen.

Die definierten Metriken bieten einen Überblick über die einzelnen Verarbeitungsschritte von ChessReader und es ist möglich Lücken in der Verarbeitung, wie in Kapitel 11.2 demonstriert, aufzudecken. Die Auswertung der verschiedenen Corpus Metadaten ist zum aktuellen Zeitpunkt minimal gehalten. Es werden die Handschriftseigenschaften ausgewertet. Eine Auswertung der verschiedenen Scoresheet Metadaten wurde jedoch, aufgrund von Zeitdruck nicht implementiert. Die Grundlage hierfür ist aber vorhanden.

Die ausgewerteten Metriken sind in den beiden gewünschten Formaten, textbasiert und in einem Report, vorhanden. Der Report erfüllt die Anforderung, dass die wichtigsten Daten übersichtlich visualisiert und informativ dargestellt werden. Für das Textformat wurde klar unterschätzt, wie gross die Datenmenge der Metrik-Resultate schlussendlich sein wird. Die textbasierte Darstellung in der `results.json` Datei ist ca. 100'000 Zeilen lang für eine Evaluierung über den eng-

lischsprachigen Teil des Chess-Scoresheet Corpus. Dies hat zwar keinen negativen Einfluss auf den daraus generierten Report, verfehlt allerdings das Ziel, dass das Textformat zur Analyse einzelner Daten verwendbar sein soll.

Die Steuerung der Metriken ist über die Konfigurationsdatei möglich. Im Report werden nur Metriken ausgewertet, welche auch in der Konfigurationsdatei ausgewählt wurden.

Nebenanforderungen

Die Evaluierungspipeline interagiert, wie gewünscht, nur über die definierten API-Schnittstellen mit dem Backend. Die Code-Separierung ist somit gewährleisten. Falls dies gewünscht ist, könnte somit in Zukunft die Evaluierungspipeline in ein eigenes Git Repository ausgelagert werden.

Die Anwendung des Seperation of Concern Prinzips auf die Architektur der Evaluierungspipeline erfüllt die Zielsetzung, dass Änderungen an einem Schritt der Pipeline nur minimale Anpassungen an anderen Komponenten verursachen. Dies vereinfacht zudem die Einbindung von neuen Metriken.

Die bestehenden Metriken und Visualisierungen wurden so implementiert, dass Änderungen am ChessReader Backend wenig Einfluss auf die Evaluierungspipeline haben. Das Hinzufügen oder Entfernen von OCR Engines in ChessReader, bedarf einzig einer Anpassung in der Konfigurationsdatei der Evaluierungspipeline.

12.2 Ausblick

Die Evaluierungspipeline bietet eine solide Basis um eine detailierte Analyse über die Genauigkeit von ChessReader durchzuführen. Sie hat somit einen klaren Mehrwert für die Weiterentwicklung der Anwendung. In einem nächsten Schritt sollte die Evaluierungspipeline für diesen Zweck genutzt werden.

Die Pipeline selber kann in diversen Punkten ergänzt, oder verbessert werden. Ein Ansatz ist, die bestehenden Datenhaltung der Metriken in der `results.json` Datei zu überarbeiten, um Detail-Analyse der Rohdaten zu vereinfachen. Eine Option hierfür wäre das Ablegen der Daten in einer eigenen Datenbank, wie SQLite[16]. So wären Daten mit der Structured Query Language (SQL) filterbar. Diese Idee wurde bereits während dem Projekt besprochen, konnte allerdings aus Zeitgründen nicht mehr umgesetzt werden.

Die Pipeline könnte ebenfalls um weitere Metriken ergänzt werden, wie die Analyse des Einfluss verschiedener Scoresheet Eigenschaften, Stiftfarben oder Notationssprachen. Hilfreich wäre auch, wenn Diagramme anhand eines Filters nur für Scoresheets mit einer bestimmten Eigenschaft dargestellt werden können, was den *Model 6* Test des ML Test Frameworks noch besser umsetzen würde. Voraussetzung hierfür wäre jedoch, dass die Darstellung der Diagramme interaktiv ist. Dies könnte entweder mit einer eigenen Implementation einer Webansicht oder mit einem Dashboard Produkt wie Grafana[17] umgesetzt werden.

Generell könnten auch noch weitere Tests des ML Test Frameworks umgesetzt werden. Ein Beispiel hierfür wäre der Test *Infra 4: Model quality is validated before attempting to serve it* [24]. Dieser könnte umgesetzt werden, in dem die Evaluierungspipeline automatisch in einer Continuous Deployment (CD) Pipeline ausgeführt wird und Änderungen an ChessReader nur auf ChessReader.org deployed werden können, wenn diese eine ausreichende Erkennungsfähigkeit aufweisen.

Wenn der neu gebaute Chess-Scoresheet Corpus betrachtet wird, bietet dieser auch Erweiterungsmöglichkeiten. Das Ergänzen von neuen Scoresheets mit verschiedenen Eigenschaften, stellt dabei eine kontinuierliche Aufgabe dar. Um das Annotieren von Scoresheets im definierten JSON Format zu vereinfachen, könnte ein Werkzeug entwickelt werden, welches es möglich macht, ein Scoresheet über ein Benutzerinterface zu annotieren. Die gemachten Annotation könnten dann automatisch im JSON Format gespeichert werden.

[16] sqlite.org
[17] grafana.com

Literatur

[1] FIDE International Chess Federation, „FIDE Laws of Chess taking effect from 1 January 2023," in *FIDE Handbook*, 01/01/2023, Article 8: The recording of the moves. Adresse: https://handbook.fide.com/chapter/E012023.

[2] Chess.com. „Chess score sheet," Chess.com. (2024), Adresse: https://www.chess.com/terms/chess-score-sheet.

[3] K. P. Murphy, „Ensemble learning," in *Machine learning: a probabilistic perspective*, Ser. Adaptive computation and machine learning series, Cambridge, MA: MIT Press, 2012, Kap. 16, S. 580–581, ISBN: 9780262018029.

[4] B. Fischer, *Fischer score card- wikimedia commons*, 1970. Adresse: https://commons.wikimedia.org/wiki/File:Fischer_Score_Card.jpg.

[5] B. Horváth und C. Dreher, „Document Digitization for Chess Scorecards," unpublished Bachelor Thesis, Zurich University of Applied Sciences (ZHAW), 19. Juni 2020.

[6] A. Abduli, „Document Digitization for Chess Scorecards," unpublished Semester Thesis, Zurich University of Applied Sciences (ZHAW), 18. Dez. 2020.

[7] V. Caglayan und B. Nielsen, „Digitalisierung von Schach-Formularen mittels Character Recognition Ensembling und Zug-Korrektur," unpublished Bachelor Thesis, Zurich University of Applied Sciences (ZHAW), 6. Nov. 2021.

[8] N. Ambrosini und G. Hellinger, „Digitalization of Chess Scorecards," unpublished Bachelor Thesis, Zurich University of Applied Sciences (ZHAW), 17. Juni 2022.

[9] M. Hostettler und L. Boner, „Digitalization of Chess Score Cards," unpublished Semester Project, Zurich University of Applied Sciences (ZHAW), 23. Dez. 2022.

[10] M. Hostettler und L. Boner, „Digitalization of Chess Score Cards," unpublished Bachelor Thesis, Zurich University of Applied Sciences (ZHAW), Juni 2023.

[11] FIDE International Chess Federation, „FIDE Laws of Chess taking effect from 1 January 2023," in *FIDE Handbook*, 01/01/2023, Appendix C. Algebraic Notation. Adresse: https://handbook.fide.com/chapter/E012023.

[12] Beao, *Chessboard and chess pieces*, 24. Feb. 2010. Adresse: https://commons.wikimedia.org/wiki/File:SCD_algebraic_notation.svg.

[13] C. Douglas. „About Douglas Crockford." (2024), Adresse: https://www.crockford.com/about.html.

[14] C. Douglas. „JSON," Introducing JSON. (2024), Adresse: https://www.json.org/json-en.html.

[15] Google. „Google JSON Style Guide." (2024), Adresse: https://google.github.io/styleguide/jsoncstyleguide.xml?showone=Property_Name_Format#Property_Name_Format.

[16] C. Douglas. „Introducing JSON." (2024), Adresse: https://www.json.org/img/object.png.

[17] „What is OCR? - optical character recognition explained - AWS," Amazon Web Services, Inc. (2024), Adresse: https://aws.amazon.com/what-is/ocr/.

[18] Margarita Nefedova. „How to recognize text fields," Cloud OCR SDK. (21. Juni 2023), Adresse: https://support.abbyy.com/hc/en-us/articles/360017326359-How-to-recognize-text-fields.

[19] AWS. „Detecting text in an image - Amazon Rekognition.“ (2024), Adresse: https://docs.aws.amazon.com/rekognition/latest/dg/text-detecting-text-procedure.html.

[20] Microsoft. „How to call the read API - azure AI services.“ (27. Feb. 2024), Adresse: https://learn.microsoft.com/en-us/azure/ai-services/computer-vision/how-to/call-read-api.

[21] Google. „Detect text in images — cloud vision API,“ Google Cloud. (23. Mai 2024), Adresse: https://cloud.google.com/vision/docs/ocr.

[22] V. I. Levenshtein, „Binary codes capable of correcting deletions, insertions, and reversals,“ *Soviet physics. Doklady*, Jg. 10, S. 707–710, 1965. Adresse: https://api.semanticscholar.org/CorpusID:60827152.

[23] S. John. „Developing Linguistic Corpora: a Guide to Good Practice.“ (2004), Adresse: https://users.ox.ac.uk/~martinw/dlc/chapter1.htm.

[24] E. Breck, S. Cai, E. Nielsen, M. Salib und D. Sculley, „The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction,“ in *Proceedings of IEEE Big Data*, 2017.

[25] R. T. Fielding, „Representational State Transfer (REST).,“ in *Architectural Styles and the Design of Network-based Software Architectures"; Doctoral dissertation*, 2000, Kap. 5, S. 76–105. Adresse: https://api.semanticscholar.org/CorpusID:19536483.

[26] R. Muletta, *Arbeit an ChessReader Sommer 2023*, E-mail, 31. Mai 2024.

[27] P. A. Laplante, „What is separation of concern?“ In *What every engineer should know about software engineering*, Ser. What every engineer should know 40, OCLC: ocm74965184, Boca Raton: Taylor & Francis, 2007, S. 85, ISBN: 9780849372285.

[28] B. W. Guido van Rossum und A. Coghlan. „PEP 8 – Style Guide for Python Code,“ PEP 8 – Style Guide for Python Code. (9. Dez. 2023), Adresse: https://peps.python.org/pep-0008/#descriptive-naming-styles.

[29] A. George, A. Troussard und P.-H. Levieil, „Machine Learning for Chess Movement Recognition,“ unpublished Semester Project, Swiss Federal Institute of Technology Lausanne (EPFL), 21. Dez. 2023.

[30] L. Cieliebak. „Implementation of Machine Learning for Chess Movement Recognition Paper,“ Github. (28. März 2024), Adresse: https://github.com/LunaCiel/ChessReader.

[31] L. of Congress. „ISO-Sprachencode,“ Codes for the Representation of Names of Languages. (21. Dez. 2017), Adresse: https://www.loc.gov/standards/iso639-2/php/code_list.php.

[32] K. P. Murphy, „The false positive vs false negative tradeoff,“ in *Machine learning: a probabilistic perspective*, Ser. Adaptive computation and machine learning series, Cambridge, MA: MIT Press, 2012, Kap. 5, S. 180–184, ISBN: 9780262018029.

Verzeichnisse

Abbildungsverzeichnis

Tabellenverzeichnis

Auflistungsverzeichnis

A Evaluierungsreport

Es folgt ein automatisch generierter Report mithilfe aller englischen Scoresheets aus dem Chess-Scoresheet Corpus.

Report - ChessReader API v2.0.0

This report was created to evaluate the scoresheet processing pipeline of ChessReader and the used OCR engines.

Project: ChessReader
Version: 2.0.0
Date: 05. June 2024 21:43

Summary

The following table shows a summary of the predictions of the providers and their accuracies.

provider	total character accuracy	total ply accuracy
ABBYY	86.9%	70.9%
AWS	63.0%	35.9%
Azure	80.4%	46.6%
CNN	85.0%	58.4%
Google	81.9%	53.2%
Google2	81.9%	53.2%
ChessReader	89.2%	74.5%

Intersection over union average over entire corpus: **79.5%**

Providers

The following OCR provider were used to generate predictions for the plies in the scoresheets:
ABBYY, AWS, Azure, CNN, Google, Google2, ChessReader

The providers are displayed in the diagrams and tables with an abbreviation of the name. Following are the used abbreviations and the corresponding full name:

- **ABBYY** - ABBYY FineReader
- **AWS** - Amazon Rekognition
- **Azure** - Azure AI Vision
- **CNN** - Selfmade Convolutional Neural Network
- **Google** - Google Vision API Document Text Detection
- **Google2** - Google Vision API Text Detection
- **ChessReader** - ChessReader API

Note: The ChessReader provider is not an external OCR engine. It is a combination of the other providers and additional logic to get the best prediction. This prediction is the one used, in the frontend. The ChessReader provider is listed here, because it is represented in the diagrams in the same way as the other providers.

Corpus

The following data describes the corpus used for the evaluation and the different properties the scoresheets and the handwritten plies have.

Sheet Information

The following table shows the number of sheets and the sum of all plies in the corpus.

property	count
scoresheet	49
ply	3111

Sheet Properties

The following information describes the sheet properties of the scoresheets contained in the corpus.

is perspective asymmetric :
The scoresheet is displayed asymmetrically, because of the camera angle.
is low contrast:
The image is of low quality. This can be influenced by different factors. For example the image has a low resolution or the text on the scoresheet is blurry. Low contrast can also be considered low quality.
is sheet incomplete:
A scoresheet is incomplete if a corner of the sheet is missing.
has shadows:
The scoresheet is shadowy.
has borderless table cells:
The cells of the scoresheet table have no borders. One missing side is enough to set this property.

property	count
is perspective asymmetric	15
is low contrast	11
is sheet incomplete	3
has shadows	1

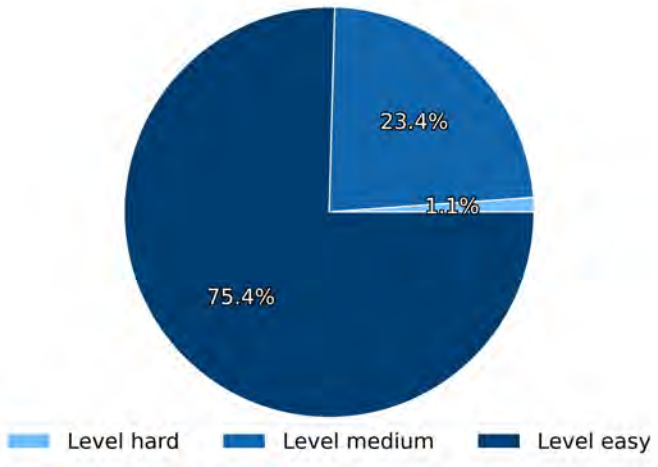
Handwriting Properties

The following information describes the handwriting properties of the plies in the corpus.

has additional text :
There is more text in the cell than just the ply. This can be a timestamp or just some other letters from a ply that is next to this one, which crosses the cell border.
has correction:
The ply was corrected or some letters are crossed out on the scoresheet by its author.
is crossing boxes:
Describes if the letters of a ply are crossing the cell border.
legibility level:
Describes the readability of the handwriting. Hard means the ply is illegible, medium means the ply is legible after a second look, and easy means the ply is legible at first glance.

property	count
legibility level hard	35
legibility level medium	729
legibility level easy	2347
has additional text	629
has correction	44
is crossing boxes	1023

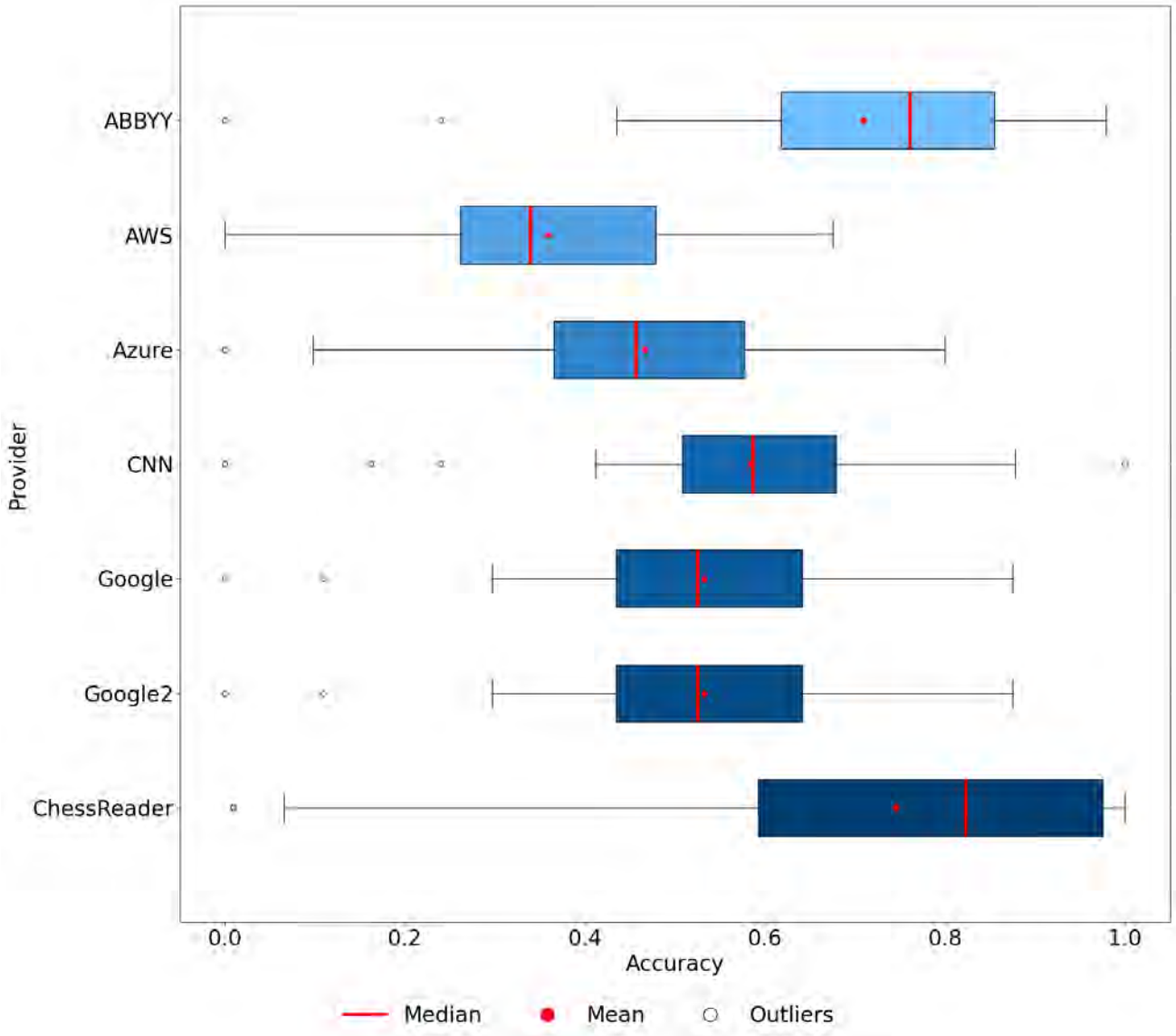
Corpus Legibility Levels



Accuracy

The following data describes the ply and character accuracy metrics respectively for each provider.

Ply Accuracy Per Provider



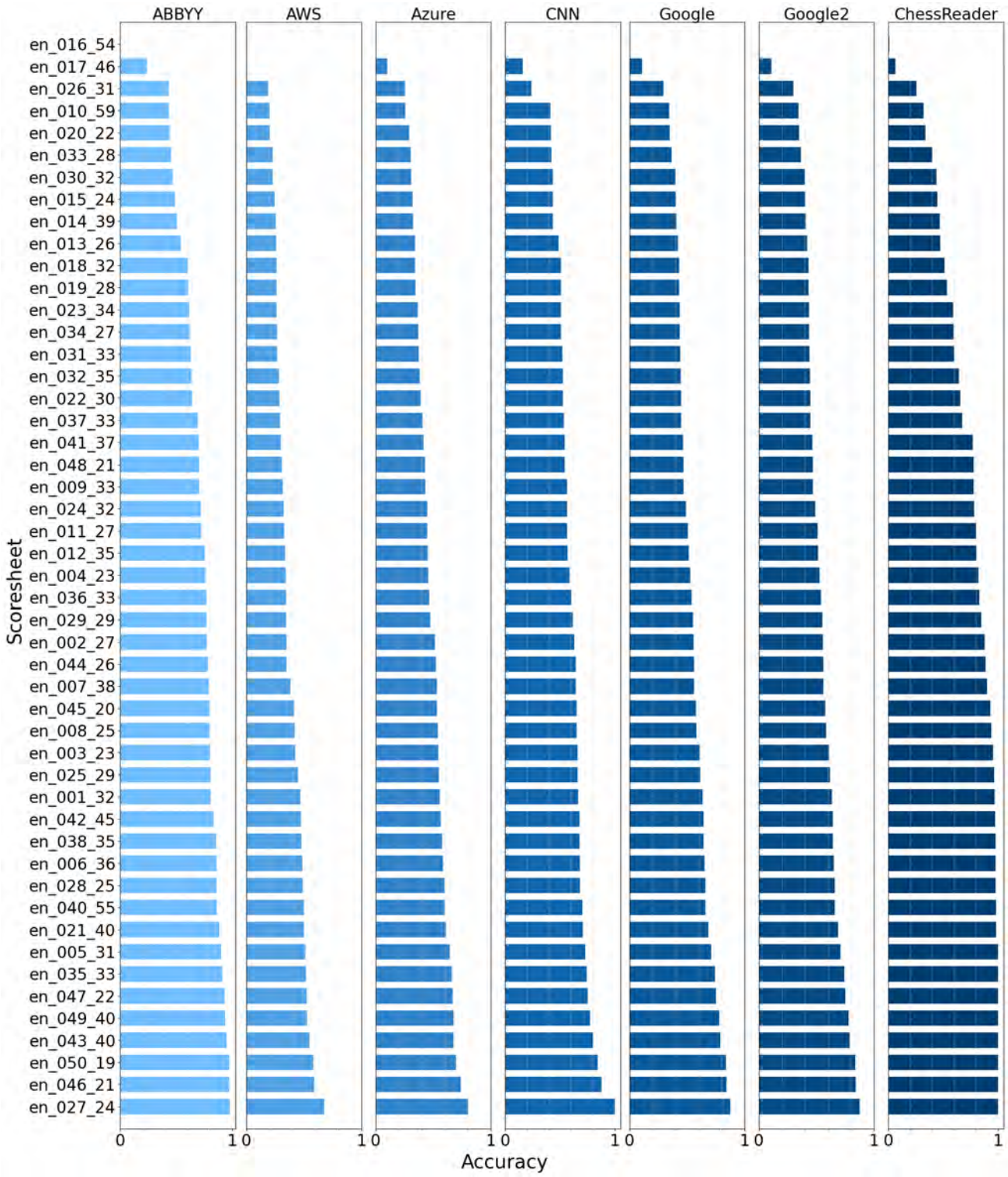
provider	total ply accuracy	total ply accuracy median	total ply accuracy std
ABBYY	70.9%	76.1%	19.7%
AWS	35.9%	33.9%	13.9%
Azure	46.6%	45.7%	15.8%
CNN	58.4%	58.7%	17.1%
Google	53.2%	52.5%	17.2%
Google2	53.2%	52.5%	17.2%
ChessReader	74.5%	82.3%	26.3%

Outliers Of Ply Accuracy From ChessReader Over Corpus

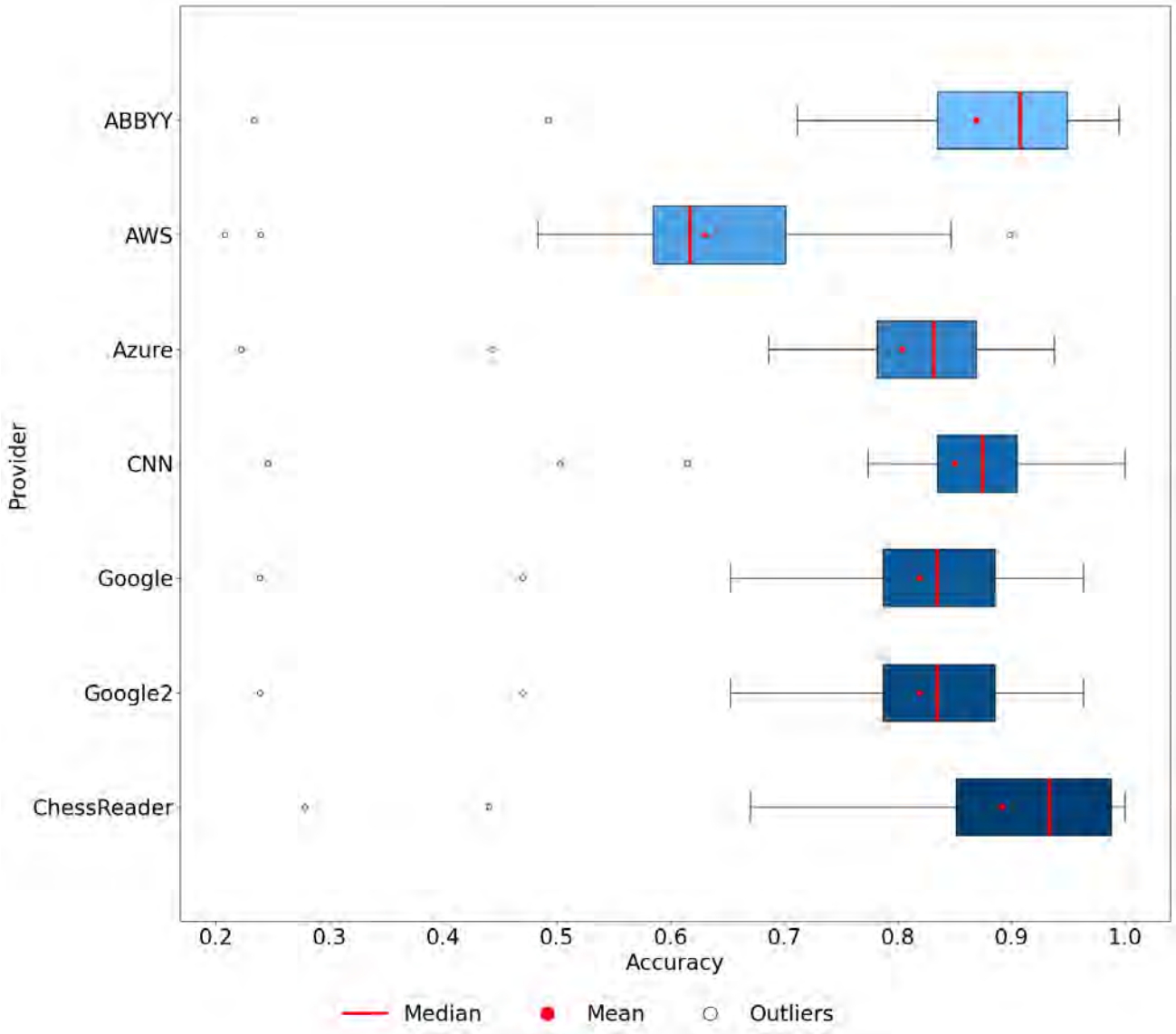
The following list contains information about the outliers from the ChessReader prediction over the whole corpus, shown in the above boxplot.

- **english/en_016_54** - 0.9%

Ply Accuracy Per Scoresheet Per Provider



Character Accuracy Per Provider



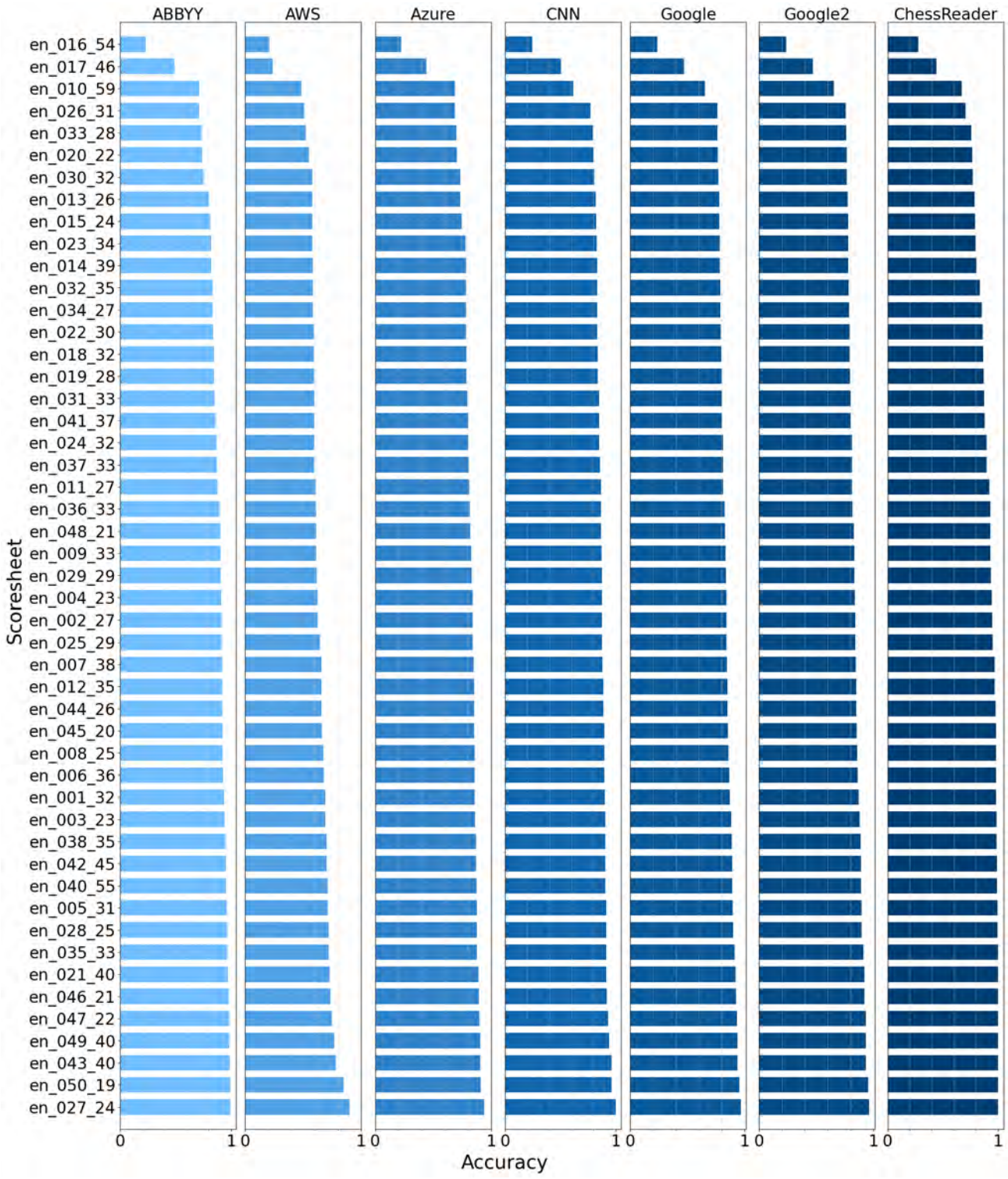
provider	total character accuracy	total character accuracy median	total character accuracy std
ABBYY	86.9%	90.8%	13.1%
AWS	63.0%	61.7%	11.6%
Azure	80.4%	83.2%	11.6%
CNN	85.0%	87.5%	11.7%
Google	81.9%	83.5%	11.6%
Google2	81.9%	83.5%	11.6%
ChessReader	89.2%	93.4%	14.1%

Outliers Of Character Accuracy From ChessReader Over Corpus

The following list contains information about the outliers from the ChessReader prediction over the whole corpus, shown in the above boxplot.

- **english/en_016_54** - 27.8%
- **english/en_017_46** - 44.0%

Character Accuracy Per Scoresheet Per Provider



Box Accuracy

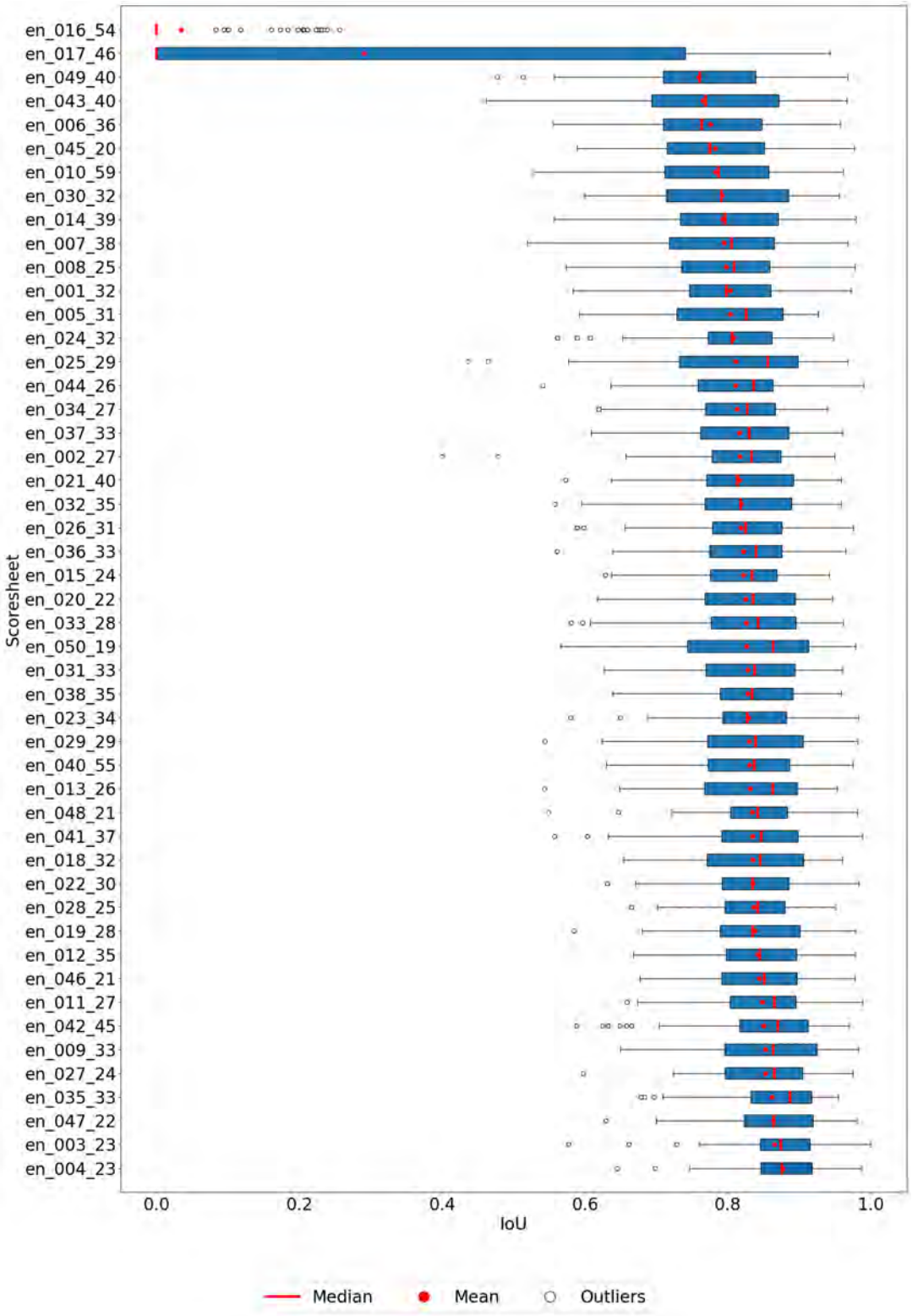
The following data describes the accuracy of the box predictions which is defined as the intersection over union of the box prediction and label.

The box accuracies were calculated using the following formula:

$$\text{IoU} = \frac{|Box_l \cap Box_p|}{|Box_l \cup Box_p|}$$

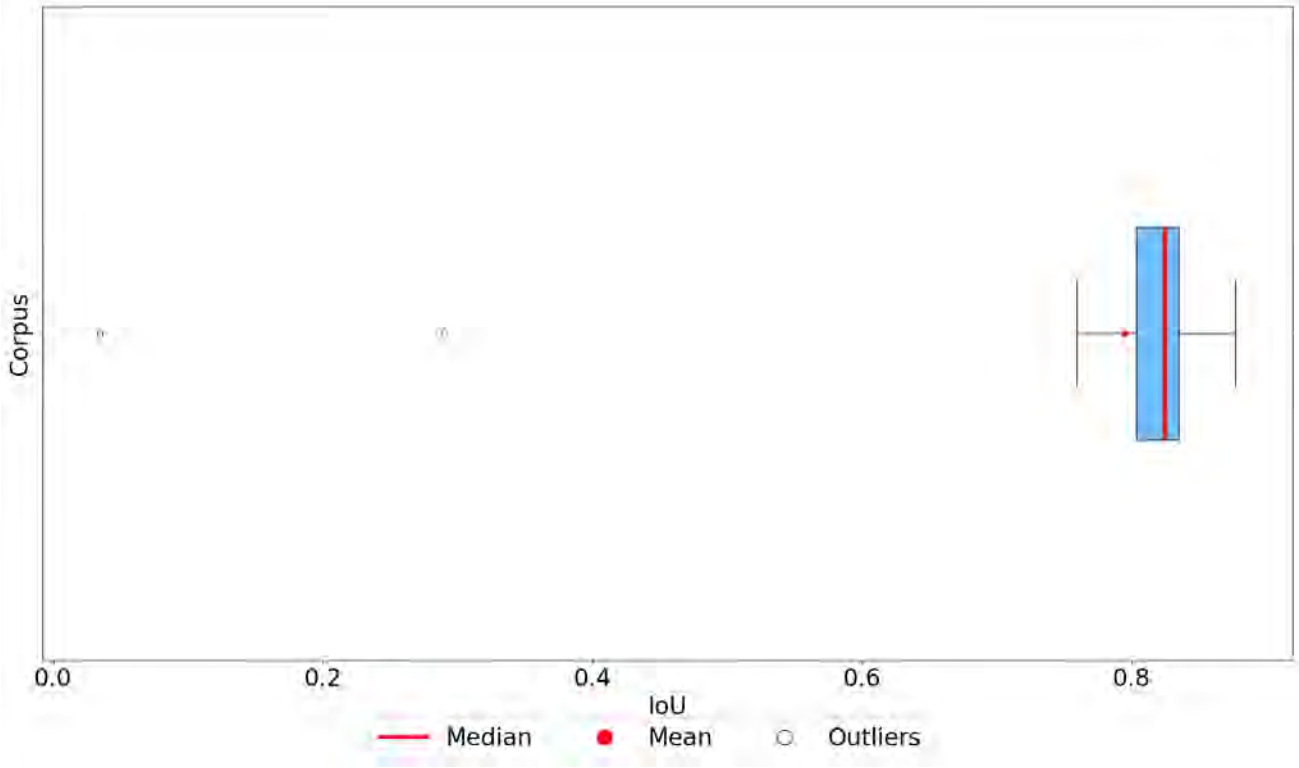
$$Box_l = \text{The box label from the corpus}$$
$$Box_p = \text{The box prediction from ChessReader}$$

Box Accuracy Per Scoresheet



Box Accuracy Over Corpus

Intersection over union average over entire corpus: **79.5%**



Outliers Of Box Accuracy From ChessReader Over Corpus

The following list contains information about the outliers from the ChessReader prediction over the whole corpus, shown in the above boxplot.

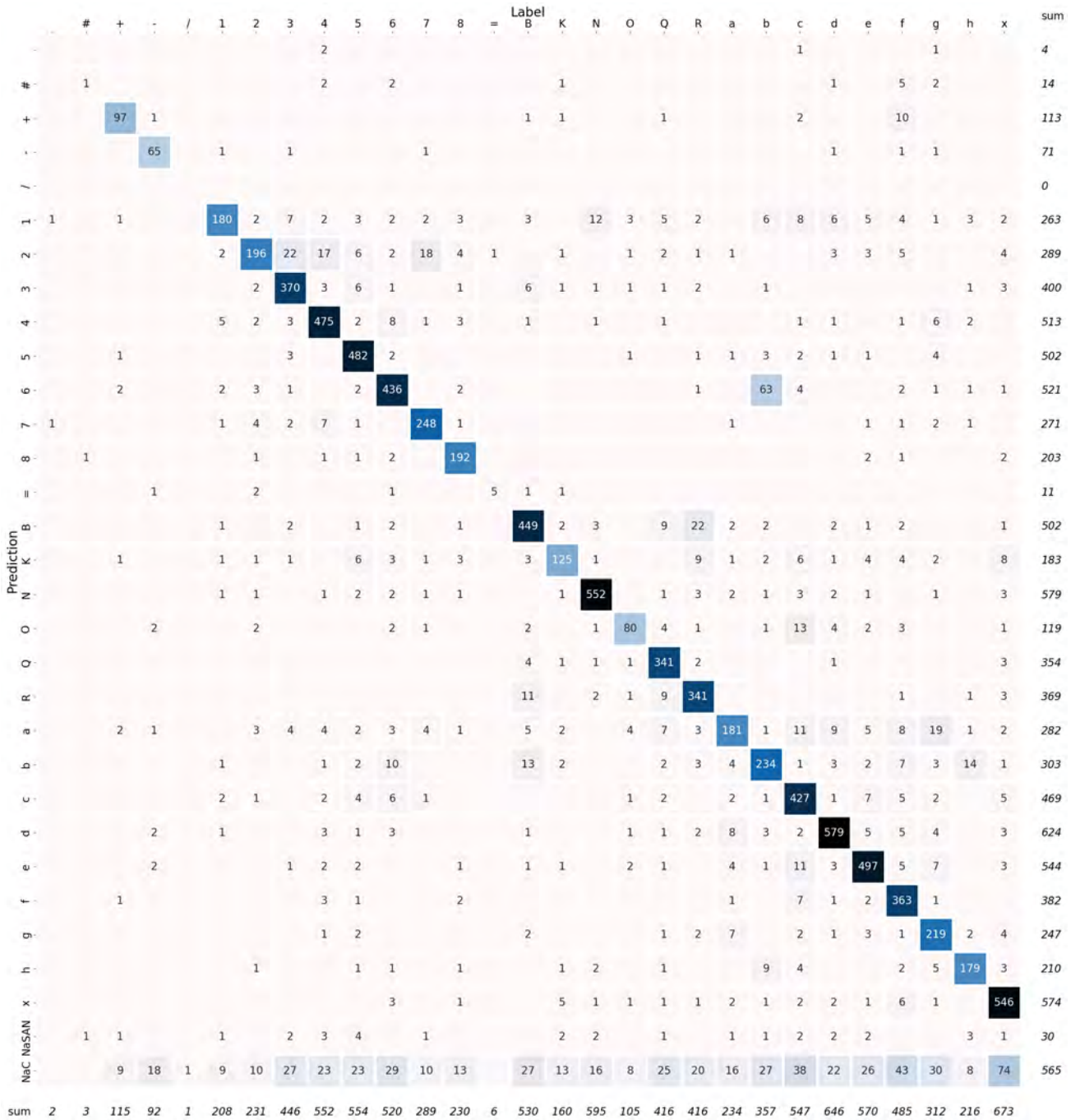
- **english/en_016_54** - 3.4%
- **english/en_017_46** - 29.1%

Character Confusion

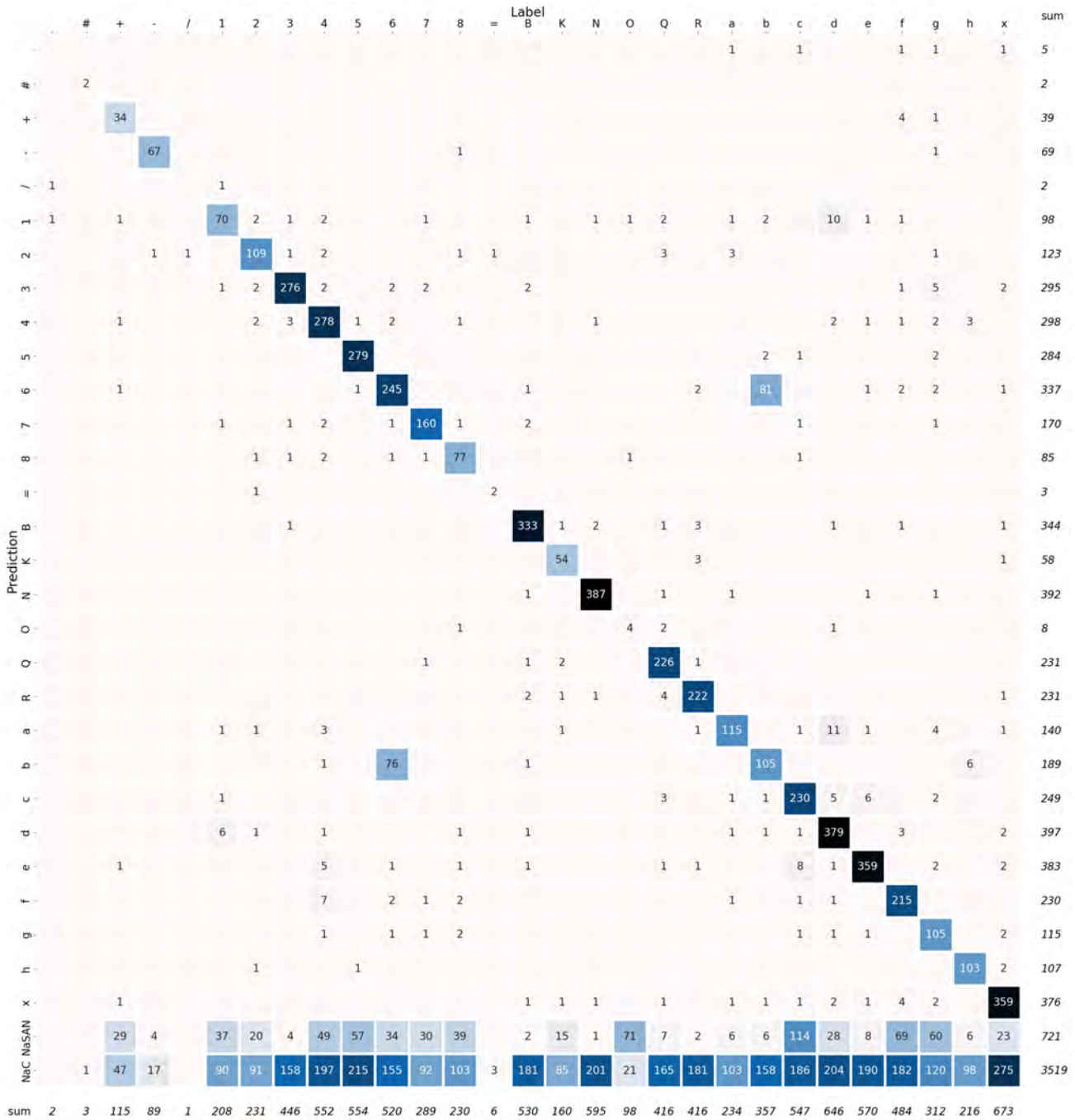
The following data shows the character confusion matrices for each provider.

The field **NaC (Not a Character)** describes the cases where the label character was not recognized at all.
The field **NaSAN (Not a SAN)** describes the cases where the label character was recognized as something, which is not part of the SAN character set.

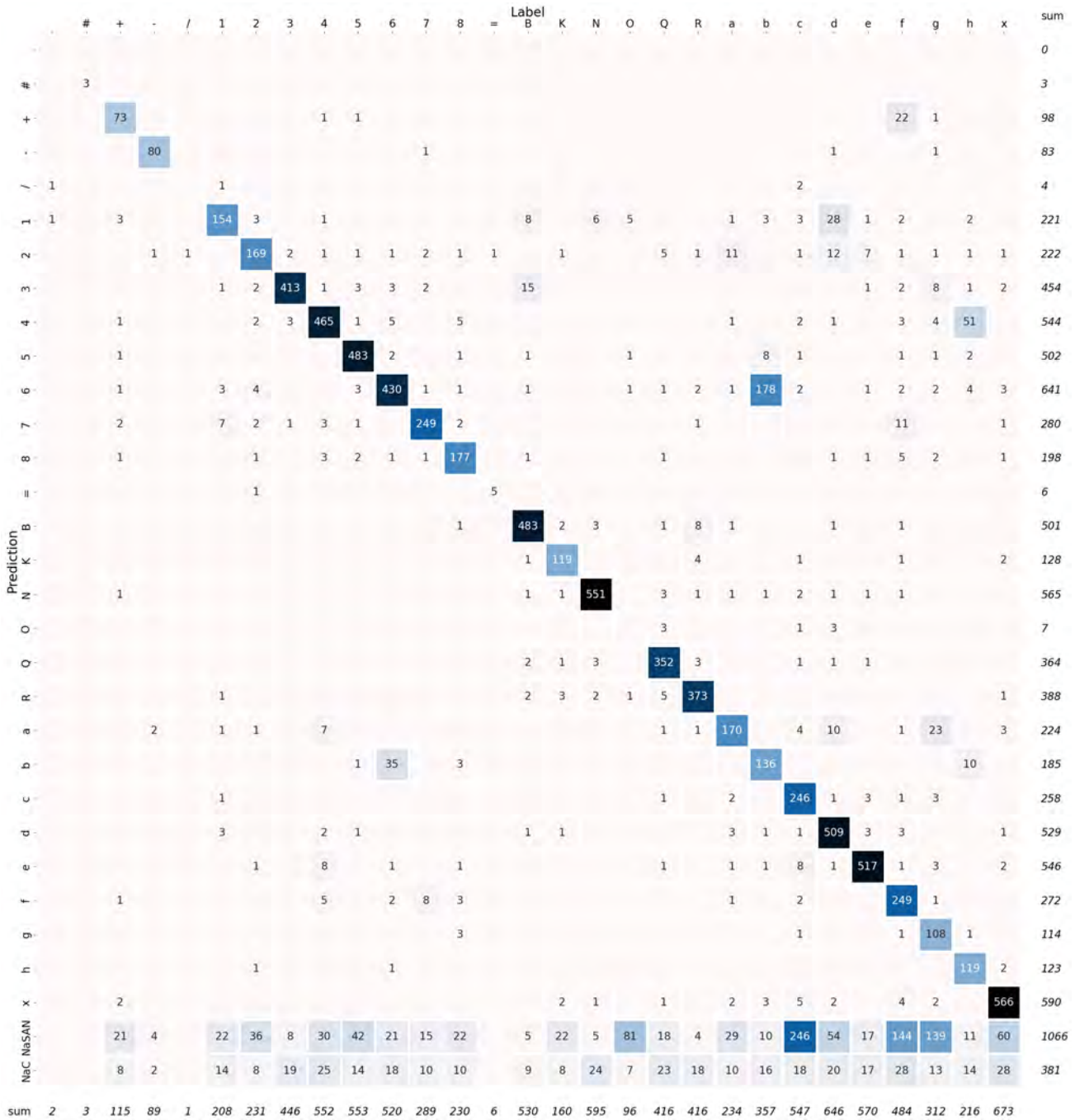
ABBYY Character Confusions



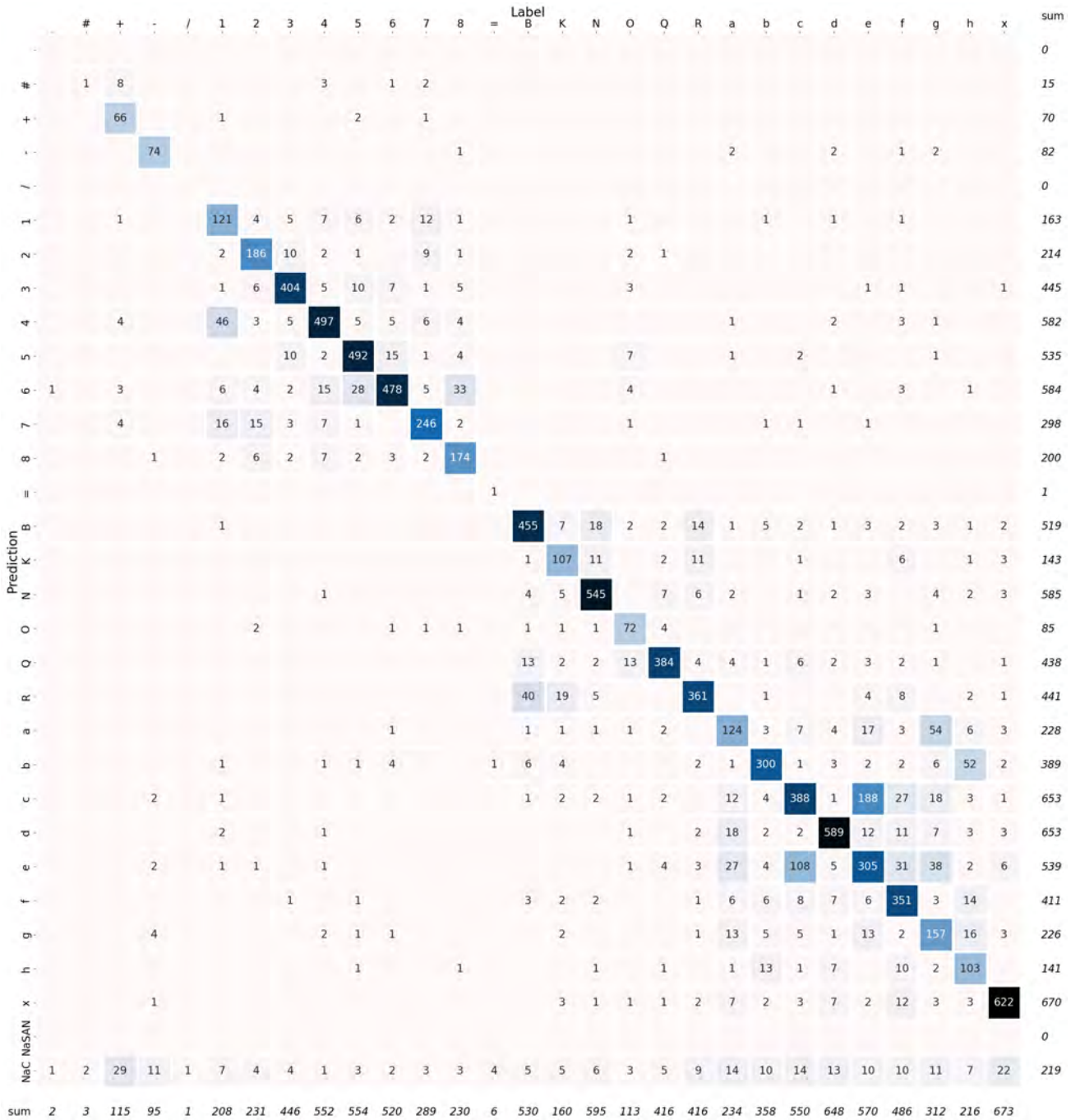
AWS Character Confusions



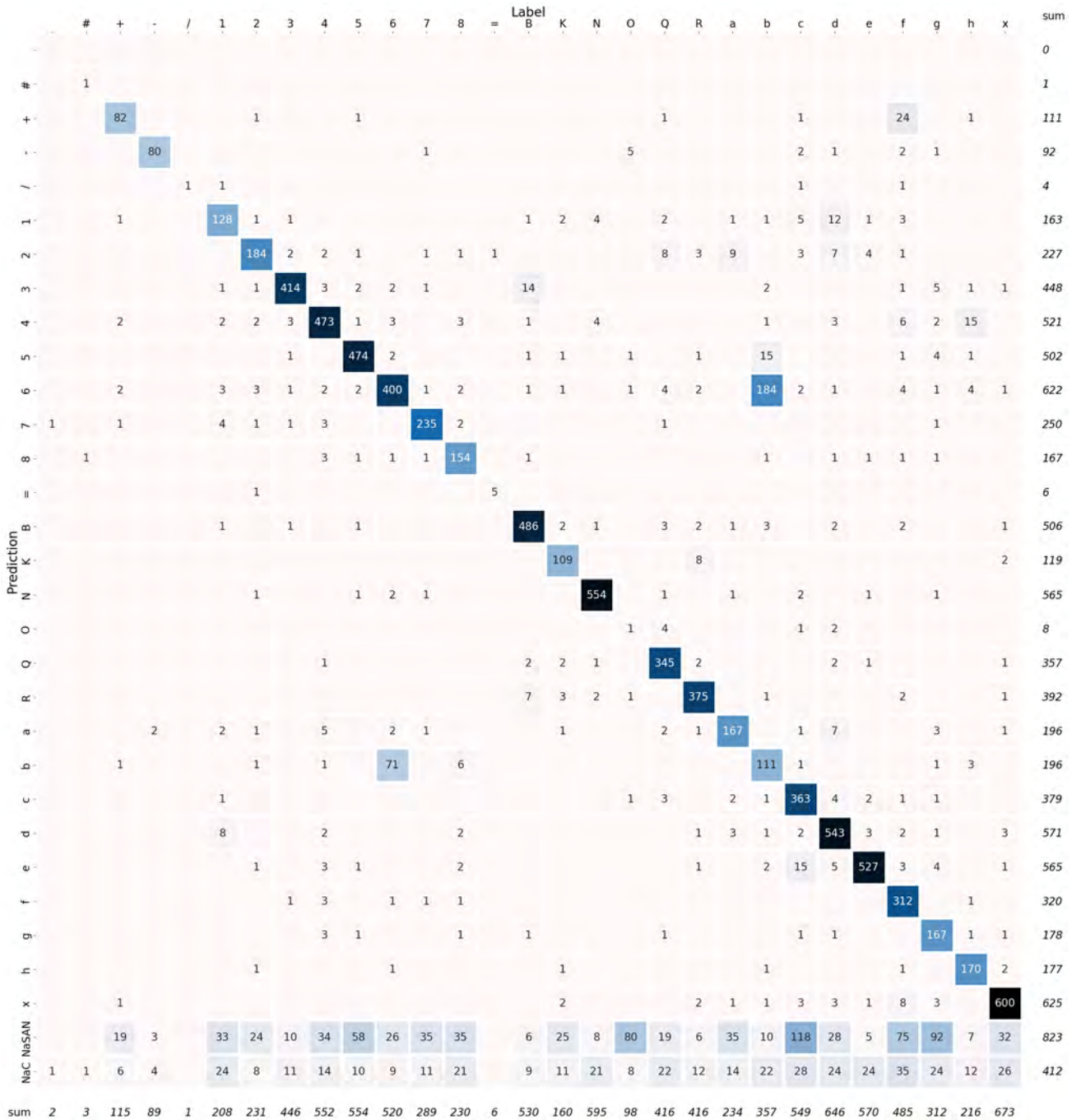
Azure Character Confusions



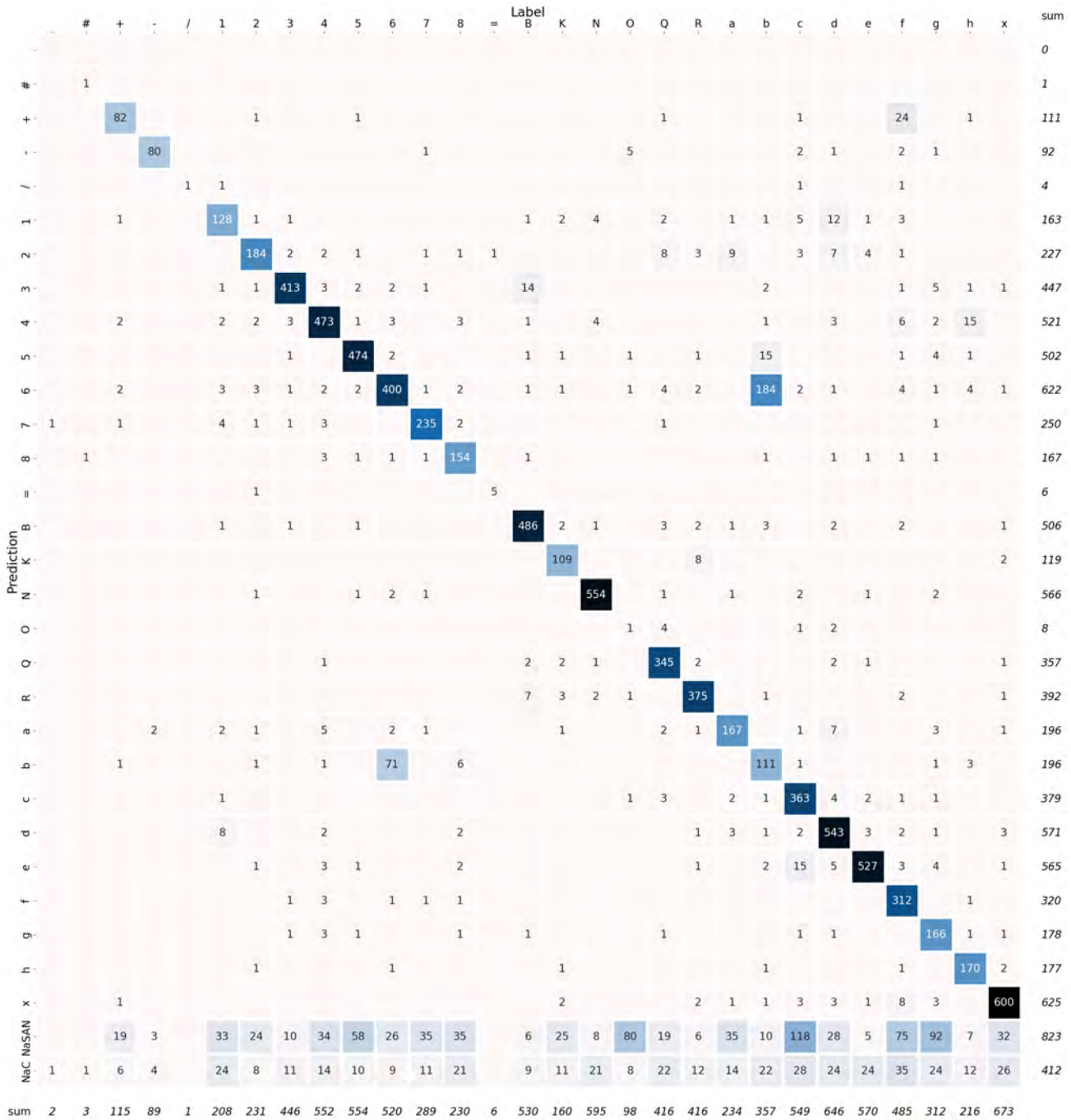
Cnn Character Confusions



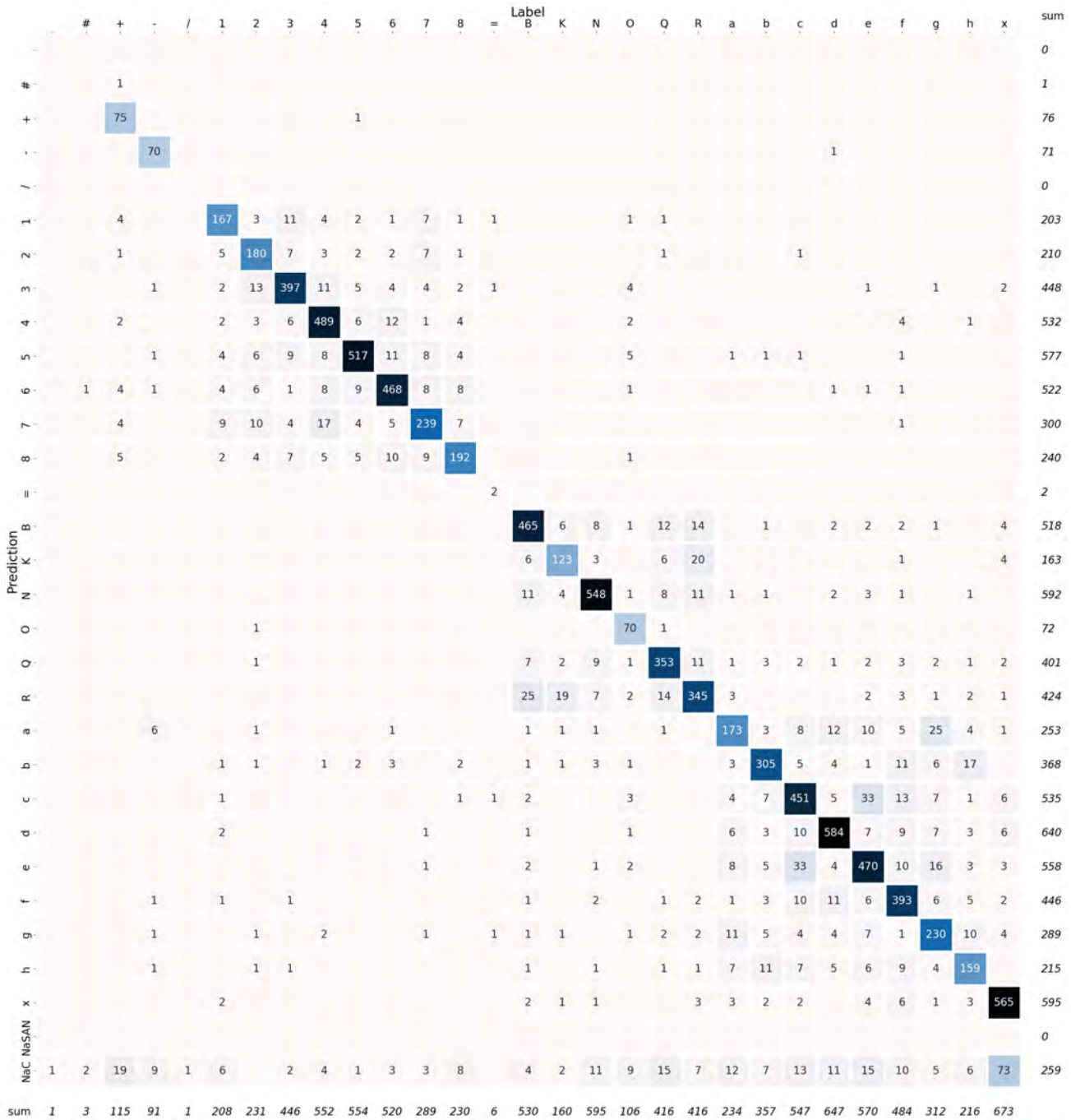
Google Character Confusions



Google2 Character Confusions



Chessreader Character Confusions



Confidence Confusion

The following data shows a confusion matrix for the predicted ply confidence. If the confidence of the ply is 0.9 or higher, it is marked as "High Confidence". When lower it is marked as "Low Confidence".

The confidence confusion was calculated using the following formulas:

$$
\begin{aligned}
TP &= \text{confidence} > \text{threshold, prediction is correct} \\
FP &= \text{confidence} > \text{threshold, prediction is wrong} \\
TN &= \text{confidence} < \text{threshold, prediction is wrong} \\
FN &= \text{confidence} < \text{threshold, prediction is correct}
\end{aligned}
$$

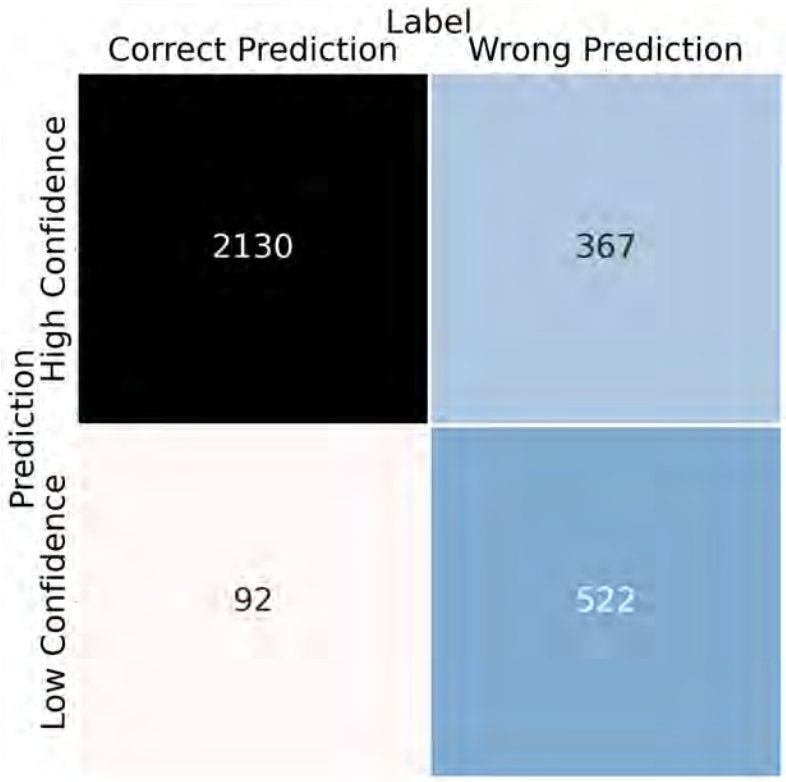
$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$f1 = 2 * \frac{precision \cdot recall}{precision + recall}$$

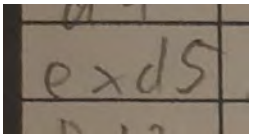
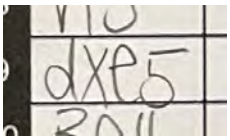
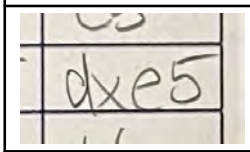
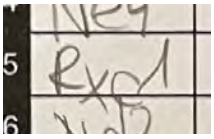
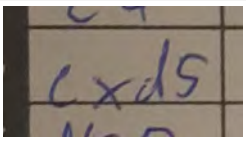
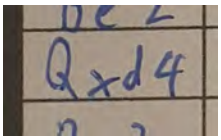
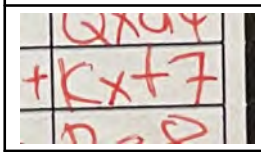
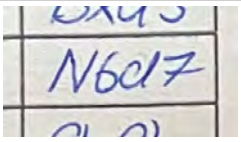
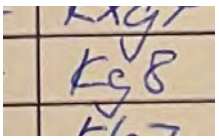
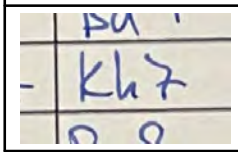
Confidence Confusion



accuracy	precision	recall	f1
85.2%	85.3%	95.9%	90.3%

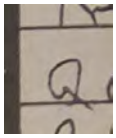
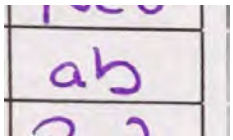
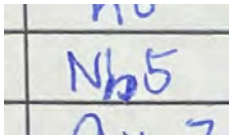
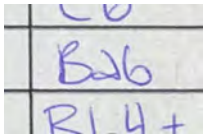
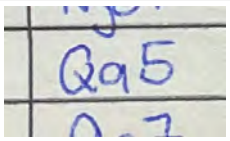
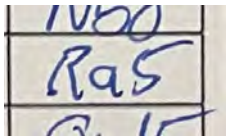
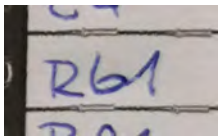
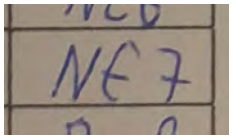
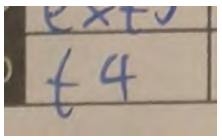
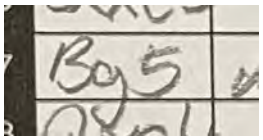
Top False Positive Results

The following table shows the predictions with the highest confidence, that are wrong.

ply box image	scoresheet	ply index	prediction	label	confidence
	english/en_019_28	4	cxd5	exd5	100%
	english/en_031_33	16	dxc5	dxe5	100%
	english/en_031_33	17	dxc5	dxe5	100%
	english/en_030_32	48	Kd2	Rxc1	100%
	english/en_012_35	10	exd5	cxd5	100%
	english/en_017_46	12	Qxd4	Be2	100%
	english/en_034_27	49	Kh8	Kxf7	100%
	english/en_007_38	17	Nbd7	N6d7	100%
	english/en_007_38	73	Re8	Kg8	100%
	english/en_038_35	43	Kg8	Kh7	100%

Top False Negative Results

The following table shows the predictions with the lowest confidence, that are correct.

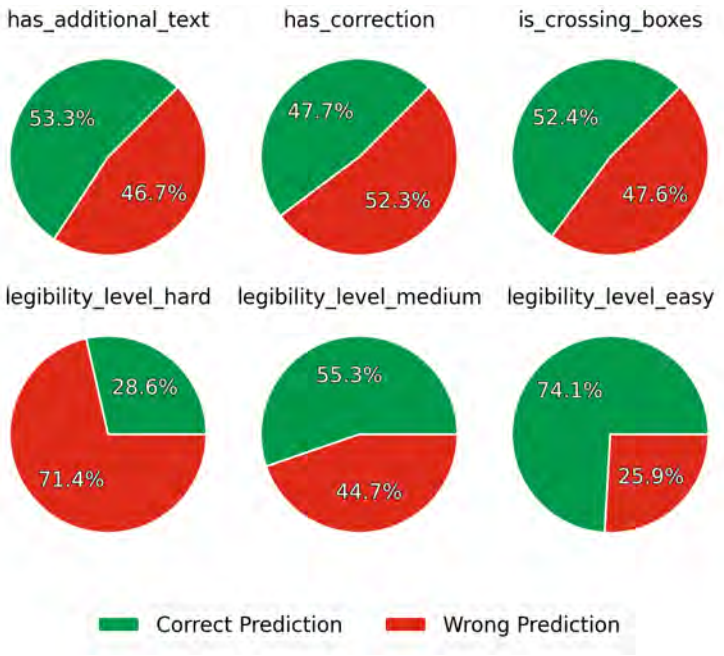
ply box image	scoresheet	ply index	prediction	label	confidence
	english/en_016_54	26	Qc2	Qc2	37%
	english/en_048_21	5	a6	a6	38%
	english/en_003_23	17	Nh5	Nh5	43%
	english/en_029_29	7	Ba6	Ba6	44%
	english/en_002_27	19	Qa5	Qa5	50%
	english/en_022_30	45	Ra5	Ra5	51%
	english/en_040_55	98	Rb1	Rb1	52%
	english/en_012_35	21	Nf7	Nf7	53%
	english/en_017_46	38	f4	f4	53%
	english/en_026_31	12	Bg5	Bg5	54%

Handwriting

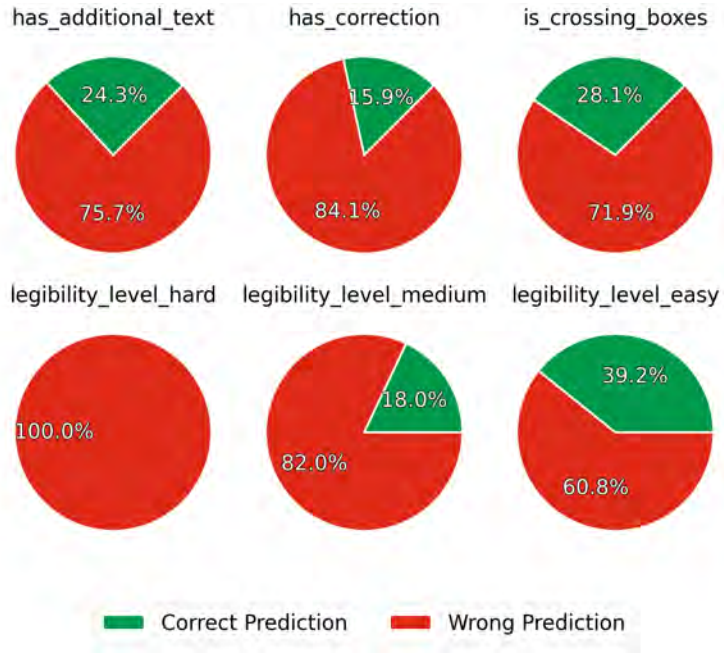
The following data shows how affected each provider is by the different handwriting properties. An explanation for each property can be found in the Corpus chapter.

The diagrams show the percentage of how many of the cells with that property are correctly predicted by the provider. 100% represents all plies/cells that have that property are correctly predicted.
If a property does not occur in the corpus the corresponding diagram is grayed out.

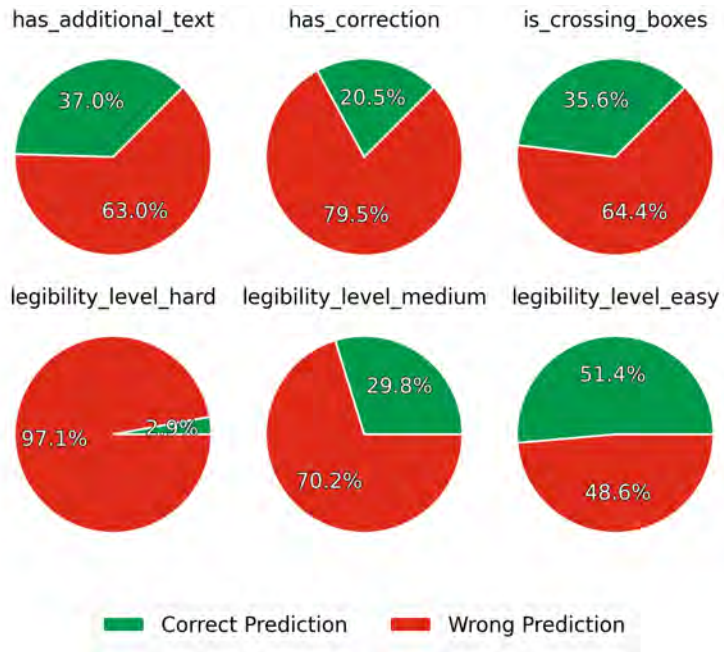
Handwriting Property Recognition For ABBYY Provider



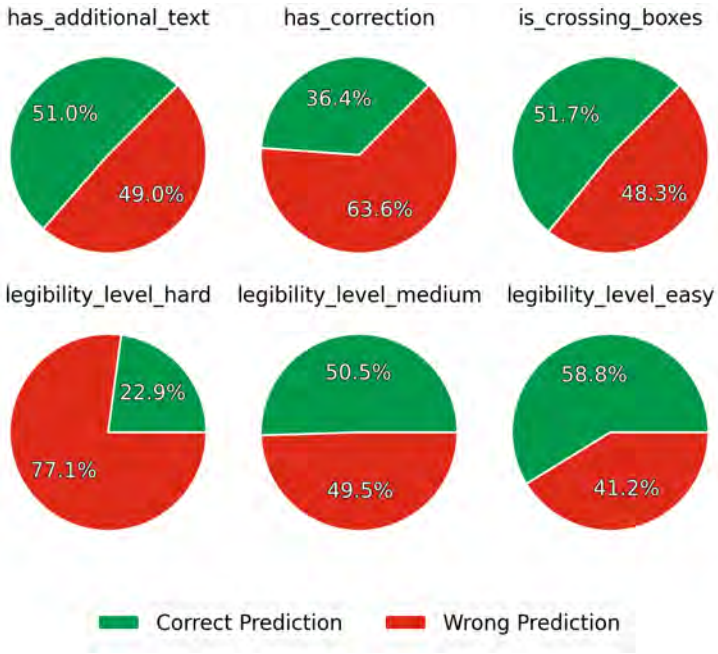
Handwriting Property Recognition For AWS Provider



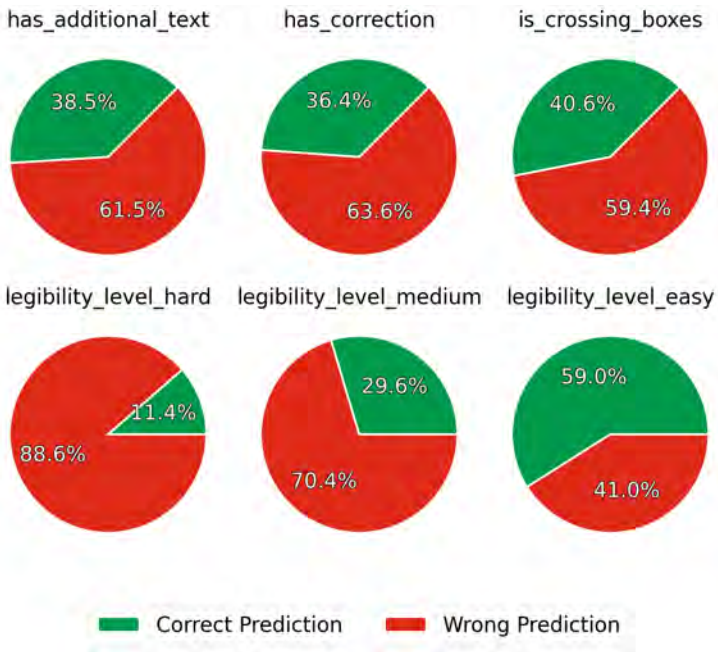
Handwriting Property Recognition For Azure Provider



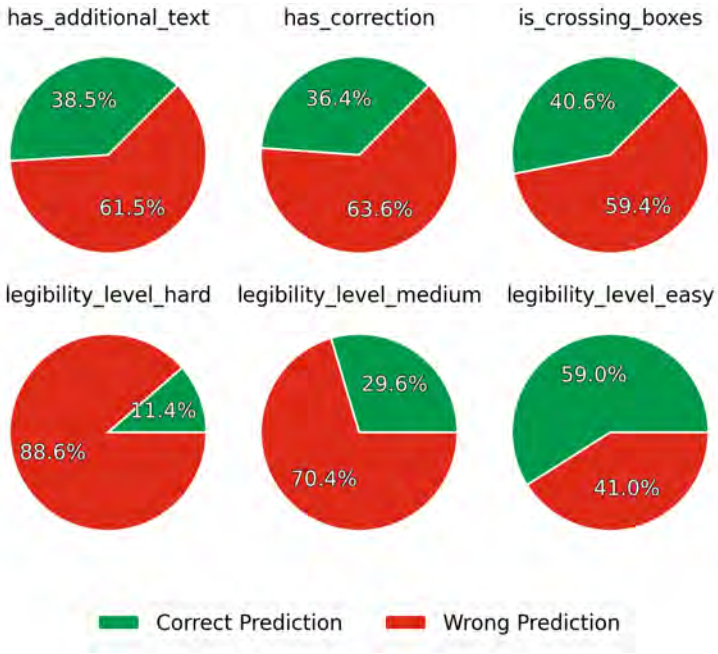
Handwriting Property Recognition For Cnn Provider



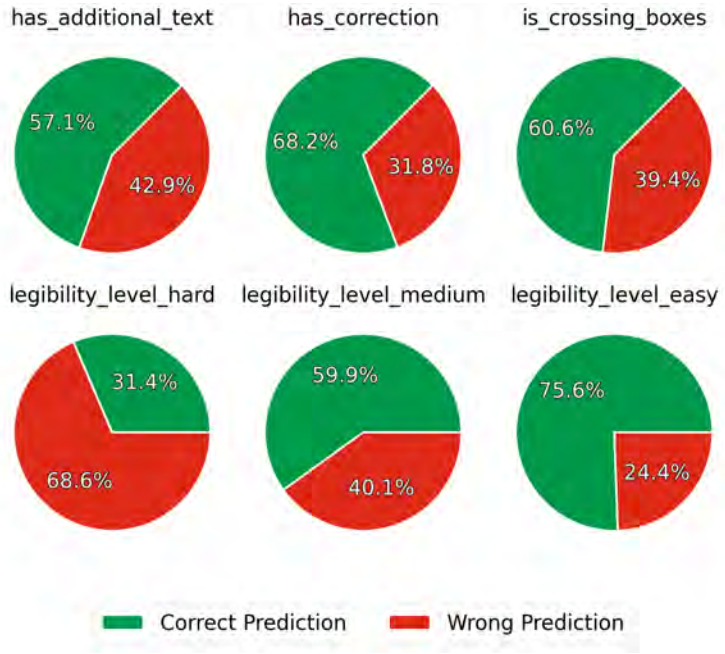
Handwriting Property Recognition For Google Provider



Handwriting Property Recognition For Google2 Provider



Handwriting Property Recognition For Chessreader Provider

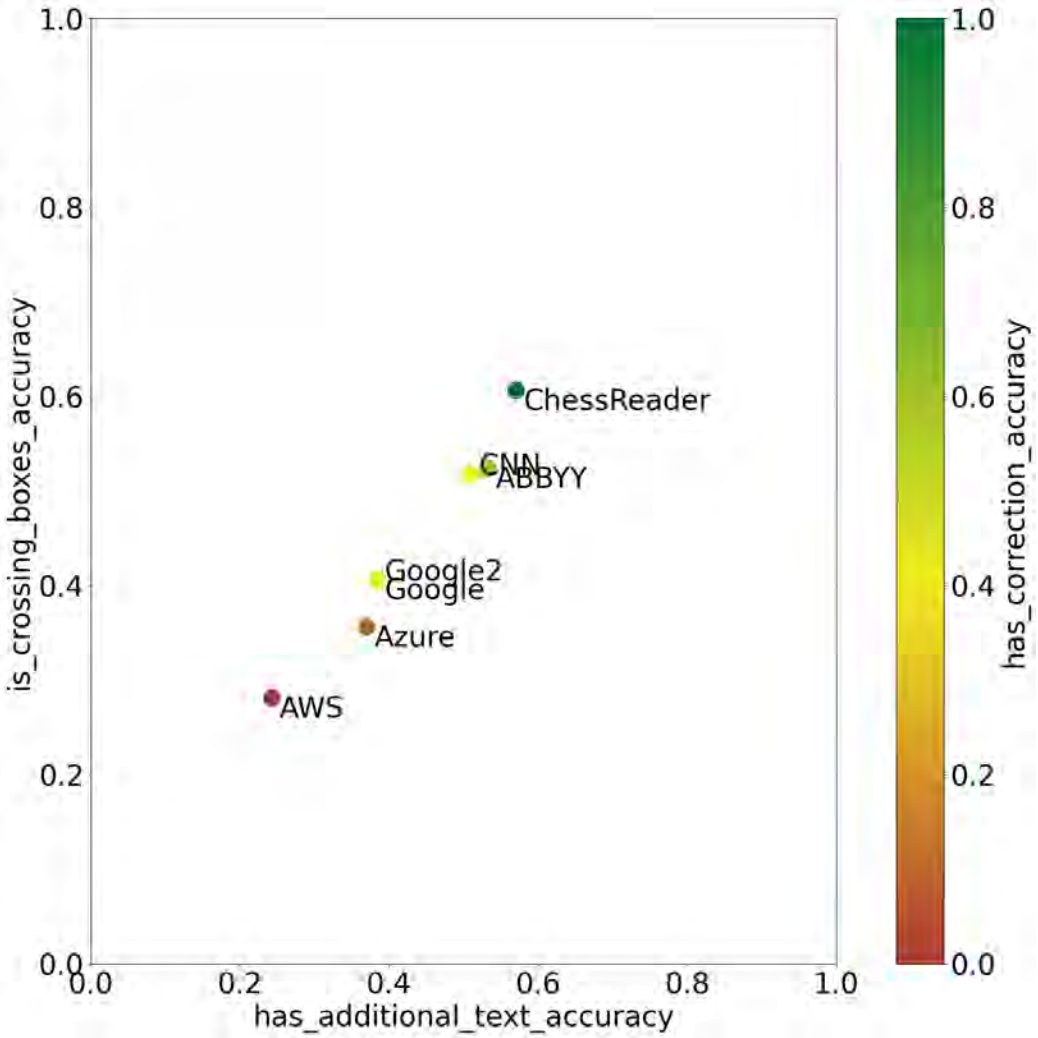


The following tables the shows percentage of the properties that were correctly predicted. 100% represents all plies/cells that have that property are correctly predicted.

index	has additional text	has correction	is crossing boxes	legibility level hard	legibility level medium	legibility level easy
ABBYY	53.3%	47.7%	52.4%	28.6%	55.3%	74.1%
AWS	24.3%	15.9%	28.1%	0.0%	18.0%	39.2%
Azure	37.0%	20.5%	35.6%	2.9%	29.8%	51.4%
CNN	51.0%	36.4%	51.7%	22.9%	50.5%	58.8%
Google	38.5%	36.4%	40.6%	11.4%	29.6%	59.0%
Google2	38.5%	36.4%	40.6%	11.4%	29.6%	59.0%
ChessReader	57.1%	68.2%	60.6%	31.4%	59.9%	75.6%

Handwriting Property Recognition Provider Comparison

The following diagram shows a combination of properties and their accuracies for different providers. The three properties are "is_crossing_boxes", "has_additional_text", and "has_correction". The percentages represent the proportion of correct predictions relative to all labels with these properties for each provider. Perfect performance on all properties is indicated by a dark green point located in the top right. Points not being located on the diagonal with a corresponding color indicates a provider having a strength/weakness in predicting plies with the related property.



B Anwendungsanleitung der Evaluierungspipeline

Es folgt ein Ausschnitt aus der README Datei vom ChessReader API Github Repository[18] in welchem die Installation und Anwendung der Evaluierungspipeline beschrieben wird.

[18] https://github.com/spinningbytes/ChessReader_API

Evaluating ChessReader

ChessReader has the capability to generate a PDF report which evaluates its scoresheet processing capabilities and the used OCR engines based on a few extendable metrics. The code for evaluating ChessReader can be found on the ChessReader_API GitHub repository in the `evaluate` folder.

Back-end setup

Evaluation database setup

It is recommended to run the evaluation in a separate database, as a lot of matches are being automatically generated. This can be done in MySQL Workbench the following ways:

After login into the server, create a database (which in this program is synonymous with Schema). You do this by going to the left hand window and selecting "Schemas" at the bottom of the upper part. Then right-click anywhere in the new "window" and click on "Create Schema...".

Alternatively you can click on "File" at the top of the window and select "New Query Tab". In the new Window, write `CREATE SCHEMA <database-name>;` and press `Ctrl + Enter`.

Environment setup

Install the requirements for the evaluation pipeline:

```
pip install -r evaluate/requirements.txt
```

Edit the file called `.env.evaluate` and write into it the following variables

```
#Format: mysql://user:password@ip:port/database
# Replace chessreader_eval with your evaluation database name
DATABASE=mysql://root:password@127.0.0.1:3306/chessreader_eval
#Replace ... with any arbitrary string
JWT_SECRET_KEY=...
BASE_URL=http://127.0.0.1:5000
FLASK_DEBUG=development
```

Starting the back-end in evaluation mode

For the evaluation, the back-end needs to run. For the backend to use the evaluation database it can be started in the following way:

```
python app.py evaluate
```

Running an evaluation

Download the corpus

The Chess-Scoresheet corpus the evaluation runs on can be found the public Dropbox. It is important that this corpus is used, as the evaluation pipeline depends on the data to be formatted in a specific way.

Configuring the evaluation pipeline

In the `evaluate/config.py` file a few configurations need to be made:

```
# Set this to the path the corpus was downloaded to
CORPUS_PATH = 'Path/to/Corpus'
# Defines what languages from the corpus should be evaluated.
CORPUS_ALLOWED_LANGUAGES = ["english"]

# Deactivate metrics by setting the corresponding boolean to false. The
metric will not be contained in the report
METRIC_EVALUATION = {
    METRICS.PLY_ACCURACY: True,
    METRICS.CHARACTER_ACCURACY: True,
    METRICS.CONFIDENCE_CONFUSION: True,
    METRICS.CORPUS_PROPERTIES: True,
    METRICS.HANDWRITING_ACCURACY: True,
    METRICS.CHARACTER_CONFUSION: True,
    METRICS.BOX_ACCURACY: True,
    METRICS.PLY_STD: True,
    METRICS.CHARACTER_STD: True,
    METRICS.PLY_MEDIAN : True,
    METRICS.CHARACTER_MEDIAN : True,
}
```

Running the pipeline

The evaluation pipeline can be run by executing

```
python evaluate/evaluate.py
```

ATTENTION: running the pipeline with a corpus of more that 50 scoresheets may take 40 minutes+ as ChessReader needs to generate results for each sheet first.

While running, it will display what action is currently being executed:

```
Importing ChessReader_Corpus/Corpus_v5/english/en_019_28/en_019_28.png...
.
.
Calculating results for ChessReader_Corpus/Corpus_v5/english/en_019_28...
.
```

```
.
Cummulating results for METRICS.BOX_ACCURACY...
Cummulating results for METRICS.PLY_STD...
.
.
Generating diagramm for METRICS.PLY_ACCURACY...
Generating diagramm for METRICS.HANDWRITING_ACCURACY...
.
.
Generating output report...
DONE!
Result JSON file is stored at evaluate/output/results.json
Report is stored at
evaluate/output/chessreader_evaluation_report_20240605_0018.pdf
```

Output Formats

The output of the pipeline can be found under `evaluate/output`

- `file_matchid_mapping.json` contains the mapping between the scoresheets and the corresponding `match_id` from ChessReader. If this file already exists the corpus is not imported again and the existing `match_id`s are used.
- `results.json` contains all results the pipeline generated in text form.
- `diagrams` contains all generated diagrams
- `chessreader_evaluation_report_timestamp.pdf` is the final report, which contains the diagrams, metric numbers and explanations of the run metrics.

C Swagger Dokumentation

Es folgt eine Auflistung aller ChessReader Endpunkte aus der Swagger Dokumentation. Hier werden nur die zusammengeklappten Endpunkte dargestellt. Eine volle Auflistung kann in einer laufenden ChessReader Backend Instanz folgendermassen geöffnet werden:

- Clonen des ChessReader API Repositories von GitHub und starten des Backends:

```
$ git clone https://github.com/spinningbytes/ChessReader_API.git
$ cd ChessReader_API/
$ python app.py
```

- Öffnen der URL `http://localhost:5000/swagger/`

/static/spec.json Explore

Chessreader API 2.0.0

[Base URL: localhost:5000/]
/static/spec.json

Schemes
HTTP

Authorize

Authorization Endpoints related to user authentication

POST /api/auth/signup Sign Up

POST /api/auth/login Login

Matches Endpoints related to matches

GET /api/matches Get Matches

POST /api/matches Create Match

GET /api/matches/{match_id} Get Match by ID

DELETE /api/matches/{match_id} Delete Match

GET /api/matches/{match_id}/image Get Match Image

GET /api/matches/{match_id}/boxes Get Move Boxes

PUT /api/matches/{match_id}/boxes Update Move Boxes

GET /api/matches/{match_id}/pgn Get Match PGN

GET `/api/matches/{match_id}/moves` Get Match Moves

GET `/api/matches/{match_id}/metadata` Get Match Metadata

POST `/api/matches/{match_id}/metadata` Add Match Metadata

GET `/api/matches/{match_id}/state` Get Match State

POST `/api/matches/{match_id}/state` Update Match State

GET `/api/matches/{match_id}/moves/confirm/single` Get Match Moves

POST `/api/matches/{match_id}/moves/confirm/single` Confirm / Update Single Move

POST `/api/matches/{match_id}/moves/confirm/multiple` Confirm Multiple Moves

GET `/api/matches/{match_id}/{move_index}/legalmoves` Get Legal Moves for a Move Index

Users Endpoints related to users

GET `/api/users/{user_id}` Get User by ID

POST `/api/users/{user_id}` Update User

GET `/api/users` Get Users

GET `/api/users/my/scancount` Get My Scan Count

Models

SignupRequest

LoginRequest

PutBoxRequest

PostMatchMetadataRequest

PostMatchStateRequest

PostConfirmSingleRequest

PostConfirmMultipleRequest

PostUserRequest

D ChessReader Datenbank

Es folgt ein Vergleich der ChessReader Datenbank vor und nach der Bachelorarbeit. Abbildung 60 zeigt den Aufbau der Datenbank vor Beginn der Bachelorarbeit und Abbildung 61 zeigt den neuen Datenbank Aufbau.

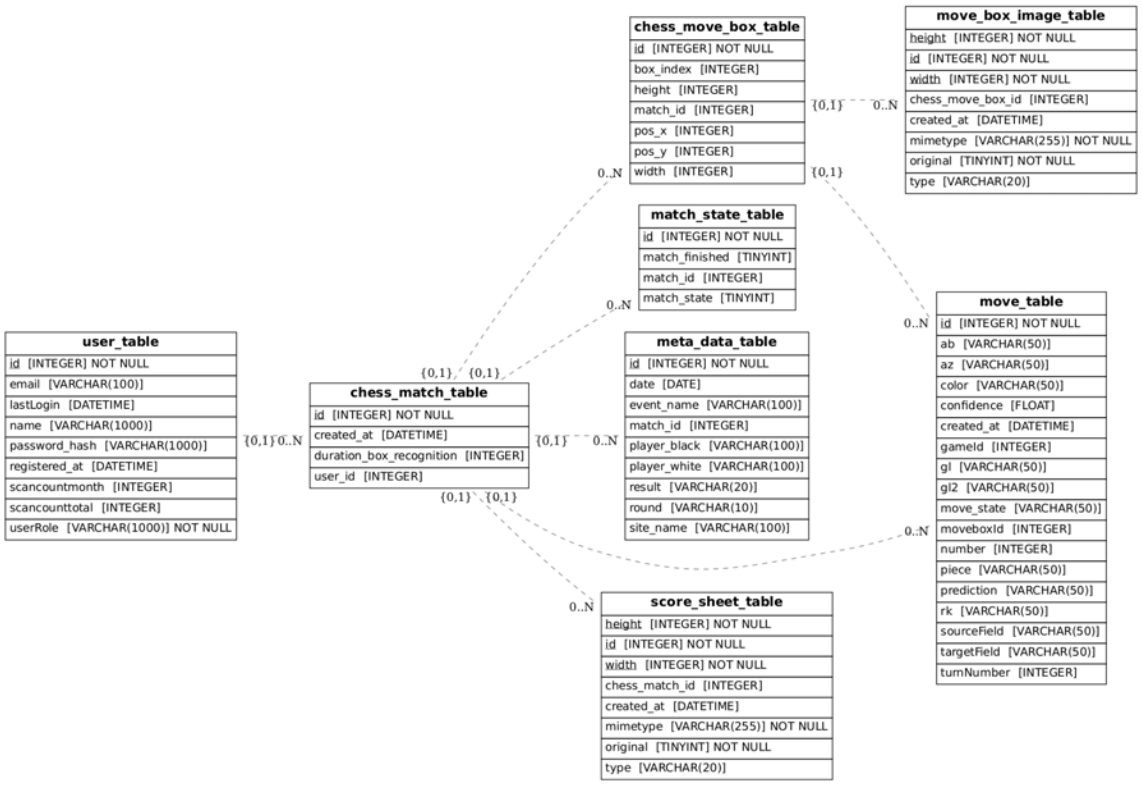


Abbildung 60: Datenbank Aufbau vor Beginn der Bachelorarbeit

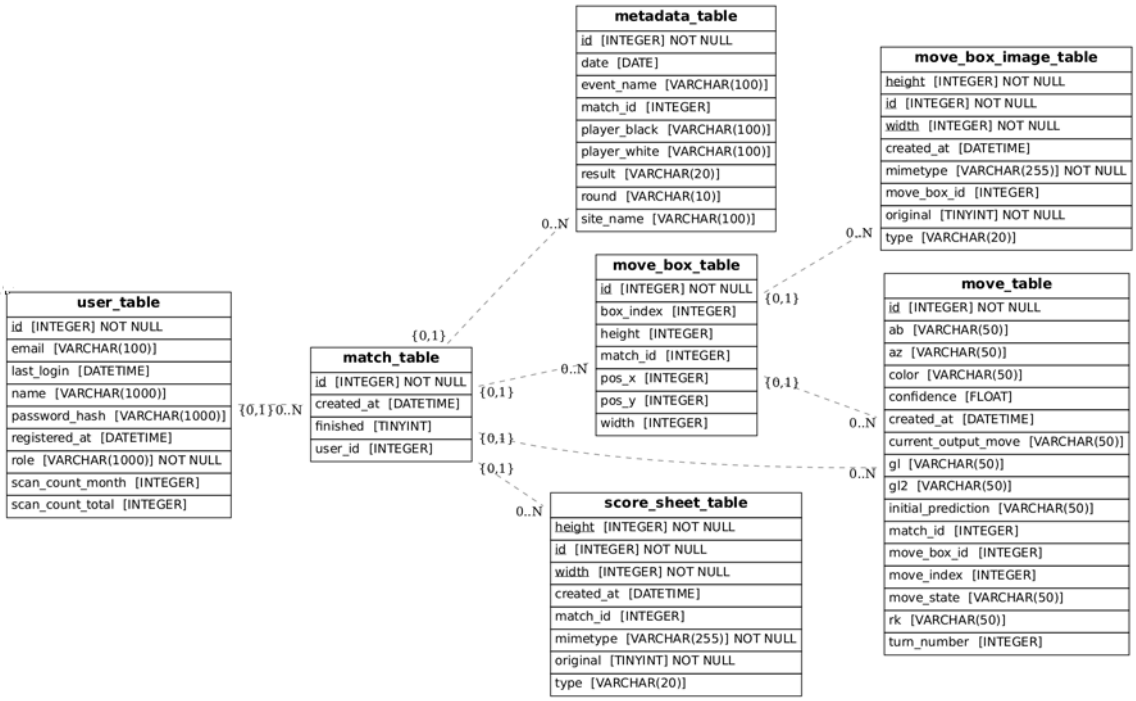


Abbildung 61: Datenbank Aufbau nach der Bachelorarbeit