School of Engineering

Centre for Artificial Intelligence

Bachelorarbeit Computer Science

Deep-Learning-basierte Erkennung handschriftlicher Schachnotationen im ChessReader-Projekt

Autoren		Nils Ersoy Nicola Schüepp
Hauptbetreuung		Prof. Dr. Mark Cieliebak
Nebenbetreuung		Pius von Däniken
Datum		05.06.2026

1 Zusammenfassung

Die Digitalisierung handschriftlich notierter Schachzüge stellt eine anspruchsvolle Aufgabe dar, da Schreibstile stark variieren und Zeichen der Standard Algebraic Notation (SAN) zum Teil nur feine visuelle Unterschiede aufweisen. Zugleich ist ihre automatisierte Erkennung für die Analyse von Partien von hoher praktischer Relevanz. Vor diesem Hintergrund untersucht diese Arbeit, wie ein zusätzliches, intern betreibbares OCR-Modell für die bestehende ChessReader-Software entwickelt werden kann, um die Abhängigkeit von externen Diensten zu reduzieren.

Ziel der Arbeit war die Entwicklung und Evaluation eines robusten Modells zur Erkennung handschriftlicher SAN-Züge mit hoher Erkennungsgenauigkeit. Hierfür wurden synthetisch erzeugte Trainingsdaten mit realen handschriftlichen Daten kombiniert. Die Evaluation erfolgte auf einem separaten, nicht für das Training verwendeten Test-Datensatz. Architektonisch wurden ein Transformer-Encoder-Decoder mit CNN-Backbone sowie eine Variante basierend auf CTC untersucht.

Die besten Ergebnisse erzielte das Encoder-Decoder-Modell mit einer Ply Accuracy von 84.35 % und einer Character Accuracy von 94.02 %. Das CTC-Modell zeigte mit 83.80 % Ply Accuracy und 94.07 % Character Accuracy eine vergleichbare Leistung. In den Experimenten führte insbesondere die Einbindung realer handschriftlicher Daten zu einer deutlichen Verbesserung von 19.80 % bei der Ply Accuracy. Zudem wurde für eine Modellvariante mit grösseren CNN-Filtern gegenüber der entsprechenden Vergleichsarchitektur eine weitere Verbesserung von 10.55 % erreicht.

Die Ergebnisse verdeutlichen, dass sowohl die Wahl einer geeigneten Modellarchitektur, als auch die Qualität und Repräsentativität der Trainingsdaten, entscheidend für die Erkennungsleistung sind. Gleichzeitig bleiben mehrdeutige oder schwer lesbare Handschriften eine klare Herausforderung. Insgesamt liefert die Arbeit eine tragfähige Grundlage für die Weiterentwicklung der handschriftlichen Schachzugerkennung im ChessReader-Projekt.

Schlüsselwörter: Schach, Optical Character Recognition, Transformer, CTC, Deep Learning

2 Abstract

Digitizing handwritten chess moves is a challenging task, as handwriting styles vary greatly and characters in Standard Algebraic Notation (SAN) sometimes differ only by subtle visual cues. At the same time, their automated recognition is of great practical importance for game analysis. Against this backdrop, this thesis investigates how an additional, internally run OCR model can be developed for the existing ChessReader software to reduce dependence on external services.

The goal of this work was to develop and evaluate a robust model for recognizing handwritten SAN strings with high recognition accuracy. To this end, synthetically generated training data was combined with real handwritten data. The evaluation was performed on a separate test dataset not used for training. Architecturally, a transformer encoder-decoder with a CNN backbone and a variant based on CTC were investigated.

The encoder-decoder model achieved the best results, with a ply accuracy of 84.35 % and a character accuracy of 94.02 %. The CTC model demonstrated comparable performance, with a ply accuracy of 83.80 % and a character accuracy of 94.07 %. In the experiments, the inclusion of real handwritten data in particular led to a significant improvement of 19.80 % in ply accuracy. In addition, a model variant with larger CNN filters achieved a further improvement of 10.55 % compared to the corresponding baseline architecture with smaller filters.

The results demonstrate that both the choice of a suitable model architecture and the quality and representativeness of the training data are crucial to recognition performance. At the same time, ambiguous or difficult-to-read handwriting remains a clear challenge. Overall, this thesis provides a solid foundation for the further development of handwritten chess move recognition in the ChessReader project.

Keywords: Chess, Optical Character Recognition, Transformer, CTC, Deep Learning

3 Vorwort

Wir beide (Nils & Nicola) kennen uns bereits seit der Primarschule. Vor vier Jahren haben wir uns unabhängig voneinander für das Informatik-Studium an der ZHAW entschieden. Bereits da war die Bachelorarbeit ein Thema, über das wir uns Gedanken gemacht haben. Wir sagten uns, wenn es ein Thema gibt, welches uns beide interessiert, dann werden wir die Bachelorarbeit gemeinsam schreiben. So kam es, dass wir beim Durchschauen der Ausschreibungen, uns beide für das ChessReader-Projekt interessiert haben. Nach einem Gespräch mit dem Verantwortlichen Prof. Dr. Mark Cieliebak einigten wir uns darauf, gemeinsam die Aufgabe anzunehmen.

Daher möchten wir an erster Stelle unserem Betreuer, Prof. Dr. Mark Cieliebak, für das Vertrauen, aber auch die Unterstützung während der Bachelorarbeit danken. Genauso bedanken möchten wir uns bei Pius von Däniken für den wöchentlichen Austausch und die wertvollen Hinweise, die er uns mit auf den Weg gegeben hat. Ein weiterer Dank geht an unsere beiden Arbeitgeber für die erhaltene Flexibilität und das Benutzen der Infrastruktur für gemeinsames Arbeiten. Zuletzt möchten wir uns noch bei unserem Umfeld bedanken, welches uns bei der Überarbeitung vom Bericht und auch sonst tatkräftig unterstützt hat.

Inhaltsverzeichnis

1 Zusammenfassung **2**

2 Abstract **3**

3 Vorwort **4**

4 Einleitung **9**

4.1 Ausgangslage 9

4.2 Zielsetzung 10

4.3 Projektrahmen 10

4.4 Bestehende Arbeiten 11

4.4.1 HCS-Datensatz und BiLSTM-Ansatz von Majid und Eicher 11

5 Theoretische Grundlagen **12**

5.1 Schach 12

5.1.1 Das Spiel 12

5.1.2 Partieformular (Scoresheet) 13

5.1.3 Standard Algebraic Notation (SAN) 14

5.2 Optical Character Recognition 15

5.3 Convolutional Neural Networks 16

5.3.1 Convolutional Layer 16

5.3.2 Pooling Layer 17

5.3.3 Fully Connected Layer 18

5.4 Transformer Model 18

5.4.1 Token, Embeddings und Positional Information 18

5.4.2 Attention und Self-Attention 18

5.4.3 Encoder und Decoder 19

5.4.4 Transformer in der Bilderkennung und OCR 20

5.4.5 Greedy Decoding und Beam Search 20

5.5 Connectionist Temporal Classification 21
5.5.1 Das Blank-Token 21
5.5.2 Decodierung: Collapse-Regel 21
5.5.3 Greedy Decoding und Beam Search 22
5.5.4 Vergleich mit Transformer-Decoder 22
5.6 Evaluationsmetriken 22

6 Vorgehen **24**
6.1 Analyse des bestehenden ChessReaders 24
6.2 Methodische Schwerpunkte 24
6.3 Datensätze 25
6.3.1 Einheitliche Bezeichnungen 25
6.3.2 Grundlage der synthetischen Datensätze 25
6.3.3 Synthetischer Basisdatensatz (*SYN-200K*) 26
6.3.4 Synthetischer Full-Cycles-Corpus (*SYN-FC*) 26
6.3.5 Synthetischer Weighted-Cycles-Corpus (*SYN-WC*) 26
6.3.6 HCS-Corpus (*HCS*) 27
6.3.7 ChessReader-Corpus v6 Englisch (*CR-v6-EN*) 27
6.4 Experimentelles Vorgehen 28

7 Modell **29**
7.1 Architekturentscheid: CNN-Backbone und Transformer-Encoder-Decoder 29
7.2 Architektur des Basismodells 29
7.2.1 CNN-Backbone 30
7.2.2 Height Collapse 30
7.2.3 Transformer-Encoder 31
7.2.4 Transformer-Decoder 31
7.2.5 Training-Setup 32
7.2.6 Gesamtarchitektur 32

8 Experimente **34**

8.1 Transformer-Encoder-Decoder 34
8.1.1 Baseline: Training auf *SYN-200K* 34
8.1.2 Wechsel zu *SYN-FC* 35
8.1.3 Mehr Encoder- und Decoder-Layers 38
8.1.4 Gewichteter Datensatz und Finetuning mit realen Daten 39
8.1.5 Erhöhung der Filtergrösse 43
8.1.6 Weitere Experimente ohne signifikante Verbesserung 45
8.2 CTC Decoding 46
8.2.1 Architektur-Änderungen CTC-Modell 46
8.2.2 Ergebnisse CTC-Modell 47
8.3 Untersuchung von Wichtigkeit der Datensätze 48
8.3.1 Finetuning mit verschiedenen Datenmengen 48
8.3.2 Pretraining auf realen Daten 50
8.4 Gesamtübersicht der Ergebnisse 52

9 Fehleranalyse 53

9.1 Greedy Decoding vs. Beam Search 53
9.2 Edit-Distance-Verteilung 54
9.2.1 Beam Search Ranking-Analyse 55
9.3 Character Confusion Matrix 56
9.4 Probleme der Validierungsdaten 59

10 Diskussion 61

10.1 Wichtigste experimentelle Erkenntnisse 61
10.2 Limitationen 61
10.3 Ausblick auf zukünftige Arbeiten 62
10.3.1 Überarbeitung des Fine-Tuning-Datensatzes 62
10.3.2 Fokus auf problematische Zeichen 62
10.3.3 Überarbeitung des Testdatensatzes 62
10.3.4 Implementierung von Beam Search 63

11 Fazit **64**

Glossar **65**

Literatur **66**

Abbildungsverzeichnis **68**

Tabellenverzeichnis **70**

A KI-Nutzungsdeklaration **71**

B Ablage der Endergebnisse **72**

C Evaluationsreport **73**

4 Einleitung

Gemäss den Regeln des internationalen Schachverbands müssen die gespielten Züge bei Schachturnieren auf einem Partieformular, dem sogenannten Scoresheet, notiert werden. Diese ermöglichen es, Partien nachträglich zu rekonstruieren und analysieren. Darüber hinaus kann das Scoresheet bei Unklarheiten oder Streitfällen als Beweismittel herangezogen werden. Die Schachzüge werden dabei in der Standard Algebraic Notation (SAN) notiert, einer kompakten aber eindeutigen Schreibweise [1].

Da Schachpartien meist in Persona gespielt werden, werden die Züge von Hand auf das Scoresheet geschrieben. Dies hat aber diverse Nachteile, beispielsweise ist die Aufbewahrung aufwendig und mit einem hohen Platzbedarf verbunden, ausserdem können die handschriftlichen Züge schwer lesbar sein. Zusätzlich können die gespielten Partien nicht einfach digital analysiert und nachgespielt werden.

Genau hier setzt das ChessReader-Projekt [2] an, welches eine Webapplikation bereitstellt, die es ermöglicht, jene handschriftlichen Züge von einem Scoresheet zu digitalisieren.

4.1 Ausgangslage

In früheren Bachelor- und Projektarbeiten wurde im Rahmen des ChessReader-Projekts ein Tool entwickelt, bei dem es möglich ist, Scoresheets hochzuladen und dieses dann versucht die Schachzüge zu erkennen und auszuwerten. Die Applikation besteht aus einem in Python entwickelten Backend und einem Frontend, das mit TypeScript und dem Angular-Framework umgesetzt wurde. Im Hintergrund läuft eine MySQL-Datenbank, in welcher die Scoresheets, deren Box-Informationen (Höhe, Breite, Position), sowie die verschiedenen Ergebnisse der OCR-Modelle und viele weitere Daten gespeichert werden. Die Software ist Closed Source und befindet sich in zwei privaten GitHub-Repositories.

Die Erkennung der handschriftlichen Züge erfolgt in mehreren Schritten. Zuerst muss ein Foto oder ein Scan des Scoresheets hochgeladen werden. Danach werden die einzelnen Boxen, in welchen die Züge notiert wurden, mit der Hilfe der Google Cloud Vision API lokalisiert und ausgeschnitten. Der Benutzer gibt nun an, welche die letzte Box ist, die einen Schachzug enthält (dies verhindert, dass in leeren Boxen nach Schachzügen gesucht wird). Im nächsten Schritt werden die ausgeschnittenen Boxen an die verschiedenen OCR-Modelle, welche im ChessReader verwendet werden, gesendet. Diese Modelle versuchen anschliessend die handschriftlichen Züge zu erkennen.

Aktuell werden mehrere OCR-Modelle verwendet, darunter AWS Textract, Azure Computer Vision, PaddleOCR sowie ein eigens entwickeltes Convolutional Neural Network (CNN). Der ChessReader nutzt alle aktivierten Modelle gleichzeitig. Die verschiedenen Ergebnisse, werden dann mit Hilfe der Simple Chess Engine darauf geprüft, welche davon überhaupt legale Schachzüge sind. Anschliessend wird aus allen gültigen Ergebnissen jenes ausgewählt, das gewichtet den höchsten Confidence-Wert aufweist.

Um die verschiedenen Modelle zu vergleichen und generell den ChessReader zu prüfen, existiert eine Evaluationspipeline. Diese liest eine bestimmte Menge von vorgegebenen Scoresheets ein, lässt die Boxen der Züge ebenfalls durch die Google Cloud Vision API ausschneiden und schickt die einzelnen Boxen an die verschiedenen OCR-Modelle. Die Ergebnisse werden in einer Datenbank gespeichert, woraus am Schluss diverse Metriken erstellt und zu einem Report zusammengefasst werden. Neben den verschiedenen Modellen, wird auch der ChessReader als Gesamtsystem inklusive der Simple Chess Engine evaluiert.

4.2 Zielsetzung

Als Erstes wurde das bestehende ChessReader-Projekt und die bereits durchgeführten Arbeiten vorgestellt. Es wurde aufgezeigt, dass das ChessReader-Projekt viele Möglichkeiten bietet. Dabei erwähnt wurde jedoch, dass der Schwerpunkt dieser Arbeit auf der Verbesserung und insbesondere der Internalisierung der OCR-Modelle liegen soll, da der ChessReader derzeit stark von externen Diensten abhängt. Obwohl hierdurch bislang keine schwerwiegenden Probleme aufgetreten sind, verursacht die Nutzung externer Dienste laufende Kosten, welche für eine frei verfügbare Software nachteilig sind. Die School of Engineering verfügt zudem über die technischen Ressourcen, ein OCR-Modell intern zu betreiben und dieses innerhalb von Projekt- und Bachelorarbeiten weiterzuentwickeln.

Auf dieser Grundlage wurde beim Kickoff-Meeting beschlossen, den Fokus auf die Entwicklung eines eigenständigen OCR-Modells zu legen, das handschriftliche Schachzüge erkennen kann. Vorgehensweise und konkrete Architekturentscheidungen wurden bewusst offen gelassen. Ziel ist die Entwicklung eines möglichst präzisen Modells bei akzeptabler Laufzeit. Für ein Scoresheet mit etwa 60 Zügen wird eine Verarbeitungslaufzeit im Bereich von eins bis zwei Minuten angestrebt. Als primäre Metrik der Genauigkeit dient die Ply Accuracy (Anteil korrekt erkannter Züge). Dies da die Hauptaufgabe der ChessReader-Software darin besteht, gesamte Züge korrekt zu erkennen. Dies ist für die Benutzer das einzig relevante Ergebnis. Für die Evaluation wird die bereits bestehende Evaluationspipeline genutzt, um direkte Vergleiche mit anderen Modellen zu ermöglichen.

4.3 Projektrahmen

Da ein Modell von Grund auf entwickelt wird und dies einen hohen Zeitaufwand erfordert, liegt der Fokus dieser Arbeit vollständig auf dessen Entwicklung. Im Rahmen der Bachelorarbeit sind daher keine weiteren Anpassungen oder Verbesserungen am bestehenden Projekt geplant. Änderungen am Quellcode von Backend oder Frontend sowie an der zugrunde liegenden Architektur und Datenbank sind ausgeschlossen. Die Anpassung bzw. Internalisierung der Box-Erkennung ist ebenfalls nicht vorgesehen. Zum einen würde eine solche Erweiterung den Umfang der Arbeit deutlich überschreiten, zum anderen ist die Box-Erkennung sehr ressourcenintensiv. Ob ein interner Betrieb davon sinnvoll ist, müsste in einem separaten Projekt abgeklärt und evaluiert werden.

4.4 Bestehende Arbeiten

Die Digitalisierung handschriftlicher Scoresheets von Schachpartien ist ein Nischenthema innerhalb der Handschrifterkennung. Neben dem ChessReader-Projekt existieren nur wenige wissenschaftliche Arbeiten, die sich explizit mit dieser Aufgabe befassen. Die relevanteste verwandte Arbeit stammt von Majid und Eicher [3], die sowohl einen öffentlichen Datensatz als auch ein eigenes Erkennungsmodell vorgestellt haben.

4.4.1 HCS-Datensatz und BiLSTM-Ansatz von Majid und Eicher

Majid und Eicher veröffentlichten 2021 den *Handwritten Chess Scoresheet* (HCS) Datensatz [4], einen öffentlich zugänglichen Korpus handschriftlicher Scoresheets mit zugehörigen Labels. Der Datensatz enthält eingescannte Scoresheets aus realen Schachpartien und bildet damit die Vielfalt echter Handschriften ab, einschliesslich individueller Schreibstile, unregelmässiger Platzierung innerhalb der Zugfelder und weiterer praxisnaher Variationen.

In ihrer Arbeit [3] entwickelten Majid und Eicher ein System zur Erkennung der handschriftlichen Züge auf Basis eines BiLSTM-Netzwerks (Bidirectional Long Short-Term Memory). Ihr Ansatz verwendet ein CNN als Feature-Extraktor, gefolgt von einem bidirektionalen LSTM, das die extrahierten visuellen Merkmale als Sequenz verarbeitet. Die Decodierung der Ausgabe erfolgt mittels CTC [5], einem Verfahren, das die Zuordnung von Eingabesequenzen zu Ausgabelabels ohne explizite Segmentierung ermöglicht.

Ihr Modell wurde auf dem HCS-Datensatz trainiert und evaluiert. Die Arbeit demonstriert, dass ein auf handschriftliche Schachzüge spezialisiertes OCR-System grundsätzlich realisierbar ist und liefert mit dem HCS-Datensatz eine wichtige Datengrundlage für weiterführende Forschung. Weder der Quellcode noch das trainierte Modell sind öffentlich verfügbar, weshalb kein direkter Vergleich mit dem in dieser Arbeit entwickelten Modell möglich ist.

Der HCS-Datensatz wurde auch in der vorliegenden Arbeit als eine der beiden realen Datenquellen verwendet (siehe Abschnitt 6.3.6). Im Unterschied zu Majid und Eicher verfolgt diese Arbeit einen Transformer-basierten Ansatz und untersucht zusätzlich die Rolle synthetischer Trainingsdaten sowie verschiedener Trainingsstrategien.

5 Theoretische Grundlagen

Vor dem Beginn der Modellentwicklung wurden die relevanten theoretischen Grundlagen sowie die zugrundeliegenden Themen und Technologien untersucht, von den Spielregeln des Schachs über Methoden der Texterkennung bis hin zu neuronalen Netzwerken.

5.1 Schach

Das Schachspiel hat eine lange Geschichte, die tausende von Jahren zurückreicht [6]. Diese muss aber nicht aufgearbeitet werden, um die theoretischen Grundlagen dieser Arbeit zu verstehen.

5.1.1 Das Spiel

Es reicht zu wissen, dass Schach ein strategisches Brettspiel für zwei Personen ist. Diese hocken sich an einem 8-mal-8 Felder grossen Brett gegenüber und bewegen abwechselnd Figuren ihrer Farbe (Weiss oder Schwarz), wobei Weiss den ersten Zug macht. Jede Farbe hat 16 Figuren, welche zu Beginn eines Spieles ihre bestimmten Positionen einnehmen. Ein Spiel endet entweder mit einem Sieg für Weiss bzw. Schwarz oder mit einem Unentschieden. Im Schach wird teilweise auch von Halbzügen gesprochen, wobei ein Halbzug einem Zug einer Person entspricht und ein vollständiger Zug aus zwei Halbzügen besteht. Da die Unterscheidung für diese Arbeit irrelevant ist, wird im weiteren Verlauf nur von Zügen gesprochen.[7]

Abbildung 1 zeigt ein Beispiel für ein Schachspiel zwischen den beiden Schachgrossmeistern Bobby Fischer und Tigran Petrosjan.

Abbildung 1: Bobby Fischer gegen Tigran Petrosjan 1971 in Buenos Aires [8]

5.1.2 Partieformular (Scoresheet)

Schach ist schon lange kein reines Gelegenheitsspiel mehr, sondern hat sich zu einem Sport entwickelt, der sogar vom Internationalen Olympischen Komitee (IOC) anerkannt wird [9]. Das Schachspiel wurde also professionalisiert. So gibt es verschiedenste grosse Turniere, bei denen es auch um eine Menge Geld geht. Daher ist es nicht verwunderlich, dass die Teilnehmenden ihre Partien protokollieren müssen. Wie eingangs erwähnt, werden dafür die gespielten Züge in der Standard Algebraic Notation (SAN) auf Scoresheets notiert.

Ein Scoresheet ist ein einfaches Blatt Papier, das neben einigen Metadaten zur Partie aus einer Tabelle mit zwei Spalten besteht, siehe Abbildung 2. Jede Zeile bietet Platz für zwei Schachzüge (auf der linken Seite für Weiss, auf der rechten Seite für Schwarz). Je nach Design steht links daneben noch eine Zugnummer.[10]

Abbildung 2: Beispiel eines Scoresheets von Bobby Fischer [11]

5.1.3 Standard Algebraic Notation (SAN)

Auf dem Schachbrett werden Linien mit einem Buchstaben (a-h) und Reihen mit einer Zahl (1-8) gekennzeichnet, daraus ergibt sich eine eindeutige Position, wie in Abbildung 3 dargestellt. Die weissen Figuren beginnen auf den Reihen eins und zwei, die schwarzen Figuren auf den Reihen sieben und acht [12].

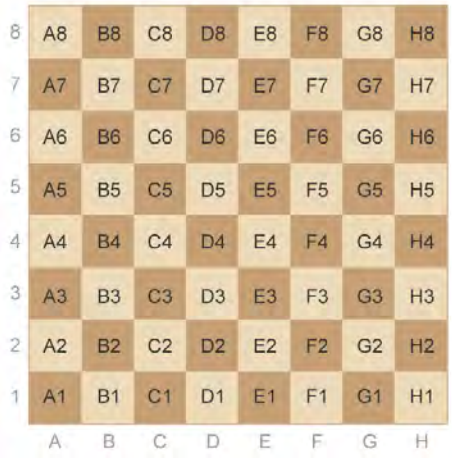


Abbildung 3: Koordinatensystem des Schachbretts [13]

Die in den Scoresheets verwendeten Notationen können aus der Tabelle 1 gelesen werden.

Notation	**Beschreibung**	**Beispiel**
⟨**Figur**⟩⟨**Ziel**⟩	Eine Figur (Bauern ausgeschlossen) bewegt sich von einem Feld auf ein anderes.	Nf3
⟨**Figur**⟩⟨**Startreihe**⟩⟨**Ziel**⟩	Zwei Figuren derselben Art auf derselben Linie können auf dasselbe Feld ziehen.	R1a3
⟨**Figur**⟩⟨**Startlinie**⟩⟨**Ziel**⟩	Zwei Figuren derselben Art auf derselben Spalte können auf dasselbe Feld ziehen.	Nbd2
⟨**Figur**⟩**x**⟨**Ziel**⟩	Eine Figur schlägt eine gegnerische Figur.	Nxe5
⟨**Ziel**⟩	Ein Bauer bewegt sich von einem Feld auf ein anderes.	e4
⟨**Startlinie**⟩**x**⟨**Ziel**⟩	Ein Bauer schlägt eine gegnerische Figur.	exd5
⟨**Startlinie**⟩**x**⟨**Ziel**⟩ **e.p.**	Ein Bauer macht einen "en passant".	exd6 e.p.
O-O / **O-O-O**	Kurze oder lange Rochade.	O-O
⟨**Bauernzug**⟩⟨**Figur**⟩ oder ⟨**Bauernzug**⟩**=**⟨**Figur**⟩	Ein Bauer wird zu einer anderen Figur befördert.	e8Q e8=Q
⟨**Zug**⟩**+**	Ein Zug setzt den Gegner in Schach.	Qh5+
⟨**Zug**⟩**#** oder ⟨**Zug**⟩**++**	Ein Zug setzt den Gegner schachmatt.	Qh7# Qh7++

Tabelle 1: SAN Regeln [1]

Für die Figuren verwendet die SAN je nach Sprache unterschiedliche Buchstaben. In der Abbildung 4 werden die für Deutsch und Englisch verwendeten Notationen aufgeführt [14].

(a) Deutsch

SAN	Figur	Deutsch
K	♔	König
D	♕	Dame
T	♖	Turm
L	♗	Läufer
S	♘	Springer
kein Buchstabe	♙	Bauer

(b) English

SAN	Figur	Englisch
K	♔	King
Q	♕	Queen
R	♖	Rook
B	♗	Bishop
N	♘	Knight
kein Buchstabe	♙	Pawn

Abbildung 4: SAN Figuren (Deutsch & Englisch) [14]

5.2 Optical Character Recognition

Optical Character Recognition (OCR) beschreibt das Verfahren zur Erkennung von Text in Bildern und deren Umwandlung in eine digitale, maschinenlesbare Form. Der OCR-Prozess besteht typischerweise aus mehreren Verarbeitungsschritten, um von einem Eingabebild zu maschinenlesbarem Text zu gelangen. Der Prozess lässt sich in folgende Schritte unterteilen [15]:

- **Preprocessing:** Der erste Schritt besteht darin, das Bild für die Texterkennung vorzubereiten. Das Ziel ist es die Qualität des Bildes zu verbessern, indem Rauschen und unerwünschte Merkmale entfernt werden. Ein paar der wichtigsten Preprocessing-Schritte sind:
  - **Binarisierung:** Umwandlung des Graustufenbildes in ein Schwarzweissbild, sodass Text klar vom Hintergrund getrennt wird. Häufig werden adaptive Schwellenwertverfahren eingesetzt, die lokale Helligkeitsunterschiede berücksichtigen.
  - **Rauschunterdrückung:** Beseitigung von Bildstörungen wie Flecken, Körnigkeit oder Artefakten mittels Filter, um die Erkennungsqualität zu verbessern.
  - **Normalisierung:** Angleichung von Helligkeit, Kontrast und Ausrichtung des Bildes, um konsistente Eingabebedingungen für den Erkennungsalgorithmus zu gewährleisten.
  - **Grössenänderung:** Skalierung des Bildes auf eine einheitliche Auflösung, damit das Modell Zeichen unabhängig von der originalen Bildgrösse verarbeiten kann.
- **Segmentierung:** Aufteilung des Bildes in einzelne Zeichen oder Wörter.
- **Feature-Extraktion:** Identifikation von Features, die zur Unterscheidung von Zeichen notwendig sind.
- **Klassifizierung:** Zuordnung der extrahierten Features zu einem bestimmten Zeichen oder Wort.
- **Postprocessing:** Korrektur von Fehlern, z.B. durch Rechtschreibprüfung oder Kontextanalyse.

5.3 Convolutional Neural Networks

Bei der Entwicklung des OCR-Modells wurde eine Architektur ausgewählt, welche auf einem Convolutional Neural Network (CNN) basiert. Bei CNN handelt es sich um eine Netzarchitektur für Daten mit einer Gitterstruktur, wie zum Beispiel Bilder. Sie sind also besonders gut geeignet für die Erkennung von visuellen Mustern und Merkmalen in Bildern, wie bei handschriftlichen Schachzügen.[16]

Ein CNN besteht aus mehreren Schichten, dem Convolutional Layer, dem Pooling Layer und dem Fully Connected Layer, wie in Abbildung 5 zu sehen ist.

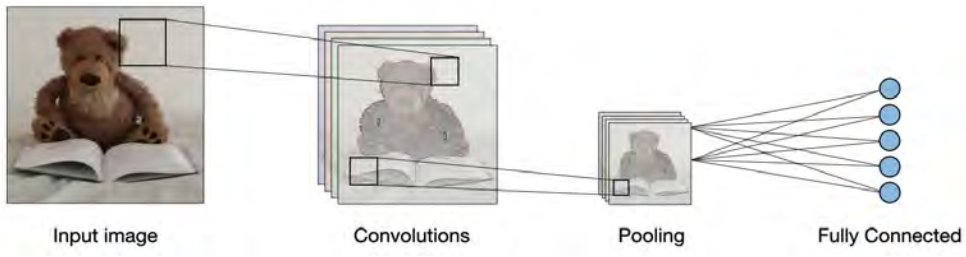


Abbildung 5: CNN-Aufbau anhand eines Beispiels [17]

5.3.1 Convolutional Layer

Ein CNN besteht aus mehreren Convolutional Layers, die hintereinander geschaltet sind. Sie sind die zentralen Bausteine eines CNNs.

In einem Convolutional Layer werden Filter bzw. Kernel über das Eingabebild gelegt und Schritt für Schritt verschoben, um lokale Merkmale zu extrahieren, wie in Abbildung 6 dargestellt. Jeder Filter erkennt dabei ein bestimmtes Merkmal oder Muster (z.B. Kanten oder Ecken). Zu Beginn sind diese relativ einfach und später werden sie immer komplexer (z.B. Formen oder ganze Figuren). Ein Convolutional Layer ist unter anderem definiert durch die Anzahl der Filter, die Filtergrösse, die Schrittweite (Stride) und die Padding-Optionen.[16]

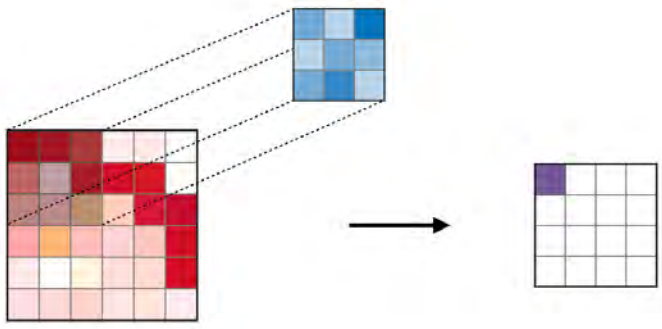


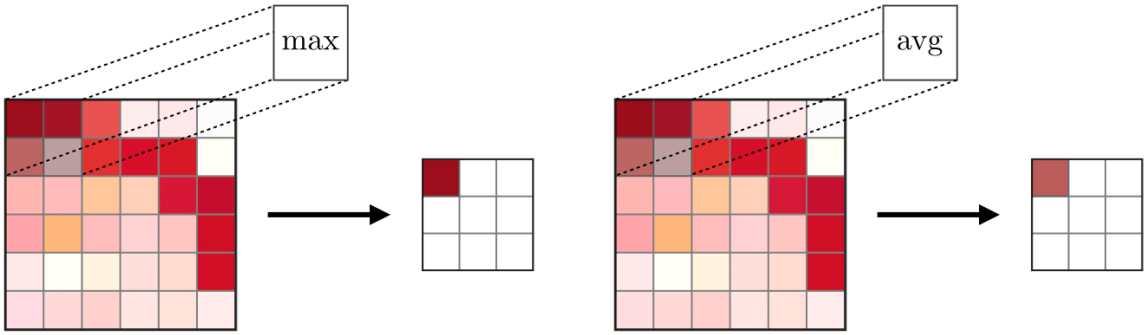
Abbildung 6: Convolutional Layer mit Filter-Verschiebung über das Eingabebild [17]

Nach der Convolutional Operation wird eine nichtlineare Aktivierungsfunktion (z.B. ReLU[1]) angewendet, damit das Modell auch komplexe Muster erkennt.[16]

5.3.2 Pooling Layer

Pooling Layers dienen zur Reduzierung der Dimensionalität, um die Anzahl der Parameter bei der Eingabe zu reduzieren. Dabei gehen Informationen verloren, weshalb sichergestellt werden muss, dass relevante Merkmale möglichst erhalten bleiben. Dies führt zu weniger Rechenaufwand und schützt vor Overfitting. Zudem macht es das Modell robuster gegenüber kleinen Veränderungen im Bild. Beim Pooling wird ähnlich wie beim Convolutional Layer ein Filter über das Bild gelegt. Allerdings hat dieser keine Gewichtung sondern handelt nach einer von zwei Arten [16]:

- **Max-Pooling:** Der höchste Wert im Filterbereich wird ausgewählt wie in Abbildung 7a dargestellt.
- **Average-Pooling:** Aus allen Feldern im Filterbereich wird der Durchschnitt berechnet wie in Abbildung 7b dargestellt.



(a) Max-Pooling [17] (b) Average-Pooling [17]

Abbildung 7: Pooling Layer Arten: Max-Pooling und Average-Pooling

[1]Setzt negative Werte auf 0 und lässt positive Werte unverändert [17]

5.3.3 Fully Connected Layer

Fully Connected Layers verbinden alle Neuronen der vorherigen Schicht mit allen Neuronen der aktuellen Schicht, wie die Abbildung 8 zeigt. Dadurch werden die extrahierten Merkmale aus den vorherigen Schichten genutzt, um eine Vorhersage zu treffen. Zum Schluss der Schicht wird eine Funktion (z.B. Softmax[2]) verwendet, um die Ausgabe in Wahrscheinlichkeiten umzuwandeln.[16]

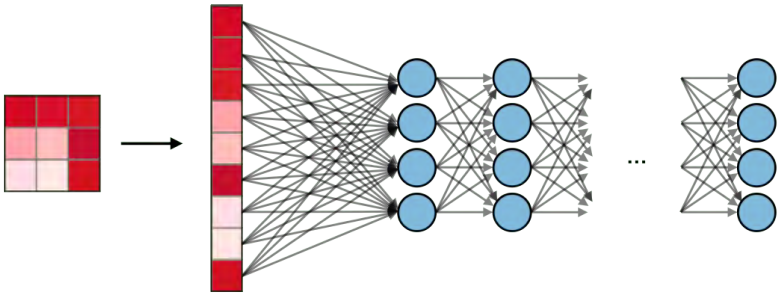


Abbildung 8: Fully Connected Layer mit vollständigen Verbindungen zwischen Neuronen [17]

5.4 Transformer Model

Transformer sind neuronale Netzwerkarchitekturen für die Verarbeitung von Sequenzen. Sie wurden ursprünglich für maschinelle Übersetzung vorgeschlagen und ersetzen rückgekoppelte Verarbeitungsschritte durch Attention-Mechanismen [18]. Dadurch können Abhängigkeiten zwischen beliebigen Positionen einer Sequenz direkt modelliert werden. Für Aufgaben wie Texterkennung ist diese Eigenschaft relevant, da das Ergebnis nicht nur aus einzelnen unabhängigen Zeichen besteht, sondern aus einer geordneten Zeichenfolge.

5.4.1 Token, Embeddings und Positional Information

Ein Transformer verarbeitet eine Eingabe als Sequenz von Tokens. Bei Text können Tokens beispielsweise Wörter, Teilwörter oder einzelne Zeichen sein. Da neuronale Netze nicht direkt mit symbolischen Tokens arbeiten, werden diese zuerst in numerische Vektoren umgewandelt. Diese Vektoren werden als Embeddings bezeichnet.

Im Unterschied zu rückgekoppelten Netzen besitzt ein Transformer keine eingebaute zeitliche Reihenfolge. Damit das Modell trotzdem zwischen der ersten, zweiten oder späteren Position einer Sequenz unterscheiden kann, werden Positional Encodings zu den Embeddings hinzugefügt [18].

5.4.2 Attention und Self-Attention

Der zentrale Baustein eines Transformers ist Attention. Attention beschreibt einen Mechanismus, bei dem ein Element einer Sequenz gewichtet auf andere Elemente zugreifen kann. Dadurch kann das Modell lernen, welche Positionen für eine bestimmte Repräsentation oder Vorhersage besonders relevant sind.[18]

[2]Wandelt die Ausgabewerte in Wahrscheinlichkeiten um (die Summe aller Werte ergibt 1) [17]

Bei Self-Attention stammen die verglichenen Elemente aus derselben Sequenz. Jedes Token kann dadurch Informationen aus anderen Positionen derselben Eingabe einbeziehen. Multi-Head Attention erweitert dieses Prinzip, indem mehrere Attention-Berechnungen parallel durchgeführt werden. Die einzelnen Attention-Heads können dadurch unterschiedliche Beziehungen innerhalb der Sequenz lernen.[18]

5.4.3 Encoder und Decoder

Ein klassischer Transformer besteht aus einem Encoder und einem Decoder. Der Encoder verarbeitet die Eingabesequenz und erzeugt daraus kontextualisierte Repräsentationen. Jede Encoder-Schicht enthält typischerweise eine Multi-Head-Self-Attention-Schicht und ein Feed-Forward Network. Residual Connections und Layer-Normalisierung stabilisieren dabei das Training [18].

Der Decoder erzeugt die Ausgabesequenz schrittweise. Dabei wird Masked Self-Attention verwendet, damit eine Position nur frühere Ausgabetokens sehen kann. So wird verhindert, dass beim Training Informationen aus zukünftigen Tokens verwendet werden. Zusätzlich besitzt der Decoder eine Cross-Attention-Schicht, welche auf die Ausgaben des Encoders zugreift. Die Vorhersage erfolgt autoregressiv. Das bedeutet, dass für jedes neue Token die bereits erzeugten Token als Kontext verwendet werden.[18] Die grundlegende Encoder-Decoder-Struktur ist in Abbildung 9 dargestellt.

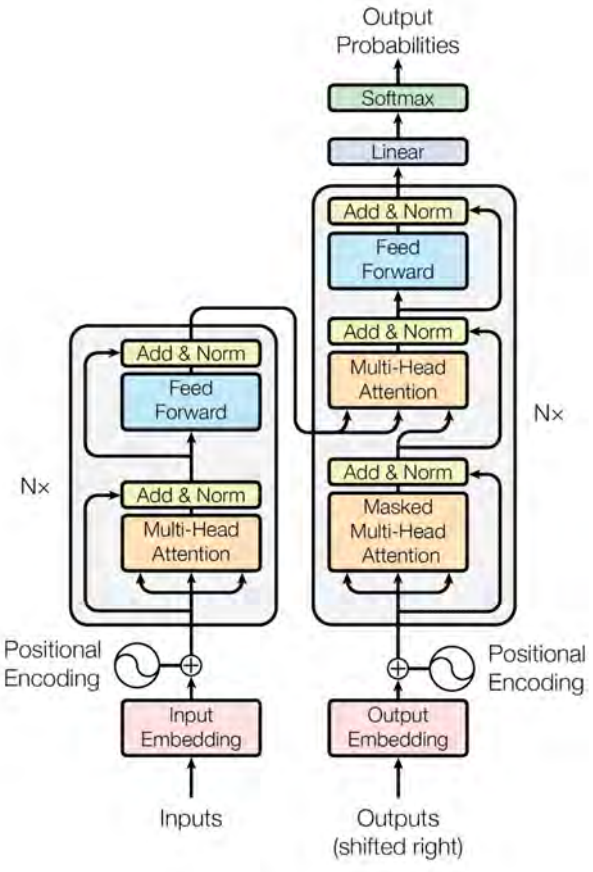


Abbildung 9: Transformer-Encoder-Decoder-Architektur nach *Attention Is All You Need* [18].

5.4.4 Transformer in der Bilderkennung und OCR

Obwohl Transformer ursprünglich für Textsequenzen entwickelt wurden, können sie auch auf Bilddaten angewendet werden. Der Vision Transformer (ViT) teilt ein Bild in kleine Patches auf und behandelt diese Patches ähnlich wie Tokens in einer Textsequenz [19]. Damit wird ein Bild in eine Sequenz visueller Tokens überführt, die durch Transformer-Schichten verarbeitet werden kann.

In der OCR werden Transformer unter anderem in Bild-zu-Text-Modellen eingesetzt. Ein Beispiel dafür ist TrOCR, welches einen Transformer-basierten Bild-Encoder mit einem Transformer-basierten Text-Decoder kombiniert [20]. Solche Modelle zeigen, warum Transformer für Texterkennung relevant sind. Die visuelle Eingabe muss mit einer variablen Ausgabesequenz verbunden werden. Für die theoretischen Grundlagen sind deshalb insbesondere Self-Attention, Encoder-Decoder-Strukturen und autoregressive Decodierung wichtig.

5.4.5 Greedy Decoding und Beam Search

Bei der Inferenz muss der Decoder entscheiden, welches Token als nächstes ausgegeben wird. Da der Transformer-Decoder autoregressiv arbeitet, wird bei jedem Schritt auf Basis der bisher erzeugten Tokens und der Encoder-Ausgabe eine Probability Distribution über das nächste Token berechnet. Zwei gängige Decodierungsstrategien unterscheiden sich darin, wie aus dieser Verteilung ein Token ausgewählt wird [21]:

- **Greedy Decoding:** Bei jedem Schritt wird das wahrscheinlichste Token gewählt. Die Decodierung ist deterministisch und schnell, garantiert aber nicht die global wahrscheinlichste Sequenz.
- **Beam Search:** Es werden mehrere Kandidatensequenzen (Beams) parallel verfolgt. Bei jedem Schritt werden die wahrscheinlichsten Fortsetzungen aller Beams berechnet und nur die besten behalten. Die Breite des Suchraums wird durch die Beam Width gesteuert.

Ein Vorteil des autoregressiven Decoders bei Beam Search ist die Möglichkeit, domänenspezifisches Wissen als Constraints einzubauen. Im Kontext der SAN-Erkennung können beispielsweise syntaktisch ungültige Züge während dem Beam Search ausgeschlossen werden. So kann verhindert werden, dass das Modell Sequenzen erzeugt, die der SAN-Grammatik widersprechen, auch wenn diese laut Modell eine hohe Wahrscheinlichkeit aufweisen.

5.5 Connectionist Temporal Classification

Connectionist Temporal Classification (CTC) ist eine Loss Function und Decoding-Methode zur Klassifikation unalignierter Sequenzen. Sie wurde entwickelt, um das Problem zu lösen, dass bei der Erkennung von Sequenzen (z. B. Audio- oder Bildsegmente) keine explizite Ausrichtung zwischen Eingabe und Ausgabe bekannt ist. Im Kontext der OCR erhält das Modell eine Sequenz von Feature-Vektoren, die aus einem CNN extrahiert wurden, und soll daraus eine kürzere Zeichensequenz vorhersagen. Das Besondere dabei ist, dass nicht bekannt ist, welche Eingabeposition zu welchem Zeichen gehört.[5]

5.5.1 Das Blank-Token

CTC führt ein spezielles Zeichen ein, das sogenannte Blank-Token (ε), welches keinem echten Zeichen entspricht. Das Modell gibt an jedem Zeitschritt eine Probability Distribution über alle echten Zeichen sowie das Blank-Token aus. Das Blank-Token dient als Trennzeichen und ermöglicht es dem Modell, an Positionen ohne relevante Ausgabe eine neutrale Vorhersage zu treffen.[22]

5.5.2 Decodierung: Collapse-Regel

Die unverarbeitete Ausgabe des Modells ist ein Pfad $\pi = (\pi_1, \pi_2, \ldots, \pi_T)$ mit einer Vorhersage pro Zeitschritt. Um daraus die endgültige Zeichenfolge zu rekonstruieren, wird die Collapse-Funktion $\mathcal{B}$ angewendet:

1. Aufeinanderfolgende identische Zeichen werden zu einem zusammengefasst.
2. Alle Blank-Tokens werden entfernt.

Ein konkretes Beispiel für den Schachzug *Nf3* verdeutlicht dies:

Pfad:	`N` – `N` – ε – `f` – `f` – `3`
Nach Schritt 1:	`N` – ε – `f` – `3`
Nach Schritt 2:	`Nf3`

Durch diese Regel kann derselbe Zieltext durch viele verschiedene Pfade repräsentiert werden. Die CTC-Loss Function summiert während des Trainings die Wahrscheinlichkeiten aller gültigen Pfade, die zur korrekten Ausgabe kollabieren.

Da die Anzahl möglicher Pfade exponentiell wächst, wird diese Summe nicht direkt berechnet, sondern über einen effizienten Forward-Backward-Algorithmus auf Basis von Dynamic Programming [5].

5.5.3 Greedy Decoding und Beam Search

Die Decodierung-Strategien sind konzeptionell dieselben wie beim Transformer-Decoder. Der Unterschied liegt darin, wie ein Pfad in eine Ausgabesequenz überführt wird. Beim CTC wird nach der Decodierung zusätzlich die Collapse-Funktion angewendet [22]:

- **Greedy Decoding:** An jedem Zeitschritt wird das wahrscheinlichste Zeichen (inkl. Blank-Token) gewählt. Der resultierende Pfad wird anschliessend mit der Collapse-Funktion reduziert.
- **Beam Search:** Mehrere Pfade von Kandidaten werden parallel verfolgt. Da CTC keine autoregressiven Abhängigkeiten modelliert, können domänenspezifische Constraints wie SAN-Grammatikregeln hier nicht auf dieselbe Weise in die Auswahl der Beams einfliessen wie beim Transformer-Decoder.

5.5.4 Vergleich mit Transformer-Decoder

Während CTC alle Ausgabeentscheidungen unabhängig voneinander und parallel, ohne explizite Ausrichtung trifft, verfolgt der autoregressive Transformer-Decoder einen anderen Ansatz zur Sequenzvorhersage. Ein autoregressiver Decoder hingegen erzeugt die Ausgabesequenz Schritt für Schritt und kann bereits erzeugte Zeichen als Kontext nutzen. Dies ermöglicht eine stärkere Modellierung von Abhängigkeiten zwischen Zeichen, ist aber entsprechend langsamer.

5.6 Evaluationsmetriken

Die Evaluations-Pipeline des ChessReader nutzt verschiedenste Metriken, um die Qualität der Zug- und Zeichenerkennung zu bewerten. Die Metriken sind in mehrere Kategorien unterteilt [23]:

Ply Accuracy. Die Ply Accuracy misst die Genauigkeit der Vorhersagen von Schachzügen. Sie berechnet, wie oft das Ergebnis der ChessReader-Software einen Schachzug (z.B. Nf3) korrekt bestimmt. Diese Metrik ist zentral für die Bewertung der Ende-zu-Ende-Systemleistung und auch für die Bewertung der Modelle in dieser Arbeit.

Character Accuracy. Die Character Accuracy funktioniert ähnlich wie die Ply Accuracy, aber auf Zeichenebene. Sie misst, wie viele der einzelnen Zeichen des Schachzugs korrekt erkannt wurden.

Confidence Confusion Matrix. Diese Metrik analysiert die Verlässlichkeit des ChessReader Confidence Scores. Unter Verwendung des Schwellenwerts 0.9 werden Vorhersagen klassifiziert als:

- True Positives (TP): Eine korrekte Vorhersage mit einer Konfidenz ab 0.9.
- False Positives (FP): Eine falsche Vorhersage mit einer Konfidenz ab 0.9.
- False Negatives (FN): Eine korrekte Vorhersage mit einer Konfidenz von weniger als 0.9.
- True Negatives (TN): Eine falsche Vorhersage mit einer Konfidenz von weniger als 0.9.

Aus dieser Matrix werden Precision, Recall und F_1-Score abgeleitet:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \quad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad F_1\text{-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Character Confusion (Matrix). Diese Metrik vergleicht die echten Züge mit den vom OCR-Modell vorhergesagten Zügen auf Zeichenebene. Dabei werden drei Fehlerarten unterschieden:

- Ein Zeichen fehlt in der Vorhersage.
- Ein Zeichen ist zu viel in der Vorhersage.
- Ein Zeichen ist falsch in der Vorhersage.

Daraus kann eine Character Confusion Matrix erstellt werden. Diese ermöglicht es, Zeichenpaare zu erkennen, welche oft verwechselt, vergessen oder hinzugefügt werden.

6 Vorgehen

In diesem Kapitel wird das methodische Vorgehen beschrieben, welches für die Durchführung der Bachelorarbeit verwendet wurde.

6.1 Analyse des bestehenden ChessReaders

Zu Beginn wurden die bereits bestehenden GitHub-Repositories analysiert und das Setup lokal getestet. Dabei wurde insbesondere geprüft, wie ausgeschnittene Zugboxen an die OCR-Modelle übergeben werden, welches Ausgabeformat von den Modellen erwartet wird und welche Metriken der bestehenden Evaluationspipeline für spätere Modellvergleiche relevant sind. Parallel dazu wurde ein Überblick über frühere Arbeiten am ChessReader-Projekt verschafft und die vorhandenen Dokumentationen analysiert, um bestehende Architektur-Entscheidungen, Datensatz-Versionen und Evaluations-Konventionen einzuordnen.

6.2 Methodische Schwerpunkte

Nach der Analyse des bestehenden Quellcodes wurden drei methodische Schwerpunkte festgelegt.

Der erste Schwerpunkt war die Integration des neuen OCR-Modells in die bestehende ChessReader-Logik. Dazu gehörten die Schnittstelle für Bilder der Schachzüge, die erwartete SAN-Ausgabe und die spätere Vergleichbarkeit mit bestehenden OCR-Modellen.

Der zweite Schwerpunkt lag auf den Datensätzen, die für das Training und die Evaluation verwendet wurden. Dabei zeigte sich, dass mehrere Datensätze aus früheren Arbeiten vorhanden waren. Im Wesentlichen gab es aber nur zwei unabhängige Datenquellen, konkret den *HCS*-Datensatz und den *ChessReader-Corpus v5*. Die weiteren realen Datensätze enthielten hauptsächlich Daten aus letzterem in anderer Struktur oder kleinerem Umfang und wurden deshalb nicht als eigenständige Datenquelle behandelt. Für synthetische Daten stand zu Beginn ein Datensatz mit generierten Schachzug-Bildern zur Verfügung, welche aus einer früheren Projektarbeit stammten [24]. Zusätzlich stand der Quellcode des Generators zur Verfügung, mit dem die synthetischen Daten erzeugt wurden. Dadurch konnten im Rahmen dieser Arbeit zusätzliche synthetische Datensätze generiert werden.

Der dritte Schwerpunkt betraf die Trainings- und Evaluationsumgebung. Für die Erarbeitung des OCR-Modells wurden durch Experimente verschiedene Modellarchitekturen, Hyperparameter und Trainingsstrategien untersucht. Diese Experimente wurden so aufgebaut, dass Modellarchitektur, verwendete Datensätze und Resultate getrennt und dadurch nachvollziehbar bleiben. Dadurch konnte später unterschieden werden, ob ein Ergebnis auf die Architektur, die verwendeten Daten oder die Trainingsparameter zurückzuführen war. Um die Vergleichbarkeit mit früheren ChessReader-Arbeiten zu gewährleisten, wurde der *ChessReader-Corpus v5* auch in dieser Arbeit als Evaluationsgrundlage verwendet.

6.3 Datensätze

Die Datensätze bestanden zu Beginn aus synthetisch erzeugten Schachzug-Bildern, sowie aus realen handschriftlichen Scoresheets inklusive zugehöriger Metadaten-Files.

Dieser Abschnitt beschreibt die Herkunft, Verarbeitung und Rolle der final verwendeten Datensätze. Ältere Datensätze wurden zur Nachvollziehbarkeit aufbewahrt, aber nicht alle Zwischenstände werden als eigenständige Datengrundlage in der Dokumentation aufgeführt. Zur besseren Lesbarkeit werden Abkürzungen für die einzelnen Datensätze eingeführt. Die technischen Namen der Datensätze werden bei der ersten Nennung angegeben, damit der Bezug zur finalen Ablagestruktur hergestellt werden kann.

6.3.1 Einheitliche Bezeichnungen

Während der Arbeit wurden Datensätze zunächst mit projektinternen oder aus früheren Arbeiten übernommenen Namen bezeichnet. Die Vereinheitlichung der Bezeichnungen erfolgte erst gegen Ende der Arbeit auf Anfrage des Betreuers. Um Verwechslungen zu vermeiden, werden in dieser Arbeit die finalen und bereinigten Namen der Datensätze verwendet. Ältere Ausgangsdaten, die nicht als finale Versionen weitergeführt wurden, werden beschreibend benannt.

Die neuen Bezeichnungen beschreiben jeweils Datenart, Herkunft, Version, Verarbeitungsschritt und Anzahl Bilder. Für synthetische Daten wird der Präfix `synthetic` verwendet und für reale Daten `real`. Beispielsweise bezeichnet der Name `real_hcs_corpus_1.0` einen realen Datensatz auf Basis des *HCS-Corpus* in der ersten Version. Abgeleitete Datensätze behalten die Rohdaten unverändert und beschreiben Verarbeitungsschritte wie `boxed` oder `90_10` am Ende des Namens. Ganz zum Schluss wird die Anzahl der Bilder angegeben, um die Übersicht zu erleichtern.

6.3.2 Grundlage der synthetischen Datensätze

Alle synthetischen Datensätze basierten auf einem vorhandenen Generator [24]. Die Scripts dazu wurden gesammelt und in ein eigenständiges Repository hochgeladen. Teil des Generators ist eine Datei mit möglichen SAN-Zügen und zugehörigen Wahrscheinlichkeiten. Die Herkunft von dieser Auflistung und deren Limitation war jedoch nicht dokumentiert. Die Liste enthält 16 002 eindeutige Einträge, beim Recherchieren konnte keine genaue Zahl ermittelt werden, wie viele verschiedene SAN-Züge tatsächlich existieren. Eine Stichprobenkontrolle zeigte jedoch, dass unter anderem die beiden Züge *Ka1#* und *B1a4+* nicht in der Liste enthalten waren. Daher kann davon ausgegangen werden, dass die Liste nicht abschliessend ist. Sicher ist jedoch, dass alle Zeichen, welche in Schachzügen vorkommen, in der Liste enthalten sind. Sie wurde deshalb für gut genug befunden, um die Vielfalt der möglichen SAN-Züge abzubilden. Ausserdem steckte bereits viel Arbeit in der Aufbereitung mit den Wahrscheinlichkeiten von den Zügen. Daher wurde der Aufwand dies zu überarbeiten als übermässig hoch eingeschätzt. Die bestehende Liste wurde deshalb als technische Grundlage der synthetischen Datengenerierung verwendet, aber nicht als garantierter Nachweis eines vollständigen SAN-Vokabulars interpretiert.

6.3.3 Synthetischer Basisdatensatz (*SYN-200K*)

Zu Beginn stand ein bereits generierter synthetischer Datensatz mit 200 000 generierten Schachzug-Bildern und zugehörigen Labels zur Verfügung.

Für die Experimente dieser Arbeit wurde daraus ein 90-10-Split erstellt. Für das Training wurden 180,000 Bilder verwendet, während die restlichen 20,000 Bilder für die Validierung genutzt wurden.

Die Bilder wurden mithilfe des Generators erzeugt. Durch die Liste mit Schachzügen und deren Wahrscheinlichkeiten treten Züge mit einer hohen Wahrscheinlichkeit entsprechend oft im Datensatz auf. Seltene SAN-Formen besitzen dabei eine sehr geringe Gewichtung, sodass sie selbst bei einem Umfang von 200,000 Bildern nur vereinzelt oder teilweise gar nicht im Datensatz enthalten sind.

Im Folgenden wird dieser Datensatz als *SYN-200K* bezeichnet.

6.3.4 Synthetischer Full-Cycles-Corpus (*SYN-FC*)

Der Datensatz `synthetic_full_cycles_corpus_320k` wurde im Rahmen dieser Arbeit erzeugt, um die Abdeckung des SAN-Vokabulars zu verbessern. Für den Full-Cycles-Corpus wurden die vorhandenen Schachzüge in der oben erwähnten Liste zyklisch generiert.

Die Datenerzeugung erfolgte direkt mit getrennten Splits. Die finale Struktur enthielt 320 040 Trainingsbilder, 32 004 Bilder zur Validierung und ebenso viele Testbilder.

Der Vorteil dieses Datensatzes liegt in der breiten Abdeckung seltener Schachzüge. Gleichzeitig entsteht dadurch eine künstliche Verteilung. Da die seltenen Züge in diesem Datensatz deutlich häufiger vorkommen, als dies in realen Schachpartien zu erwarten wäre.

Im weiteren Text wird dieser Datensatz als *SYN-FC* bezeichnet.

6.3.5 Synthetischer Weighted-Cycles-Corpus (*SYN-WC*)

Der Datensatz `synthetic_weighted_cycles_corpus_371k` wurde ebenfalls im Rahmen dieser Arbeit erzeugt. Er verfolgt wie der *SYN-FC* das Ziel, seltene SAN-Formen besser abzudecken als der ursprüngliche *SYN-200K*-Datensatz. Im Unterschied zum *SYN-FC* wurden die Häufigkeiten der im Generator vorhandenen Liste wieder miteinbezogen. Im Vergleich zu *SYN-200K* wurden die einzelnen Wahrscheinlichkeiten der Schachzüge jedoch angepasst. Dort war die höchste Wahrscheinlichkeit für einen Zug knapp 30 000, während gewisse Züge nur einmal vorkamen. Es wurde also definiert, dass der höchste Wert auf 100 reduziert wird, während die anderen Werte entsprechend skaliert wurden. Zusätzlich wird auch kein Zufallsgenerator mehr verwendet, sondern die Züge werden in Zyklen generiert, und zwar so oft wie sie in der angepassten Liste vorkommen. Dadurch werden seltene Formen gezielt einbezogen, ohne sie gegenüber sehr häufigen Zügen vollständig gleichzustellen.

Auch dieser Datensatz wurde direkt mit getrennten Splits erzeugt. Die finale Struktur enthielt 371 655 Trainingsbilder, 74 331 Bilder zur Validierung und genauso viele Testbilder.

Die gewichtete Variante reduziert das Problem einer unrealistischen Gleichverteilung seltener und häufiger Zügen. Da es sich immer noch um einen synthetischen Datensatz handelt, ersetzt dieser nicht die reale Handschrift.

Im weiteren Text wird dieser Datensatz als *SYN-WC* bezeichnet.

6.3.6 HCS-Corpus (*HCS*)

Der Datensatz `real_hcs_corpus_1.0` basiert auf dem öffentlichen Korpus *Handwritten Chess Scoresheet* (HCS) [3, 4]. Die lokale Rohfassung des Datensatzes bestand aus digitalisierten, vollständigen Scoresheets und zwei Labeldateien, eine für Training und eine für Testing. Diese Labels beschreiben die notierten Züge, enthalten aber keine weiteren Metadaten wie die Box-Koordinaten.

Für das Trainieren und Validieren der OCR-Modelle in dieser Arbeit waren vollständige Scoresheets nicht direkt geeignet. Die Trainingspipeline erwartete einzelne Bilder, die jeweils genau einen Schachzug enthalten. Deshalb mussten aus dem *HCS*-Datensatz zuerst einzelne Zugboxen erstellt werden. Initial wurden die Boxen mit der Google Vision API erzeugt. Dafür wurden JSON-Dateien mit den Koordinaten, Höhe und Breite der Boxen so wie auch weiteren Informationen generiert. Dies ermöglichte es die Boxen der Zugfelder zu überarbeiten und die Informationen kompakt abzuspeichern. Bei einer Nachkontrolle der Boxen zeigte sich, dass die Google Vision API zwar eine gute Ausgangsbasis lieferte, aber nicht alle Boxen korrekt erkannte. Zudem wurde festgestellt, dass vereinzelt die Schachzüge falsch gelabelt waren. Deshalb wurden fehlerhafte oder fehlende Boxen gezielt korrigiert und teilweise selbst eingezeichnet. Dabei fiel auch auf, dass ein paar Scoresheets doppelt vorkamen, welche entfernt wurden. Zudem gab es Schachzüge, welche nicht lesbar waren, diese wurden ebenfalls entfernt. Danach wurde ein 90/10-Split für Training und Validierung erstellt.

Dieser Datensatz besteht im Unterschied zu den synthetischen Datensätzen ausschliesslich aus echten Handschriften. Er enthält deshalb Variationen, die in synthetischen Bildern nur näherungsweise abgebildet werden können, beispielsweise individuelle Handschriften, unregelmässige Platzierung innerhalb der Boxen und sonstige Anomalien.

Im weiteren Text wird dieser Datensatz als *HCS* bezeichnet.

6.3.7 ChessReader-Corpus v6 Englisch (*CR-v6-EN*)

Der Datensatz `real_chessreader_corpus_6.0_boxed_3111` basiert auf früheren Arbeiten zur Datenaufbereitung für das ChessReader-Projekt. Ausgangspunkt war der von Moser und Meier dokumentierte *Corpus v5* [23]. Scoresheets aus diesem Korpus enthalten sowohl die deutsche als auch die englische Variante von Schachfiguren. Er enthält reale Scoresheets und dazugehörige JSON-Dateien mit Metadaten, darunter die SAN-Labels und Informationen über die Position und Grösse der Boxen.

Im Rahmen dieser Arbeit wurde der *Corpus v5* überprüft und überarbeitet. Dabei wurden unter anderem fehlerhafte Labels bereinigt und Boxen korrigiert.

Auch dieser Datensatz musste für die weitere Verwendung in einzelne Bilder mit jeweils einem Schachzug überführt werden. Dafür wurden die Box-Koordinaten aus den JSON-Dateien genutzt, um die englischsprachigen Scoresheets auszuschneiden.

Der neue Datensatz enthält 3 111 Einzelzugbilder. Dieser Datensatz hat eine besondere Rolle, weil er die Vergleichbarkeit mit früheren Evaluationen ermöglicht. Deshalb wurde er nicht als Trainingsdatensatz verwendet, sondern als Grundlage für die Evaluation.

Im weiteren Text wird dieser Datensatz als *CR-v6-EN* bezeichnet.

6.4 Experimentelles Vorgehen

Dieser Abschnitt beschreibt die Organisation der Experimente für die Ausarbeitung des OCR-Modells. Die konkreten Modellversionen, Resultate und Interpretationen folgen anschliessend in Kapitel 8.

Für die Experimente wurden Modellname, Architektur, Datensatz, Hyperparameter, Resultate und Beobachtungen systematisch dokumentiert. Zudem wurde eine Namenskonvention definiert, um Modellversionen und Datensatzvarianten eindeutig nachvollziehbar zu halten.

Die Experimente wurden nicht als voneinander unabhängige Einzelversuche betrachtet, sondern in aufeinander aufbauende Gruppen gegliedert. Zunächst wurden Transformer-Encoder-Decoder-Modelle untersucht. Anschliessend wurden Anpassungen an Architektur, Datensätzen und Trainingsstrategie vorgenommen. Nach den Experimenten mit der Transformer-Architektur wurde zusätzlich ein CTC-Modell betrachtet, um eine Alternative zum autoregressiven Decoder zu prüfen. Zum Schluss wurden Experimente mit verschiedenen Datensätzen fürs Training und Finetuning durchgeführt.

Für die Bewertung wurden Trainings-, Validierungs- und Evaluationsdaten strikt getrennt. Datensätze mit explizitem Trainings-, Validierungs- und Testsplit wurden in dieser Struktur übernommen. Bei realen Trainingsdaten ohne vorgegebenen Split wurde ein 90/10-Split verwendet. Der Datensatz *CR-v6-EN* wurde nicht für das Training eingesetzt, sondern nur für die abschliessende Evaluation verwendet. Die Ergebnisse im Kapitel 8 beziehen sich alle auf die Evaluation auf dem *CR-v6-EN*-Datensatz. Jedoch wurde für das Testen der Modelle nicht die Evaluationspipeline verwendet sondern direkt mit einem internen Skript evaluiert, um die Ergebnisse schneller zu erhalten. Deshalb weichen die Werte in Kapitel 8 leicht von den Werten ab, welche die Evaluationspipeline liefert. Der zugehörige Report ist im Anhang in Abschnitt C abgelegt.

7 Modell

Dieses Kapitel beschreibt die entwickelte Modellarchitektur, die als Grundlage für alle in Kapitel 8 durchgeführten Experimente diente. Zuerst wird der Architekturentscheid begründet, anschliessend wird das Basismodell im Detail beschrieben.

7.1 Architekturentscheid: CNN-Backbone und Transformer-Encoder-Decoder

Für das OCR-Modell wurde eine Architektur bestehend aus einem CNN-Backbone und einem Transformer-Encoder-Decoder gewählt. Diese Entscheidung entstand aus der Abwägung zweier bekannter Ansätze. Eine CNN+LSTM-Architektur wurde bereits in einer vorherigen Arbeit zum ChessReader-Projekt verwendet [25] und wurde in dieser Arbeit nicht erneut untersucht. Rein Transformer-basierte Modelle wie TrOCR oder der Vision Transformer (ViT) benötigen typischerweise sehr grosse Mengen an Trainingsdaten, da sie ohne CNN-Inductive-Bias aus dem Bild lernen müssen [19, 20]. Da nur begrenzt viele reale handschriftliche Daten zur Verfügung standen, wurde ein Mittelweg gewählt. Ein CNN-Backbone für die lokale Feature-Extraktion kombiniert mit einem Transformer-Encoder-Decoder für die Sequenzvorhersage. Diese Kombination nutzt die Dateneffizienz des CNN sowie die Fähigkeit des Transformers, mittels Self-Attention globale Abhängigkeiten über die gesamte Sequenz zu modellieren.

7.2 Architektur des Basismodells

Das entwickelte Modell folgt einer Encoder-Decoder-Architektur, bei der ein CNN-Backbone die visuellen Features extrahiert und ein Transformer-Encoder-Decoder die Zeichensequenz autoregressiv vorhersagt.

Da jedes Eingabebild genau einen Schachzug auf einer einzelnen horizontalen Zeile enthält, kann die Höhendimension nach der CNN-Feature-Extraktion kollabiert werden. Der Transformer-Encoder operiert dadurch auf einer 1-D-Sequenz, deren Länge der Feature-Map-Breite entspricht, also der Leserichtung.

7.2.1 CNN-Backbone

Der CNN-Backbone besteht aus drei aufeinanderfolgenden Convolutional Blocks. Jeder Block enthält ein Convolutional Layer, Batch Normalization und eine ReLU Aktivierungsfunktion. Nach den ersten beiden Blöcken wird jeweils ein 2×2 Max-Pooling angewendet, das die räumlichen Dimensionen halbiert, wie die Tabelle 2 zeigt.

Block	Layer	Channels	Output (bei 96×384)
1	Conv2d + BN + ReLU	$1 \to 64$	96×384
	MaxPool2d 2×2		48×192
2	Conv2d + BN + ReLU	$64 \to 128$	48×192
	MaxPool2d 2×2		24×96
3	Conv2d + BN + ReLU	$128 \to 256$	24×96

Tabelle 2: CNN-Backbone Architektur

Alle Convolutional Layers verwenden eine Filtergrösse von 3×3 mit entsprechendem Padding, sodass die räumlichen Dimensionen vor dem Max-Pooling erhalten bleiben.

7.2.2 Height Collapse

Nach dem CNN-Backbone liegt eine 2-D Feature-Map der Form $(B, 256, 24, 96)$ vor. Da die Eingabe stets eine einzelne Textzeile enthält, wird die Höhendimension durch ein Max-Pooling auf die stärksten spaltenweisen Aktivierungen reduziert. Anschliessend werden die Dimensionen permutiert, damit die Sequenzposition (Breite) als zweite Dimension vorliegt, wie vom Transformer erwartet:

$$(B,\ 256,\ 24,\ 96) \xrightarrow{\text{max over height}} (B,\ 256,\ 96) \xrightarrow{\text{permute}} (B,\ 96,\ 256)$$

Max-Pooling wird gegenüber Average-Pooling bevorzugt, da es die stärksten Aktivierungen beibehält, anstatt sie mit schwächeren Features oder Hintergrund zu verwässern. Das Ergebnis ist eine 1-D-Sequenz von 96 Tokens, jeder mit einer Dimension von 256. Vor dem Transformer-Encoder wird zusätzlich eine Layer-Normalisierung angewendet, um die durch Max-Pooling verursachten grossen Aktivierungswerte zu stabilisieren.

7.2.3 Transformer-Encoder

Der Encoder verarbeitet die 96 visuellen Tokens. Sinusförmige Positional Encodings werden zu den Embeddings addiert, damit das Modell die Position entlang der Leserichtung berücksichtigen kann. Die wichtigsten Hyperparameter des Encoders sind in Tabelle 3 zusammengefasst.

Parameter	**Wert**
Embedding Dimension	256
Anzahl Encoder-Layers	2
Anzahl Attention-Heads	4
Feed-Forward Dimension	1024 ($4\times$ Embedding Dimension)

Tabelle 3: Transformer-Encoder Hyperparameter (Basismodell)

7.2.4 Transformer-Decoder

Der Decoder erzeugt die SAN-Zeichensequenz autoregressiv. Er erhält als Eingabe die bisher erzeugten Tokens (beim Training via Teacher Forcing), wendet eine Causal Mask an und greift via Cross-Attention auf den Encoder-Output zu. Die relevanten Hyperparameter des Decoders sind in Tabelle 4 dargestellt.

Parameter	**Wert**
Embedding Dimension	256
Anzahl Decoder-Layers	1
Anzahl Attention-Heads	4
Feed-Forward Dimension	1024
Vokabulargrösse	30 (3 Sonder-Tokens `<pad>`, `<sos>`, `<eos>` + 5 Figuren + 8 Linien + 8 Reihen + 6 SAN-Symbole: `x + # = 0 -`)
Maximale Sequenzlänge	10 (`<sos>` + bis zu 7 Zeichen + `<eos>` + Puffer)

Tabelle 4: Transformer-Decoder Hyperparameter (Basismodell)

7.2.5 Training-Setup

Für das Training des Basismodells wurden die Hyperparameter in Tabelle 5 verwendet.

Parameter	**Wert**
Optimizer	AdamW
Learning Rate	3×10^{-4}
Weight Decay	10^{-4}
Batch Size	64
Cross Entropy Loss	Label Smoothing mit 10^{-2}
Gradient Clipping	1.0
Precision	16-mixed
Early Stopping	val_char_acc ändert sich nicht mehr als 10^{-3} für 10 Epochen
Bestes Modell	Höchster val_ply_acc Wert

Tabelle 5: Training-Hyperparameter (Basismodell)

7.2.6 Gesamtarchitektur

Das Basismodell umfasst insgesamt 3 019 422 trainierbare Parameter, aufgeteilt wie in Tabelle 6 dargestellt.

Komponente	**Parameter**
CNN-Backbone	370 560
Encoder-Layer Norm	512
Transformer-Encoder	1 579 520
Transformer-Decoder	1 053 440
Token Embedding	7 680
Output Projection	7 710
Total	**3 019 422**

Tabelle 6: Parameteranzahl des Basismodells

Alles zusammengeführt ergibt die Architektur in Abbildung 10. Diese diente als Ausgangslage für alle in Kapitel 8 beschriebenen Experimente.

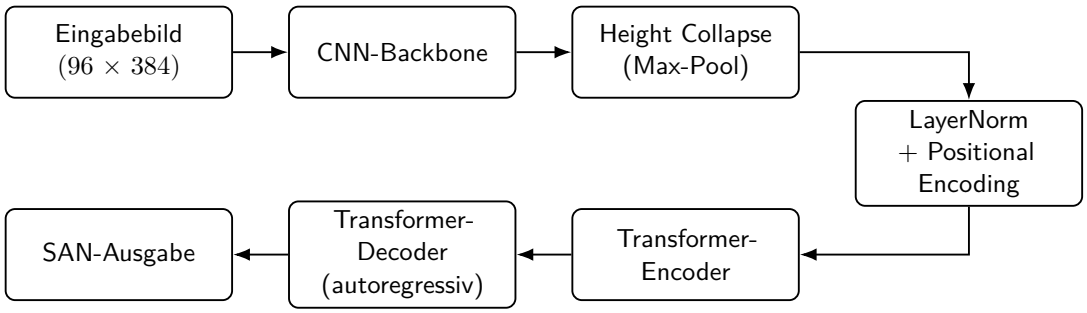


Abbildung 10: Gesamtarchitektur des Transformer-Encoder-Decoder-Modells

8 Experimente

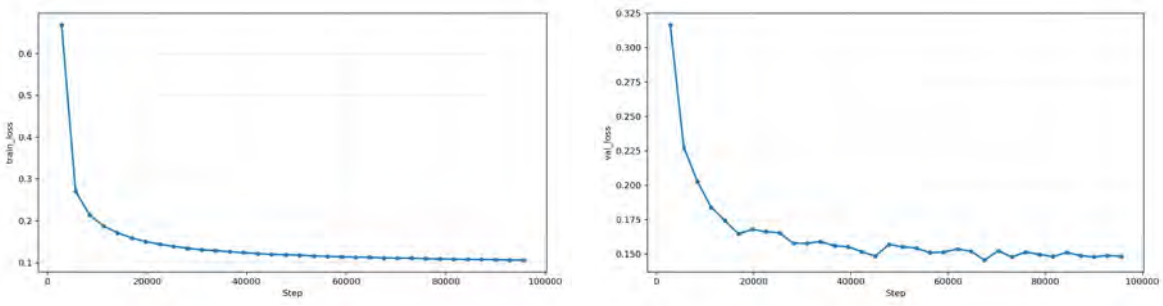
Im Rahmen dieser Arbeit wurden verschiedene Experimente durchgeführt, um die in Kapitel 7 beschriebene Basisarchitektur schrittweise weiterzuentwickeln und zu evaluieren. Die Experimente bauen chronologisch aufeinander auf. Die Erkenntnisse aus einem Experiment flossen direkt in den nächsten Schritt ein. Als gemeinsame Grundlage aller Experimente dient das in Kapitel 7.2 definierte Basismodell.

8.1 Transformer-Encoder-Decoder

Die folgenden Abschnitte dokumentieren die Experimente mit dem Transformer-Encoder-Decoder-Modell. Ausgangspunkt ist das Training auf dem synthetischen Datensatz *SYN-200K*, gefolgt von gezielten Anpassungen an Daten, Architektur und Trainingsstrategien.

8.1.1 Baseline: Training auf *SYN-200K*

Das Basismodell wurde zunächst auf dem *SYN-200K*-Datensatz trainiert, wie in Abschnitt 6.3.3 beschrieben. Das Training verlief stabil, der Train Loss fiel in den ersten Schritten rasch von ungefähr 0.67 auf unter 0.20 und konvergierte anschliessend bei etwa 0.11 (Abbildung 11a). Die Validation Loss folgte diesem Verlauf und stabilisierte sich bei rund 0.15, ohne Anzeichen von Overfitting (Abbildung 11b).



(a) Train Loss (b) Validation Loss

Abbildung 11: Loss-Kurven des Basismodells (Training auf *SYN-200K*).

Die Evaluation mittels der bestehenden Evaluationspipeline des ChessReader-Projekts ergab eine Character Accuracy von 75.3 % und eine Ply Accuracy von 48.6 %. Eine eigene Evaluation auf *CR-v6-EN* stand zu diesem Zeitpunkt noch nicht zur Verfügung.

Das Resultat war nicht zufriedenstellend und eine vertiefte Fehleranalyse des Modells erschien wenig zielführend. Der synthetische Generator verwendet eine frequenzbasierte Gewichtung bei der Zugauswahl [24]. Häufige Züge wie *Nf3* besitzen eine Sampling-Gewichtung von 29 266, seltene Züge wie *Q5a4#* hingegen eine Gewichtung von 1. Bei 200 000 Trainingsbildern kommen statistisch seltene Züge kaum oder gar nicht vor, sodass das Modell viele relevante SAN-Muster nicht erlernen konnte. Da eine tiefergehende Fehleranalyse ohne vollständige Move Coverage wenig aufschlussreich gewesen wäre, wurde als nächster Schritt der *SYN-FC*-Datensatz entwickelt.

8.1.2 Wechsel zu *SYN-FC*

Der *SYN-FC*-Datensatz (Full Cycles) wurde entwickelt, um das Abdeckungsproblem von *SYN-200K* zu beheben. Der Aufbau und die Generierung des Datensatzes sind in Abschnitt 6.3.4 beschrieben.

Im Zuge der Datenvorbereitung wurde zudem die verwendeten Schriftarten bereinigt. Bei einer Sichtprüfung der generierten Bilder wurden mehrere Schriftarten identifiziert, die für das Training ungeeignet waren. Die Probleme reichten von nicht vorhandenen Symbolen in der Schriftart über spezielle Sonderzeichen bis hin zu sehr kleiner Schrift im Verhältnis zur Box. Zudem wurden einzelne Zeichen durch vermeintlich ähnliche ersetzt, was bei einer echten Handschrift zwar funktioniert, bei synthetischen Daten jedoch zu Verwirrung führt.

Tabelle 7 zeigt exemplarisch ein paar generierte Bilder, welche zu Problemen führten.

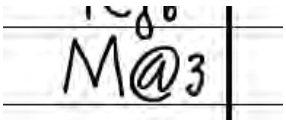
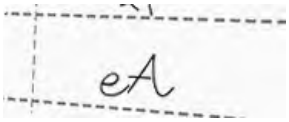
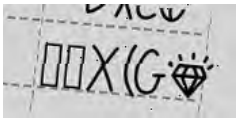
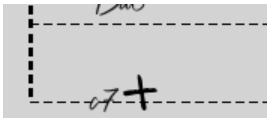
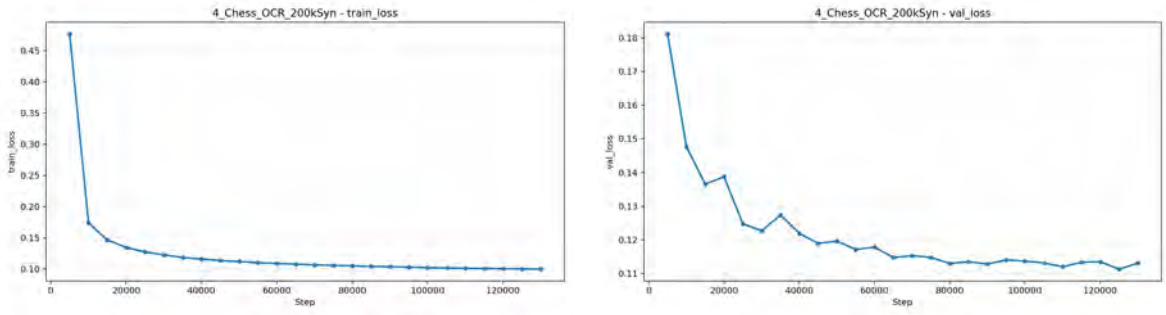
Bild	Schachzug	Problem
	Na3	Vertauschungen: *M* statt *N* und *@* statt *a*
	e4	Vertauschung: *A* statt *4*
	Nb6+	*b* ist in Schnörkelschrift
	Bxc6+	Vertauschungen: *(* statt *c* und *G* statt *6* Zeichen nicht vorhanden: *B* Spezielle Zeichen: Diamant statt *+*
	c7+	Schriftart ist sehr klein
	Rg3	Es sind zwei Boxen auf dem Bild

Tabelle 7: Beispiele von problematischen Schriftarten und getauschten Zeichen

Das Training auf *SYN-FC* verlief stabil, wie in Abbildung 12 ersichtlich. Die Evaluation auf der bestehenden Pipeline ergab eine Ply Accuracy von 35.65 % und eine Character Accuracy von 58.64 %. Damit liegt das Modell deutlich unter der Baseline das eine Ply Accuracy von 48.6 % aufweist.



(a) Train Loss (b) Validation Loss

Abbildung 12: Loss-Kurven des Modells trainiert auf *SYN-FC*.

Der Rückgang gegenüber der Baseline ist zunächst überraschend, lässt sich jedoch erklären. *SYN-200K* basiert auf einer frequenzbasierten Zugverteilung, die der realen Zugverteilung auf handschriftlichen Scoresheets ähnelt. Es ist anzunehmen, dass der Decoder dadurch implizit diese Wahrscheinlichkeiten erlernt hat, was ihm ermöglichte, auf realen Daten vergleichsweise gut zu performen. *SYN-FC* hingegen enthält keine solche Verteilung, da alle Züge gleichmässig vertreten sind. Ohne diese implizite Vorinformation muss der Decoder die Ausgabe allein aus den visuellen Features ableiten, was die schlechteren Resultate erklären könnte.

Ein zentrales Problem bei *SYN-FC* ist eine ausgeprägte Character Confusion, die in der Confusion Matrix (Abbildung 13) sichtbar wird. Das Modell sagt systematisch zu viele Zeichen voraus. Statt eines einfachen Zugs wie *Nf3* wird beispielsweise *Nf3+* oder *Nxf3* vorrausgesagt. Die Ursache liegt in der Struktur von *SYN-FC*. Da der Generator alle Zugvarianten gleichmässig abdeckt, gibt es im Datensatz deutlich mehr Einträge mit folgenden Symbolen +, #, x, etc. Ausserdem kommen Züge wie *Ndf3* oder *N1f3* öfters vor als z.B. *Nf3*. Das Modell lernt dadurch die Tendenz, zu oft Symbole hinzuzufügen, wie beispielsweise ein Plus für einen in Schach gesetzten König (was pro Spiel nicht so häufig vorkommt).

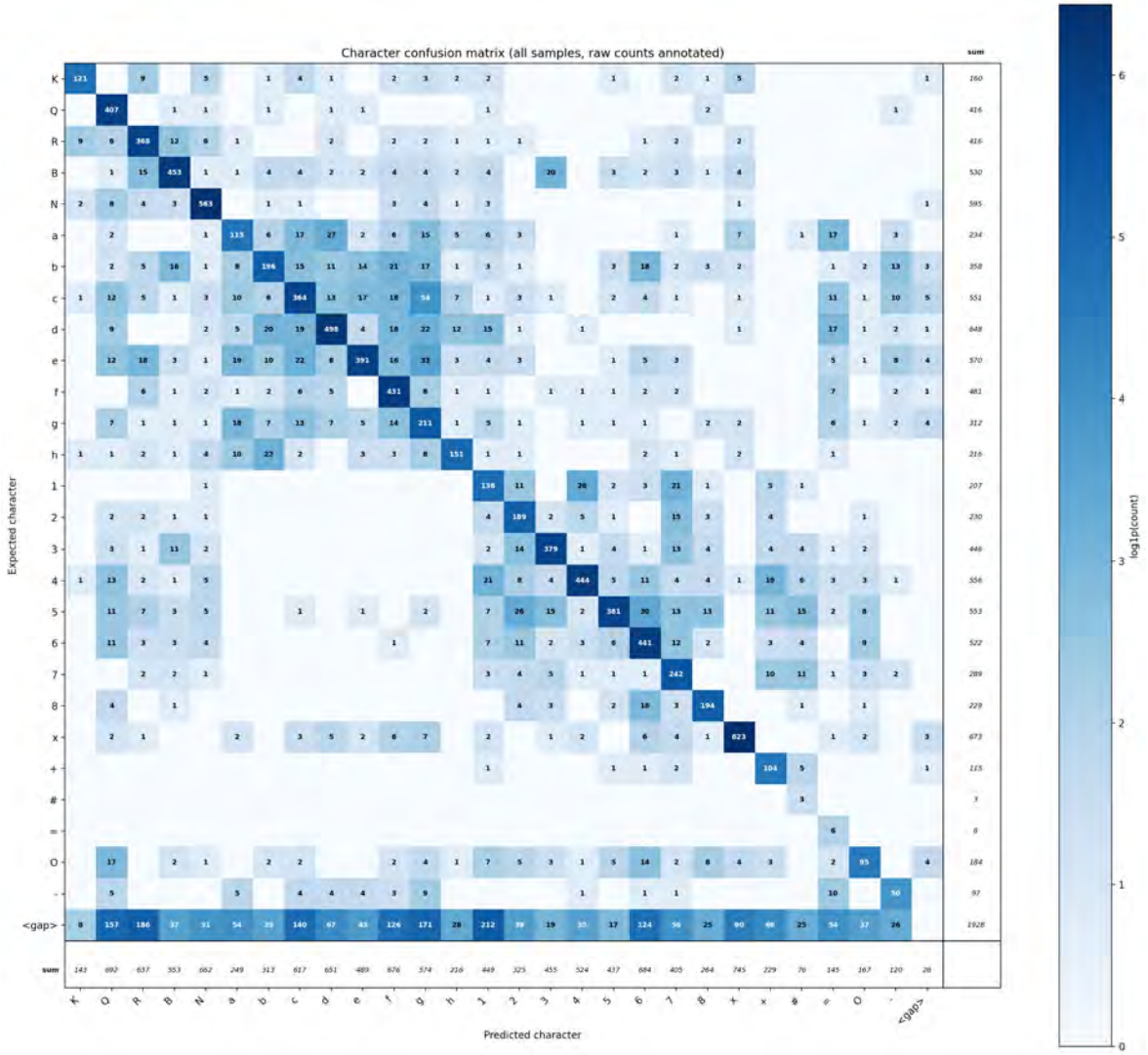


Abbildung 13: Character Confusion Matrix des auf *SYN-FC* trainierten Modells

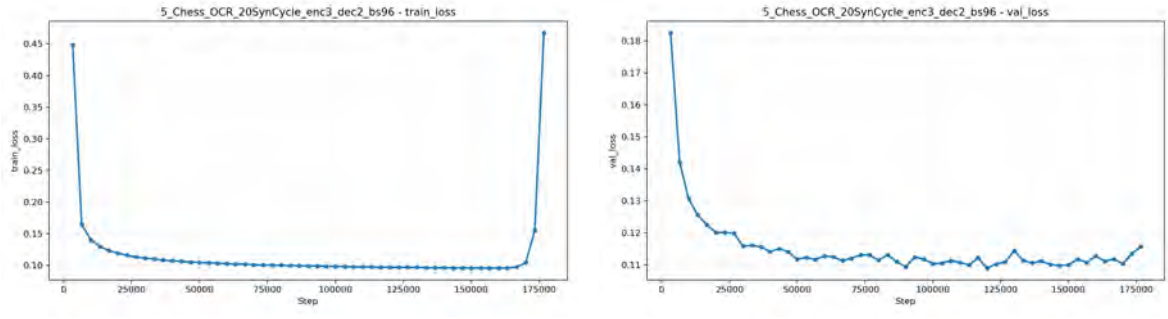
8.1.3 Mehr Encoder- und Decoder-Layers

Als nächster Schritt wurde die Modellkapazität erhöht. Das neue Modell verwendet drei Encoder-Layers und zwei Decoder-Layers, es trainiert ebenfalls auf dem *SYN-FC*-Datensatz. Durch die zusätzlichen Schichten erhöht sich die Parameteranzahl von 3 019 422 auf 4 862 622, wobei der Zuwachs vollständig auf die grösseren Encoder- und Decoder-Layers entfällt (Tabelle 8).

Komponente	**Parameter**
CNN-Backbone	370 560
Encoder-Layer Norm	512
Transformer-Encoder	2 369 280
Transformer-Decoder	2 106 880
Token Embedding	7 680
Output Projection	7 710
Total	**4 862 622**

Tabelle 8: Parameteranzahl des Modells (3 Encoder, 2 Decoder)

Das Training verlief weitgehend stabil, weist jedoch am Ende einen auffälligen Loss-Spike auf (Abbildung 14a). Etwa bei Schritt 175 000 steigt die Train Loss kurz von ca. 0.10 auf rund 0.46 an. Die genaue Ursache dieses Spikes konnte auch in Absprache mit dem Betreuer nicht abschliessend geklärt werden. Da der Spike einmalig auftrat, wurde dieser nicht als kritisch eingestuft und nicht weiter untersucht. Die Evaluation ergab eine Ply Accuracy von 43.43 % und eine Character Accuracy von 63.81 %, was eine klare Verbesserung gegenüber dem vorherigen Modell darstellt.



(a) Train Loss (b) Validation Loss

Abbildung 14: Loss-Kurven des Modells (3 Encoder, 2 Decoder) trainiert auf *SYN-FC*

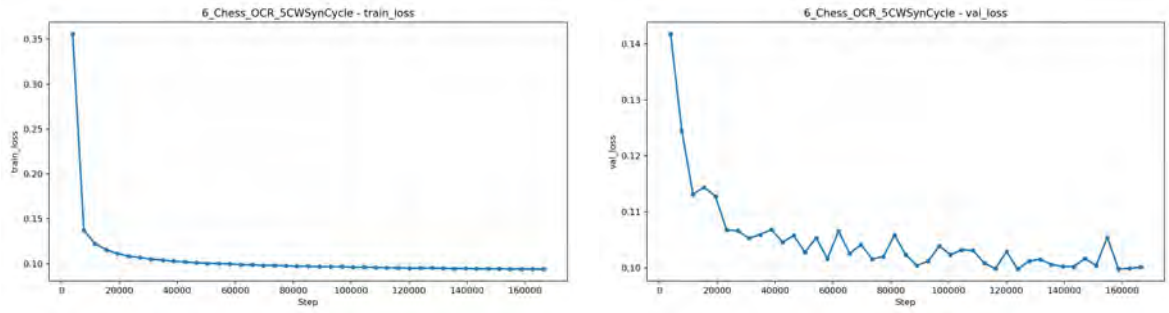
Da die Datengrundlage dieselbe ist, zeigt das Modell dieselbe Character Confusion wie in Abschnitt 8.1.2 beschrieben. Die Verbesserung der Metriken ist allein auf die erhöhte Modellkapazität zurückzuführen und zeigt, dass das Modell trotz des nach wie vor unvollständigen Datensatzes von mehr Parametern profitiert.

8.1.4 Gewichteter Datensatz und Finetuning mit realen Daten

Als nächster Schritt wurde der *SYN-WC*-Datensatz (Weighted Cycles) eingeführt und ein Pretraining-Finetuning-Ansatz verfolgt. Das zentrale Problem von *SYN-FC* lag in der unrealistischen Gleichverteilung aller Züge. Seltene SAN-Formen waren überrepräsentiert, was das Modell auf unnatürliche Häufigkeiten konditionierte. *SYN-WC* begegnet diesem Problem, indem vollständige Zugabdeckung mit einer frequenzbasierten Gewichtung kombiniert wird, sodass die Verteilung der realen Zugverteilung näher kommt. Für das Finetuning wurde der *HCS*-Datensatz verwendet, dieser umfasst rund 10 000 Schachzüge. Um die Datenmenge und Vielfalt zu vergrössern wurde jeder Zug viermal augmentiert. Dabei wurden zufällig angewendete Transformationen wie affine Transformationen, perspektivische Verzerrungen, Helligkeits- und Kontrastanpassungen sowie leichte Änderungen der Bildschärfe eingesetzt.

Aufbau und Generierung des Datensatzes sind in Abschnitt 6.3.5 beschrieben.

Pretraining auf *SYN-WC*. Das Modell wurde zunächst ohne reale Daten auf dem *SYN-WC*-Datensatz trainiert. Das Training verlief stabil (Abbildung 15). Die Evaluation ergab eine Ply Accuracy von 54.00 % und eine Character Accuracy von 74.99 %. Gegenüber dem Basismodell auf *SYN-200K* (48.6 % Ply Accuracy) ist dies eine deutliche Verbesserung und auch gegenüber dem vorherigen Modell auf *SYN-FC* (43.43 % Ply Accuracy) zeigt sich ein klarer Fortschritt.



(a) Train Loss (b) Validation Loss

Abbildung 15: Loss-Kurven des Pretrainings auf *SYN-WC*

Die Character Confusion (Abbildung 16) ist gegenüber *SYN-FC* deutlich reduziert, aber noch nicht vollständig behoben. Das Modell tendiert weiterhin dazu, gelegentlich zusätzliche Zeichen vorherzusagen, jedoch weniger ausgeprägt als zuvor. Die Gewichtung in *SYN-WC* bringt häufige Züge wie *Nf3* öfter ohne zusätzliche Zeichen vor, was die Verteilung realistischer macht und die Confusion abschwächt.

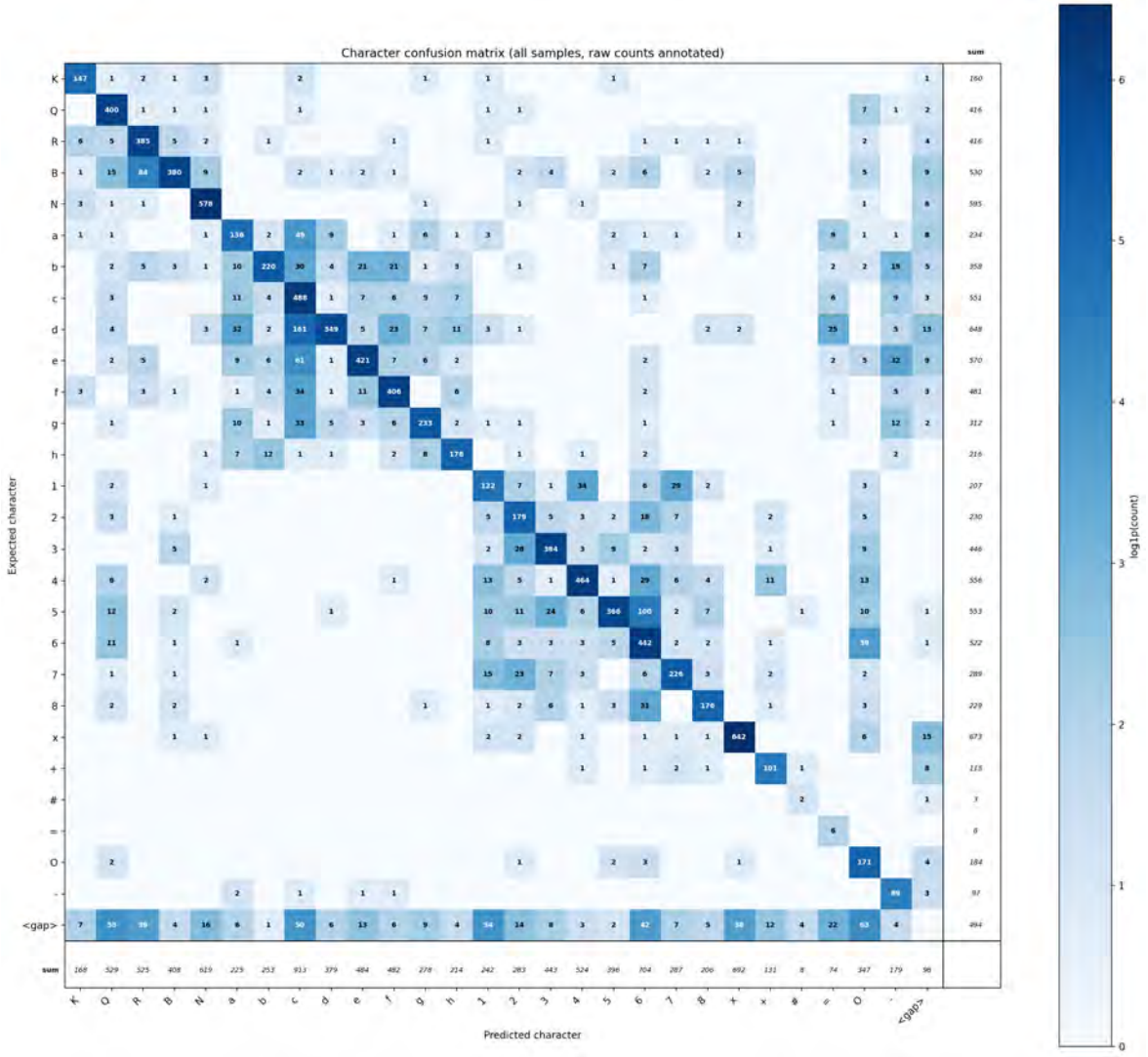
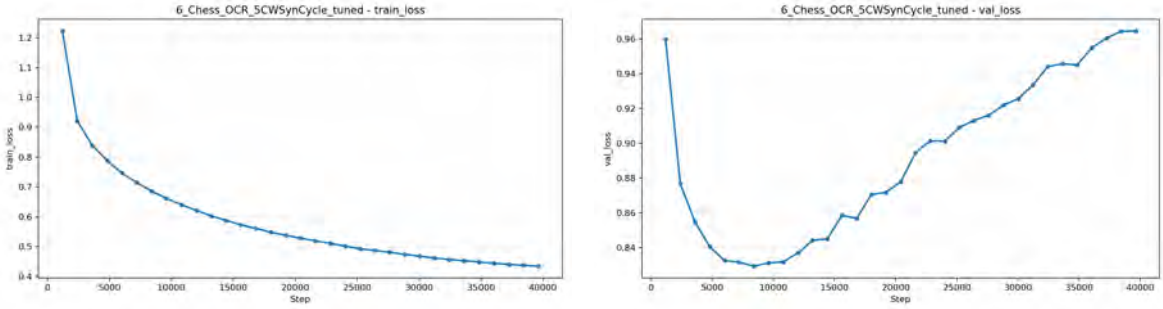


Abbildung 16: Character Confusion Matrix nach dem Pretraining auf *SYN-WC*

Finetuning mit realen Daten. Im zweiten Schritt wurde das vortrainierte Modell auf realen handschriftlichen Daten des *HCS*-Datensatzes finegetuned. Die Evaluation nach dem Finetuning ergab eine Ply Accuracy von 73.80 % und eine Character Accuracy von 88.44 %, was einem erheblichen Sprung gegenüber dem rein synthetisch trainierten Modell entspricht.

Auffällig ist der Verlauf der Validation Loss beim Finetuning (Abbildung 17). Nach einer ersten Abnahme beginnt sie wieder zu steigen, obwohl Ply und Character Accuracy über denselben Zeitraum stabil bleiben. Eine mögliche Erklärung ist ein Domain-Shift. Das Modell passt sich zunehmend an die Charakteristiken der realen Trainingsdaten an, wodurch die Sicherheit der Vorhersagen auf den Validierungsdaten abnimmt. Da das Modell bei der Selektion nach Ply Accuracy bewertet wurde und nicht nach der Validation Loss, hat dieser Anstieg keinen Einfluss auf das gespeicherte beste Modell.



(a) Train Loss (b) Validation Loss

Abbildung 17: Loss-Kurven des Finetunings auf *HCS*

Nach dem Finetuning ist die Tendenz zur Vorhersage überzähliger Zeichen nahezu vollständig verschwunden (Abbildung 18). Das Modell lernt aus den realen Beispielen, die länge eines Zugs und produziert kaum noch überflüssige Zeichen. Übrig bleiben kleinere, typische OCR-Fehler wie die Verwechslung von bestimmten Zeichen.

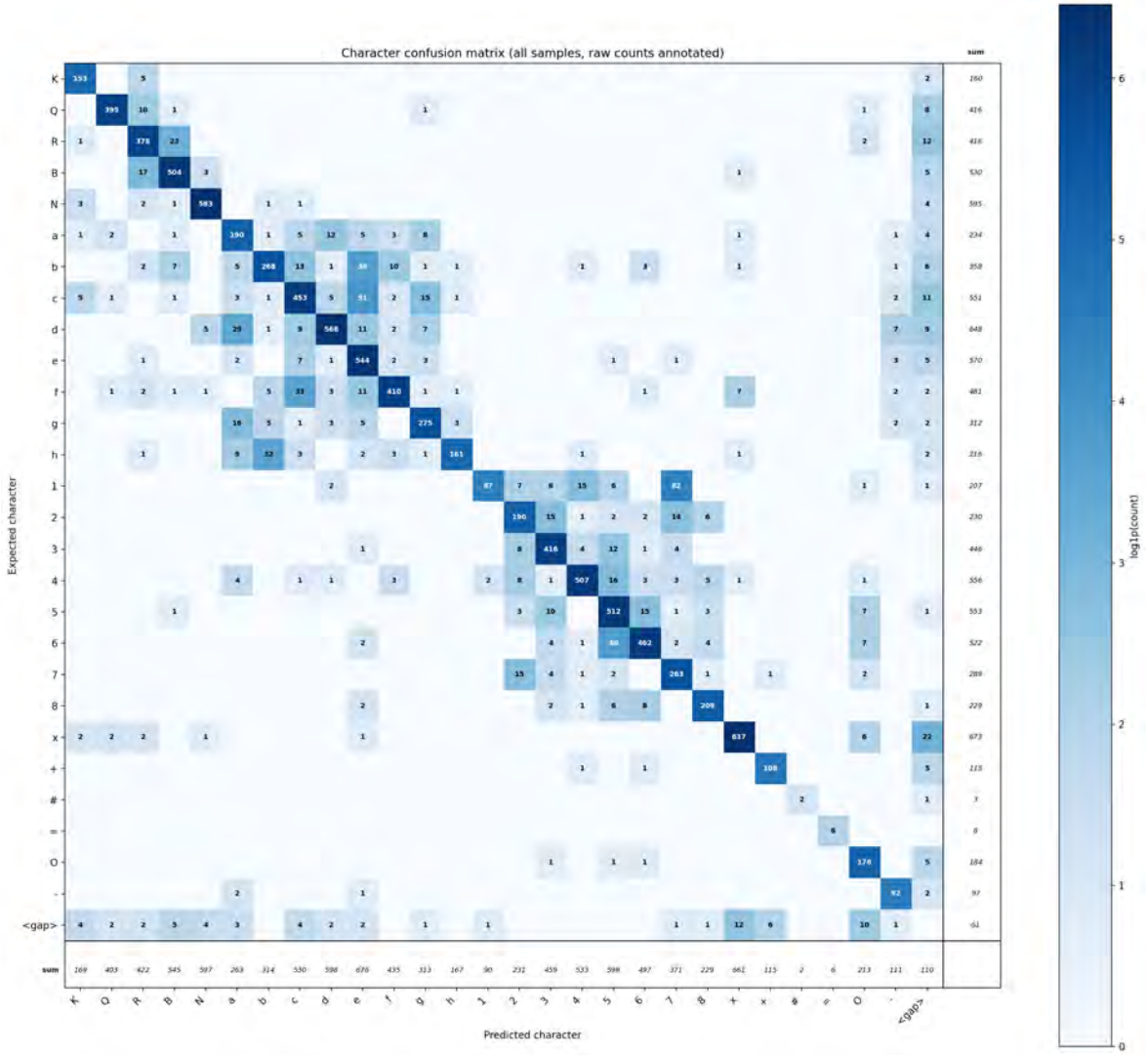


Abbildung 18: Character Confusion Matrix nach dem Finetuning auf *HCS*

8.1.5 Erhöhung der Filtergrösse

Die Character Confusion Matrix des Fine-Tuning-Modells zeigte trotz der Verbesserungen eine breite Streuung von Fehlklassifikationen über viele Zeichenpaare. Insbesondere wurden Gap-Tokens in beide Richtungen mit echten Zeichen verwechselt, und Buchstaben sowie Ziffern untereinander fehlerhaft klassifiziert. Eine mögliche Ursache liegt in der Filtergrösse von 3×3. Diese kleinen Filter erfassen nur sehr lokale Bildmerkmale und können die Gesamtform eines Zeichens nicht direkt abbilden. Handschriftliche Schachzüge unterscheiden sich häufig erst auf einer grösseren räumlichen Skala, beispielsweise durch die globale Strichführung eines Buchstabens. Daher wurden alle Convolutional Layers von 3×3 auf 7×7 Filterkerne umgestellt, bei gleichbleibendem Padding, sodass die räumlichen Dimensionen erhalten bleiben.

Eine grössere Filtergrösse erhöht die Anzahl lernbarer Parameter in jeder Conv-Schicht erheblich. Ein 7×7-Filter besitzt 49 Gewichte gegenüber 9 bei 3×3, was einem Faktor von etwa $5{,}4$ entspricht. Die Gesamtanzahl der Modellparameter stieg damit auf $6'503'582$.

Ab diesem Punkt werden nur noch die Ergebnisse nach dem Finetuning auf dem *HCS*-Datensatz berichtet, da das Pretraining allein keine ausreichenden Ergebnisse liefert. Tabelle 9 zeigt den Vergleich zum vorherigen Modell.

Modell	**Ply Accuracy**	**Character Accuracy**
Filter 3×3 (Finetuned)	73.80 %	88.44 %
Filter 7×7 (Finetuned)	**84.35 %**	**94.02 %**

Tabelle 9: Ergebnisse vor und nach Erhöhung der Filtergrösse auf 7×7.

Die Erhöhung der Filtergrösse brachte eine substanzielle Verbesserung von $+10.55$ % in der Ply Accuracy und $+5.58$ % in der Character Accuracy. Grössere Filter erlauben es dem Modell, die globale Form eines Zeichens direkt zu erfassen, was bei handschriftlichen Eingaben wichtig zu sein scheint.

Dieses Modell diente als Basis für alle weiteren Experimente.

Die Character Confusion Matrix (Abbildung 19) zeigt ebenfalls eine Reduktion der Fehlklassifikationen. Dies insbesondere bei den Gap-Tokens, die nun deutlich seltener mit echten Zeichen verwechselt werden.

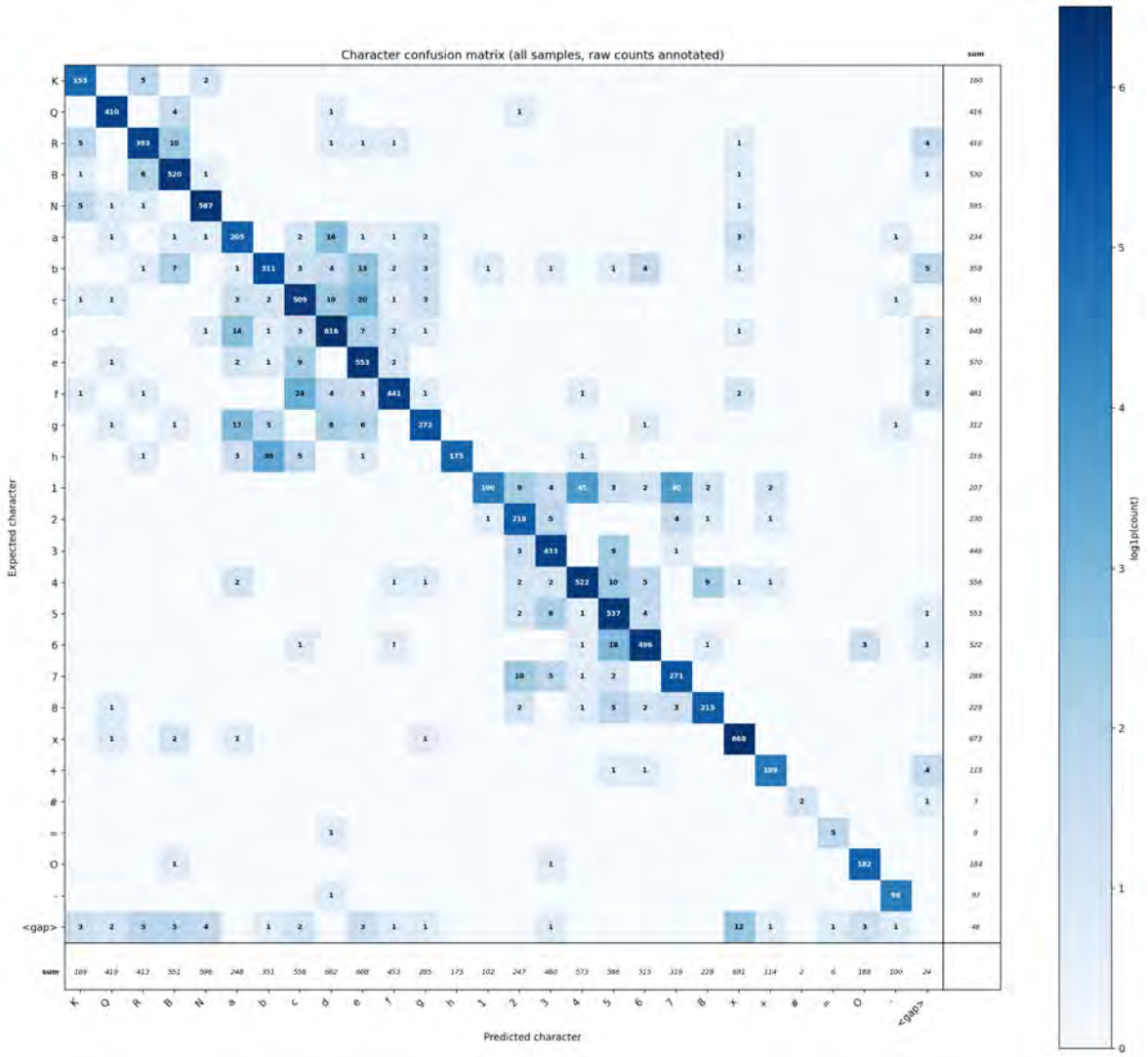


Abbildung 19: Character Confusion Matrix nach dem Finetuning mit 7×7 Filtergrösse

8.1.6 Weitere Experimente ohne signifikante Verbesserung

Nach der Etablierung der 7×7-Filterbasis wurden mehrere architekturelle Varianten untersucht, die allesamt schlechtere Resultate als das neue Basismodell lieferten. Der Vollständigkeit halber werden diese Experimente im Folgenden kurz aufgelistet und mögliche Ursachen diskutiert.

Mehr Transformer-Schichten (5 Encoder- & 5 Decoder-Layer). In Anlehnung an grössere Sprachmodelle wurde die Anzahl der Transformer-Schichten auf jeweils fünf Encoder- und fünf Decoder-Layer erhöht. Die damit verbundene deutliche Zunahme an Modellparametern führte jedoch zu keiner Verbesserung. Eine plausible Erklärung ist Overfitting. Der *HCS*-Datensatz umfasst eine begrenzte Anzahl realer Beispiele, die für ein so tiefes Modell wohl nicht ausreichen, um alle Parameter sinnvoll zu trainieren. Zudem ist die Aufgabe, eine SAN-Sequenz von nur wenigen Zeichen zu erkennen, nicht komplex genug, um von dieser Tiefe zu profitieren.

Grössere Hidden Dimension und mehr Attention-Heads. Experimentiert wurde mit einer Hidden Dimension von 384 und sechs Attention-Heads anstelle von 256 und vier. Auch hier blieb die Verbesserung aus. Wie beim tieferen Transformer ist Overfitting eine wahrscheinliche Ursache. Der erhöhten Modellkapazität steht ein kleinerer Finetuning-Datensatz gegenüber. Zusätzlich ist fraglich, ob die visuellen Features des CNN-Backbones von einer höherdimensionalen Projektion profitieren, da die informativen Bildmerkmale handschriftlicher Schachzüge strukturell einfach sind.

Vierte CNN-Schicht mit variabler Filtergrösse. Untersucht wurde ein Backbone mit vier statt drei Convolutional Blocks, wobei die Filtergrössen von 7×7 über 5×5 auf 3×3 abnehmend gesetzt wurden. Die Idee war, grobe Features mit grossen Filtern zu erfassen und sie mit kleineren Filtern zu verfeinern. Die Resultate verschlechterten sich gegenüber dem Basismodell. Eine mögliche Ursache ist, dass ein vierter Pooling-Block die Sequenzlänge von 96 auf 48 visuelle Tokens halbiert, wodurch dem Transformer weniger räumliche Information zur Verfügung steht. Zusätzlich könnte die 3×3-Filtergrösse in der letzten CNN-Schicht das Lokalitätsproblem zumindest teilweise wieder einführen, das durch die Erhöhung auf 7×7 überwunden worden war.

Mean Collapse anstelle von Max Collapse. Als Alternative zum Max-Pooling über die Höhendimension wurde ein Mean-Pooling untersucht, das ebenfalls schlechtere Resultate lieferte. Es wird vermutet, dass Mean-Pooling die starken, aber spärlichen Aktivierungen der Schriftzeichen mit schwachen Hintergrundaktivierungen mittelt und das Signal dadurch abschwächt. Max-Pooling hingegen selektiert pro Spalte die stärkste Aktivierung, was für Text auf hellem Hintergrund vorteilhafter sein dürfte.

8.2 CTC Decoding

Nachdem die letzten paar Versionen keine Verbesserungen hervorgebracht hatten, zeigte sich, dass mit der Architektur des Transformer-Decoder möglicherweise eine Grenze erreicht war. Daher wurde beschlossen zusätzlich die Architektur eines CTC-Modells zu untersuchen. Das bedeutet ein Modell, welches auf einem ähnlichen Encoder basiert, jedoch auf den autoregressiven Decoder verzichtet. Anstelle von diesem wird die Ausgabe über einen CTC-Head vorhergesagt.

8.2.1 Architektur-Änderungen CTC-Modell

Die Tabelle 10 zeigt die wichtigsten Architektur-Änderungen vom Transformer-Encoder-Decoder zum Encoder-Modell mit CTC-Head.

Komponente	**Transformer**	**CTC**
Encoder-Layers	3	6
Decoder-Layers	2	0
Vokabular	30 Token	28 Token
Vokabular-Tokens	`[PAD]`, `[SOS]`, `[EOS]`, ...	`[BLANK]`, ...
Sequenzen autoregressiv	Ja	Nein (parallel)

Tabelle 10: Architektur-Änderungen vom Encoder-Decoder zum Encoder-Only Modell

Entfernung des Decoders und Vokabular-Anpassung. Der zweischichtige Decoder wurde vollständig entfernt. Das ursprüngliche Vokabular verwendete 30 Tokens inklusive Spezial-Tokens:

- `[PAD]` für Padding
- `[SOS]` (Start-of-Sequence) für Sequenzbeginn
- `[EOS]` (End-of-Sequence) für Sequenzende
- 27 SAN-Zeichen (5 Pieces, 8 Files, 8 Ranks, 6 SAN-Symbole)

Das neue CTC-Vokabular wurde auf 28 Tokens reduziert:

- `[BLANK]` für CTC Blank-Token (intern für Alignment)
- 27 SAN-Zeichen (identisch)

Die speziellen Tokens `[PAD]`, `[SOS]`, `[EOS]` sind für CTC nicht nötig, da CTC intern mit dem Blank-Symbol arbeitet und keine sequenzielle Decodierung benötigt.

Kompensation durch zusätzliche Encoder-Layer. Da der Wegfall von Decoder, Token-Embedding und den Spezial-Tokens die Parameteranzahl erheblich reduziert hätte, wurde die Anzahl der Encoder-Layer von drei auf sechs erhöht. Damit wird eine vergleichbare Gesamtparameterzahl wie beim Transformer-Modell erreicht, was einen fairen architektonischen Vergleich ermöglicht wie die Tabelle 11 zeigt.

Komponente	**Transformer**	**CTC**
CNN Backbone (CNNStem)	2,011,520	2,011,520
Encoder LayerNorm	512	512
Transformer-Encoder-Layers (3 vs. 6)	2,369,280	4,738,560
Transformer-Decoder-Layers (2)	2,106,880	-
Token Embedding	7,680	-
Output Projection (30 vs. 28 Token)	7,710	7,196
GESAMT	**6,503,582**	**6,757,788**
Differenz: $254{,}206$ Parameter ($+3.9\,\%$)		

Tabelle 11: Parameter-Vergleich Transformer vs. CTC

Mit $6{,}757{,}788$ Parametern liegt das CTC-Modell nur geringfügig über dem Transformer-Modell.

8.2.2 Ergebnisse CTC-Modell

Das CTC-Modell wurde identisch zum Transformer-Modell trainiert und auf dem *HCS*-Datensatz finegetuned. Tabelle 12 zeigt den direkten Vergleich beider Architekturen.

Modell	**Ply Accuracy**	**Character Accuracy**
Transformer (7×7 Filter)	84.35 %	94.02 %
CTC (7×7 Filter, 6 Enc.-Layer)	**83.80 %**	**94.07 %**

Tabelle 12: Vergleich Accuracy Transformer vs. CTC

Die beiden Architekturen schneiden nahezu identisch ab. Der Transformer erzielt eine um 0.55 % höhere Ply Accuracy, während das CTC-Modell bei der Character Accuracy um 0.05 % vorne liegt. Angesichts dieser geringen Unterschiede lässt sich keine klare Überlegenheit einer der beiden Architekturen ableiten.

8.3 Untersuchung von Wichtigkeit der Datensätze

Als weiteres Experiment wurde untersucht, ob es wertvoll ist, mehr reale Trainingsdaten zu sammeln. Das Pretraining mit synthetischen Daten erreichte bereits eine hohe Genauigkeit bei der Validierung von über 99 %. Allerdings basiert diese Validierung auf synthetischen Daten und ist daher nur eingeschränkt aussagekräftig. Mangels ausreichend verfügbarer realer Daten ist es auch nicht möglich, die Validierung auf realen Daten durchzuführen. Die realen Daten werden für das Training und die Validierung beim Finetuning verwendet.

Das Ziel ist es, die Wichtigkeit von realen Daten zu untersuchen, um die Modellqualität zu verbessern.

8.3.1 Finetuning mit verschiedenen Datenmengen

Um zu untersuchen, inwieweit mehr reale Trainingsdaten zu besseren Modellen führen, wurde der *HCS*-Datensatz in drei weitere Subsets aufgeteilt. Diese enthalten jeweils 25 %, 50 % oder 75 % des gesamten Datensatzes. Mit dem ursprünglichen und den drei neuen Datensätzen wurde das Finetuning auf beiden Modellarchitekturen (Transformer und CTC) durchgeführt.

Transformer-Encoder-Decoder Finetuning-Resultate. Die Finetuning-Experimente mit dem Transformer-Modell zeigten eine sichtbare Abhängigkeit von der Datenmenge:

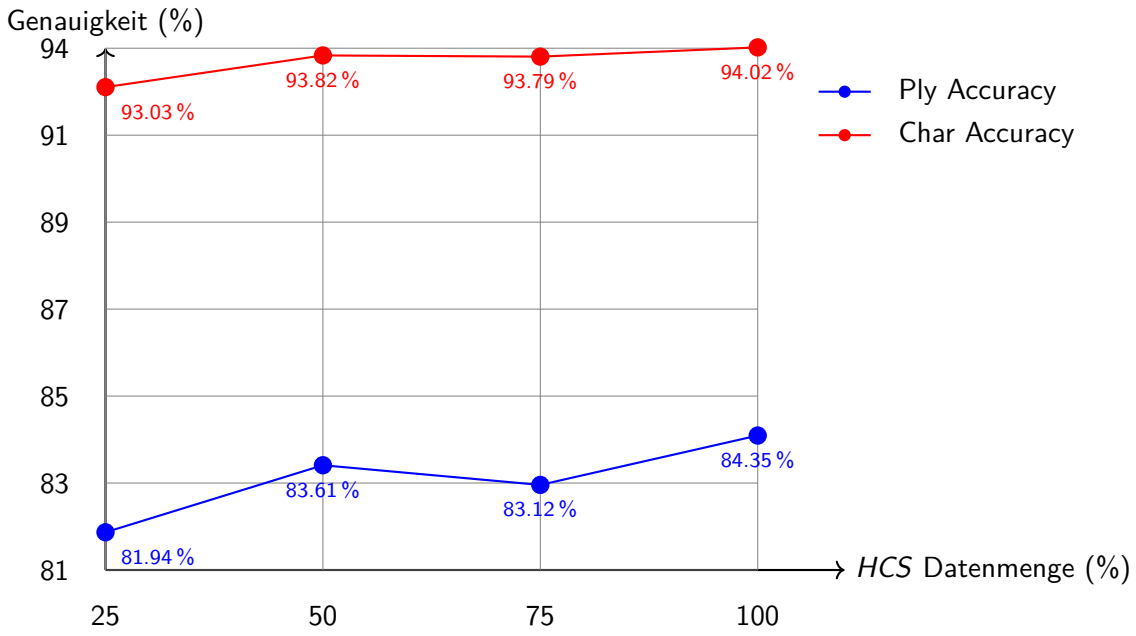


Abbildung 20: Ply und Character Accuracy beim Finetuning im Vergleich (Transformer-Modell)

Die Resultate bestätigen die Hypothese. Mit mehr realen Trainingsdaten steigt die Genauigkeit grundsätzlich an. Der Anstieg von 25 % auf 100 % der Daten beträgt 2.41 % bei der Ply Accuracy und 0.99 % bei der Character Accuracy. Dies zeigt, dass das Modell von zusätzlichen realen Trainingsdaten profitiert, auch wenn das Potenzial irgendwo eine Grenze erreichen wird.

Was allerdings auffällt, ist das zwischen 50 % und 75 % der Daten sowohl Character wie auch Ply Accuracy stagniert bzw. sogar ganz wenig zurückgeht. Es zeigt sich also das eine Erhöhung der Datenmenge nicht immer zu einer Verbesserung führt. Dennoch ist der allgemeine Trend klar erkennbar. Dies vor allem dann, wenn man nur die Verdoppelung der Datenmenge betrachtet, also von 25 % auf 50 % und von 50 % auf 100 %, dort ist die Verbesserung deutlich sichtbar. Die Vermutung liegt nahe, dass es zu wenig neuen Daten zwischen 50 % und 75 % gab, um eine signifikante Verbesserung zu erzielen. Das bedeutet, dass die Zunahme der Genauigkeit mit der Datenmenge nicht linear verläuft.

CTC-Modell Finetuning-Resultate. Die Finetuning-Experimente mit dem CTC-Modell zeigten ebenfalls eine Abhängigkeit von der Datenmenge.

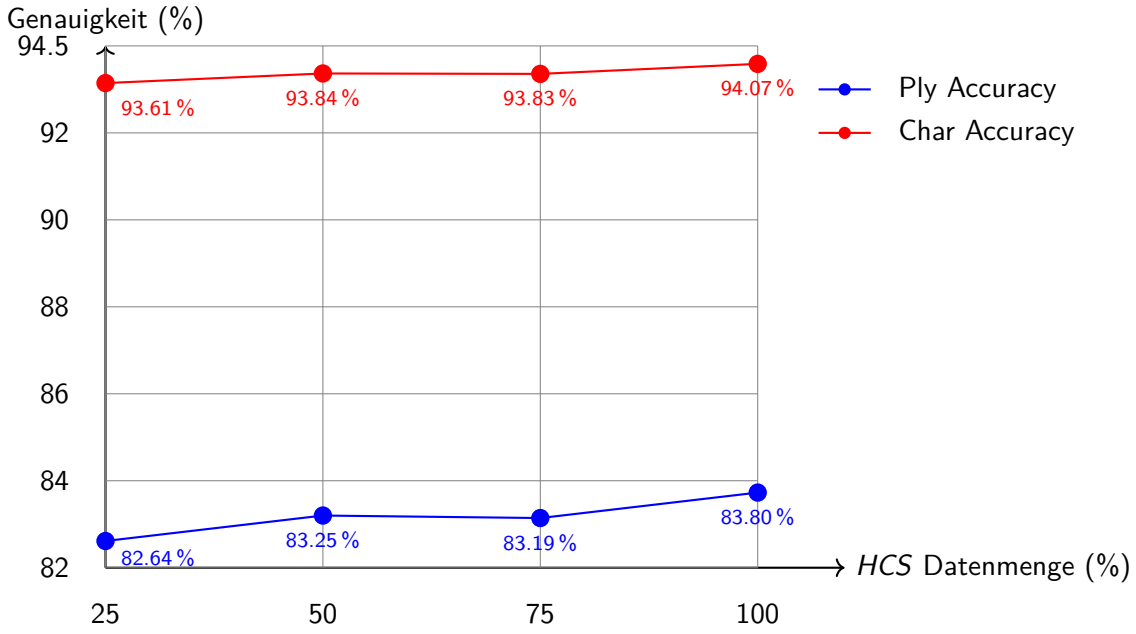


Abbildung 21: Ply und Character Accuracy beim Finetuning im Vergleich (CTC-Modell)

Das CTC-Modell zeigt insgesamt eine ähnliche Genauigkeit wie das Transformer-Modell, jedoch mit kleineren Verbesserungen durch zusätzliche Daten. Der Anstieg von 25 % auf 100 % Daten beträgt nur 1.16 % bei der Ply Accuracy und 0.46 % bei der Character Accuracy. Dies deutet darauf hin, dass das CTC-Modell bereits mit 25 % der Daten eine höhere Genauigkeit erreicht und von weiteren Daten weniger profitiert. Das gilt besonders bei der Character Accuracy, welche sich nur sehr wenig verbessert.

8.3.2 Pretraining auf realen Daten

Nebst der Untersuchung der Wichtigkeit von mehr realen Daten für das Finetuning, wurde auch untersucht, welchen Einfluss das Pretraining auf realen Daten gegenüber synthetischen Daten hat. Den bisher grössten Fortschritt bei der Genauigkeit konnte durch das Einführen von Finetuning auf realen Daten erzielt werden. Das beste Modell ist ohne Finetuning um etwa 30 bis 35 % schlechter als mit Finetuning. Daher stellte sich die Frage, ob es das Pretraining auf synthetischen Daten überhaupt braucht. Um dies zu untersuchen, wurde sowohl das Transformer-Modell als auch das CTC-Modell direkt auf realen Daten trainiert (mangels weiterer realer Daten, war dadurch kein Finetuning mehr möglich).

Experimentelle Einschränkungen und Vergleichsansatz. Für das Experiment zwischen dem Pretraining auf realen und synthetischen Daten ist es wichtig zu beachten, dass es einen wesentlichen Unterschied in den zur Verfügung stehenden Datenmengen gibt. Für synthetische Daten können potenziell unbegrenzt viele Bilder generiert werden. In dieser Arbeit wurden über 400 000 synthetische Bilder erzeugt (*SYN-WC*). Für reale handschriftliche Daten standen dagegen nur etwa 50 000 Bilder zur Verfügung.

Das Ziel bestand somit nicht primär in einem direkten Vergleich der Modelle, sondern in der Untersuchung der Relevanz synthetischer Daten. Zusätzlich sollte geprüft werden, wie gut synthetische Daten im Vergleich zu realen handschriftlichen Daten für das Training geeignet sind. Es wurde auf folgenden Daten trainiert (ohne Finetuning):

- Synthetischen Daten mit *SYN-WC*-Datensatz
- Realen Daten mit *HCS*-Datensatz

Pretraining Resultate (Transformer-Modell). Die Pretraining Experimente mit dem Transformer-Modell zeigten einen deutlichen Vorteil des realen Datensatzes:

Pretraining Daten	**Ply Accuracy**	**Char Accuracy**
Synthetisch (*SYN-WC*)	50.85 %	72.24 %
Real (*HCS*)	71.58 %	88.13 %
Differenz	**+20.73 %**	**+15.89 %**

Tabelle 13: Pretraining Resultate ohne Finetuning (Transformer-Modell)

Wie in Tabelle 13 ersichtlich, liefert Pretraining auf realen Daten deutlich bessere Ergebnisse als auf synthetischen Daten, obwohl die reale Datenmenge etwa 8-mal kleiner ist. Der Anstieg beträgt 20.73 % bei der Ply Accuracy und 15.89 % bei der Character Accuracy. Dies deutet darauf hin, dass reale Daten mit guter Qualität für das Pretraining wertvoller sind als viele synthetischen Daten, die möglicherweise eine schlechtere Qualität aufweisen.

Pretraining Resultate (CTC-Modell). Das Pretraining mit dem CTC-Modell zeigt ein ähnliches Verhalten, jedoch ist der Unterschied zwischen synthetischen und realen Daten kleiner:

Pretraining Daten	**Ply Accuracy**	**Char Accuracy**
Synthetisch (*SYN-WC*)	53.84 %	76.75 %
Real (*HCS*)	65.09 %	85.33 %
Differenz	**+11.25 %**	**+8.58 %**

Tabelle 14: Pretraining Resultate ohne Finetuning (CTC-Modell)

Wie in Tabelle 14 ersichtlich, zeigt sich auch beim CTC-Modell ein klarer Vorteil des Pretrainings auf realen Daten. Bei der Ply Accuracy erreicht das Training auf realen Daten 65.09 %, während synthetische Daten 53.84 % erzielen, dies entspricht einem Unterschied von 11.25 %. Bei der Character Accuracy zeigt sich ein ähnlich grosser Unterschied von 8.58 % (85.33 % für reale und 76.75 % für synthetische Daten).

Diese Ergebnisse unterstreichen, dass reale handschriftliche Daten, trotz ihrer geringeren Menge, für das Pretraining von beiden Modellarchitekturen bessere Ergebnisse liefern. Dieses Ergebnis der realen Daten deutet darauf hin, dass die Qualität und Authentizität der Trainingsdaten wichtiger ist als die pure Quantität synthetischer Bilder. Es unterstreicht, wie bereits beim Finetuning-Experiment aufgezeigt, dass reale Daten einen signifikanten Einfluss auf die Modellleistung haben können.

Es zeigt sich aber auch, dass die Kombination von Pretraining und Finetuning ein besseres Ergebnis liefert als nur das Pretraining, egal auf welchen Daten. Da, wie eingangs erwähnt, nicht genügend reale Daten für beide Trainingsphasen zur Verfügung standen, wurde für das Pretraining trotz der schlechteren Ergebnisse der synthetische Datensatz verwendet. Dadurch konnten die realen Daten für das Finetuning eingesetzt werden.

8.4 Gesamtübersicht der Ergebnisse

Die Tabelle 15 fasst alle Experimente mit ihren Ergebnissen zusammen. Die Modelle sind chronologisch nach ihrer Entwicklung geordnet und in zwei Gruppen unterteilt. Modelle, die nur auf synthetischen Daten trainiert wurden und Modelle, welche anschliessend auf dem *HCS*-Datensatz finegetuned wurden.

Modell	**Ply Accuracy**	**Character Accuracy**
Ohne Finetuning		
SYN-200K Baseline	48.60 %	75.30 %
SYN-FC	35.65 %	58.64 %
SYN-FC (5 Enc.-Layer + 5 Dec.-Layer)	43.43 %	63.81 %
SYN-WC	54.00 %	74.99 %
Mit Finetuning		
SYN-WC (3×3 Filter)	73.80 %	88.44 %
SYN-WC (7×7 Filter)	84.35 %	94.02 %
SYN-WC (CTC, 7×7, 6 Enc.-Layer)	83.80 %	94.07 %

Tabelle 15: Gesamtübersicht aller Ergebnisse. Evaluation jeweils auf dem *CR-v6-EN*-Datensatz.

Der grösste Einzelsprung ist auf das Finetuning mit realen Daten zurückzuführen (19.80 % Ply Accuracy Zuwachs gegenüber dem rein synthetisch trainierten *SYN-WC*-Modell). Die Erhöhung der Filtergrösse auf 7×7 brachte einen weiteren substanziellen Gewinn von 10.55 %. CTC- und Transformer-Modell liegen mit 7×7-Filtern nahezu gleichauf, was zeigt, dass der Flaschenhals nicht in der Decoder-Architektur liegt.

Um die praktische Einsetzbarkeit des Modells sicherzustellen, wurde die Zeit gemessen, die benötigt wird, um ein Scoresheet zu erkennen. Getestet wurde dies mit dem Scoresheet *en_001_32* aus dem *CR-v6-EN*-Datensatz, welches 63 Züge enthält. Dazu wurde das Scoresheet dreimal in der ChessReader-Software verarbeitet. Dabei zeigte sich, dass der ChessReader auch mit den neuen Modellen, ein Scoresheet in weniger als einer Minute verarbeiten kann. Damit erfüllen die Modelle die Anforderungen aus dem Kapitel 4.2 Zielsetzung.

9 Fehleranalyse

Die folgende Fehleranalyse bezieht sich auf die jeweils besten Modelle, der beiden Modellarchitekturen Transformer und CTC. Alle Analysen wurden wie auch in den vorherigen Abschnitten auf den Ergebnissen des Greedy Decoding durchgeführt.

9.1 Greedy Decoding vs. Beam Search

Bei der Entwicklung der OCR-Modelle wurden sowohl Greedy Decoding als auch Beam Search implementiert und bei jedem Test durchgeführt. Wie in Abschnitt 5.4.5 beschrieben, erlaubt Beam Search die Einbindung domänenspezifischer Constraints. In dieser Arbeit wurde dies genutzt, um während der Decodierung syntaktisch ungültige SAN-Züge anhand der SAN-Grammatik auszuschliessen.

Trotz dieser Möglichkeit zeigte sich in der Praxis keine signifikante Verbesserung gegenüber dem einfacheren Greedy Decoding. Da Greedy Decoding zudem deutlich schneller ist, was im Kontext des ChessReader-Projekts relevant ist, wurde es als Standard für alle Experimente gewählt. Der Grund für den geringen Unterschied liegt in der hohen Modellqualität. Das Modell scheint die SAN-Grammatik bereits implizit erlernt zu haben und produziert auch ohne explizite Grammatikprüfung praktisch keine invaliden Züge.

Dennoch gibt es eine kleine Anzahl von Fällen, in denen Beam Search durch die Grammatikprüfung das Ergebnis korrigiert. Diese betreffen typische OCR-Fehler, bei denen Zahlen und Buchstaben verwechselt werden. Tabelle 16 zeigt Beispiele, die Beam Search korrekt erkannte, während Greedy Decoding scheiterte.

Korrekt	**Greedy Decoding**	**Erklärung**
Qb8	Q68	Nach dem *Q* müsste ein kleiner Buchstabe und keine Zahl folgen.
Rb1	R6+	Nach dem *R* fehlt ein kleiner Buchstabe.
d4	dxe	Nach dem *e* fehlt eine Zahl.
f5	K5	Nach dem *K* müsste ein kleiner Buchstabe folgen oder vor der *5* ein kleiner Buchstabe stehen.
Bxb7	Bx6	Nach dem *x* fehlt ein kleiner Buchstabe.
a5	ax5	Das *x* ist überflüssig oder vor der *5* fehlt ein kleiner Buchstabe.
Rd1	ed1	Das *e* ist überflüssig oder vor dem *d* müsste ein grosser Buchstabe stehen.

Tabelle 16: Beispiele typischer OCR-Fehler (Greedy Decoding vs. Beam Search)

In allen diesen Fällen erkannte Greedy Decoding einen ungültigen oder unvollständigen Zug, während Beam Search durch die Grammatikprüfung zum korrekten Zug kam.

9.2 Edit-Distance-Verteilung

Die Edit-Distance war einer der ersten Metriken, die analysiert wurden. Die Analyse zeigt, von insgesamt 3111 Bildern wurden 2624 Züge korrekt erkannt (Edit-Distance von null). Bei den 487 falsch erkannten Zügen liegt der Grossteil (414 Züge ca. 85%) bei einer Levenshtein-Distanz von eins. Dies bedeutet, dass die überwiegende Mehrheit der Fehler nur einen einzelnen Zeichenunterschied vom korrekten Zug entfernt ist. Wie in der Abbildung 22 zu sehen ist, gibt es praktisch keine Fehler mit einer Distanz von mehr als zwei. Das bedeutet also, dass nicht ganze Züge falsch erkannt, sondern einzelne Zeichen vertauscht, hinzugefügt oder weggelassen werden.

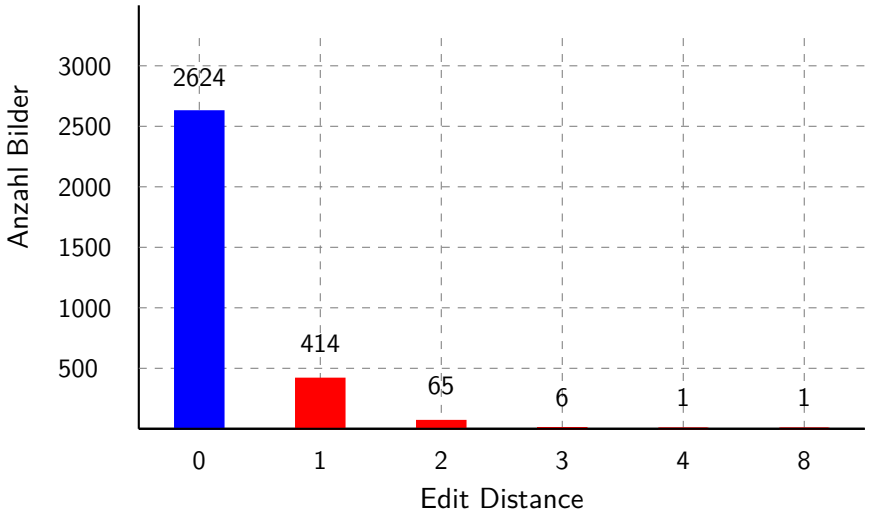


Abbildung 22: Verteilung der Edit-Distance von Zug-Vorhersagen

9.2.1 Beam Search Ranking-Analyse

Diese Erkenntnisse führte zu einer weiteren Untersuchung, ob bei den falsch erkannten Zügen die korrekten Ergebnisse in den Top-10-Positionen der Beam Search Vorhersagen liegen. Bei der Analyse zeigt sich tatsächlich, dass der korrekte Zug zu einem hohen Prozentsatz in den Top-Kandidaten liegt. Konkret sind es 487 welche von Greedy Decoding falsch erkannt wurden. Davon konnten 354 Züge (72.7 %) in den Top-10 Ergebnissen von Beam Search gefunden werden. Die Tabelle 17 zeigt die Verteilung auf welcher Position sich der korrekte Zug in den Top-10 Kandidaten befindet.

Position	Anzahl	Prozentsatz
1	11	2.3 %
2	227	46.6 %
3	62	12.7 %
4	27	5.5 %
5	18	3.7 %
6	7	1.4 %
7	1	0.2 %
8	0	0 %
9	1	0.2 %
10	0	0 %
>10	133	27.3 %

Tabelle 17: Verteilung der Beam Search Positionen für falsch erkannte Züge

Die Untersuchung zeigt, dass bereits die Top-3 Kandidaten von Beam Search in über 60 % der Fälle den korrekten Zug enthalten. Betrachtet man die Top-5 Kandidaten sind es sogar über 70 %. Ausserhalb der Top-5 lassen sich nur vereinzelt korrekte Züge finden, wie die Abbildung 23 zeigt. Es zeigt also, dass Beam Search tatsächlich seine Vorteile hat, allerdings nicht mit der aktuellen Version der ChessReader-Applikation, da diese nur einen einzigen Zug als Input akzeptiert. Wie initial erwähnt kann Beam Search dadurch aktuell keine wirkliche Abhilfe schaffen, da die Top-1-Vorhersage bereits zu stark bevorzugt wird. Dadurch kann das Endergebnis fast nicht mehr verbessert werden.

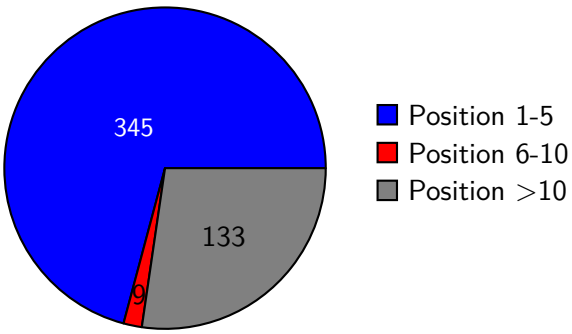


Abbildung 23: Kategorisierung der Beam Search Positionen für falsch erkannte Züge

9.3 Character Confusion Matrix

Die Analyse der Edit-Distance deutet darauf hin, dass vor allem einzelne Zeichenfehler die grössten Fehler ausmachen. Deshalb wurde die Confusion Matrix analysiert, um zu prüfen, ob es bestimmte Zeichen gibt, die besonders häufig verwechselt oder falsch erkannt werden.

Dafür wurde sowohl die Matrix vor als auch nach dem Fine-Tuning analysiert, wobei eine deutliche Reduktion der Fehlerstreuung zwischen Abbildung 24 und Abbildung 25 sichtbar ist.

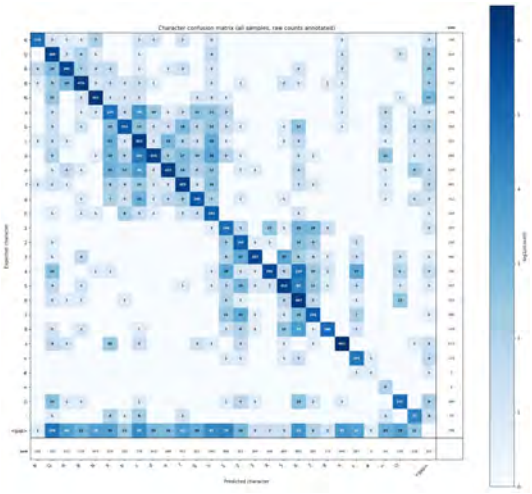


Abbildung 24: Confusion Matrix vor dem Fine-Tuning

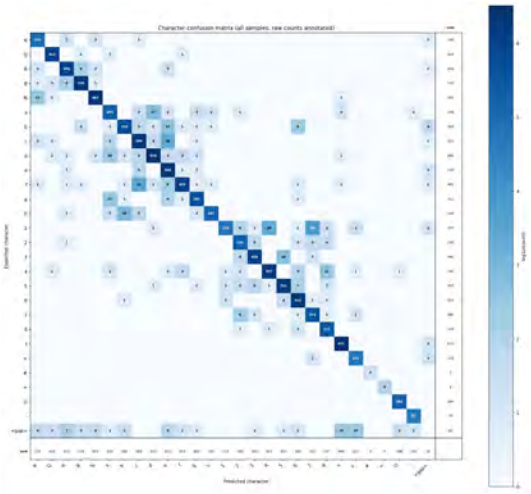


Abbildung 25: Confusion Matrix nach dem Fine-Tuning

Die Confusion Matrix vor dem Fine-Tuning zeigt eine breite Streuung der Fehler über viele verschiedene Zeichen. Zum Beispiel die Zahl *8* wird mit zehn falschen Zeichen verwechselt und der Buchstabe *O* sogar mit elf falschen Zeichen.

Nach dem Fine-Tuning zeigt sich eine deutlichere Konzentration der Fehler auf wenige Zeichen. Wieder beim Beispiel der Zahl *8*, wird diese nur noch mit drei falschen Zeichen verwechselt und der Buchstabe *O* wird sogar gar nicht mehr verwechselt. Zudem fällt auf, dass weniger oft Zeichen fälschlich hinzugefügt oder weggelassen werden. Dies ist an den <*gap*>-Zeichen zu erkennen, welche deutlich weniger vorkommen als vor dem Fine-Tuning.

Diese Konzentration ist ein positives Zeichen dafür, dass das Modell durch das Fine-Tuning besser und gezielter lernt. Denn es treten nicht nur weniger Fehler auf, sondern sie entsprechen einem bestimmten Muster. Das bedeutet, dass das Modell mehr Muster erkennen kann, auch wenn es noch nicht alle Muster korrekt erkennt.

Die Tabelle 18 zeigt die zehn häufigsten Zeichenverwechslungen, die nach dem Fine-Tuning noch auftreten und welchen Anteil sie ausmachen vom Gesamtaufkommen der beiden Zeichen.

Rang	Zeichen 1	Zeichen 2	Anzahl	Anteil
1	1	7	44	8.87 %
2	c	e	33	2.94 %
3	c	f	31	3.00 %
4	1	4	29	3.80 %
5	a	d	27	3.06 %
6	b	h	19	3.31 %
7	3	5	19	1.90 %
8	B	R	16	1.69 %
9	a	g	15	2.75 %
10	2	7	13	2.50 %

Tabelle 18: Top 10 der häufigsten Zeichenverwechslungen nach dem Fine-Tuning

Diese Fehler sind typische Probleme im Zusammenhang der optischen Zeichenerkennung. Sie lassen sich auf die optische Ähnlichkeit der Symbole und den verschiedenen Schreibweisen zurückführen.

Verwechslung von *1* und *7*. Die Verwechslung zwischen der Ziffer *1* und *7* ist eine der häufigsten Fehlerquellen. Der Grund dafür liegt vor allem in den verschiedenen Schreibweise der Zahl *7*. Gewisse Personen schreiben die Zahl ohne horizontalen Strich. Eine solche gleicht stark einer schief geschriebener *1* wie in Abbildung 26.

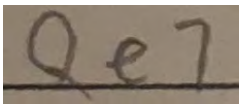


Abbildung 26: Visuelle Ähnlichkeit zwischen 1 und 7

Verwechslung von *c* und *e*. Das Beispiel in Abbildung 27 zeigt ein schlecht geschlossenes *e*, kann durchaus als *c* interpretiert werden.

Abbildung 27: Visuelle Ähnlichkeit zwischen c und e

Verwechslung von *a* mit *d* und *g*. Ein *d* mit einem sehr kleinen vertikalen Strich wie in Abbildung 28 oder ein *g* bei dem der Bogen abgeschnitten ist wie in Abbildung 29, kann leicht als *a* interpretiert werden.

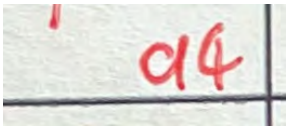


Abbildung 28: Visuelle Ähnlichkeit zwischen a und d

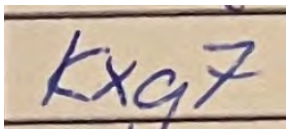


Abbildung 29: Visuelle Ähnlichkeit zwischen a und g

Verwechslung von *b* und *h* & *B* und *R*. Bei einem nicht sauber geschlossenen Bogen vom Buchstaben *b* bzw. *B* gleichen diese stark einem *h* (Abbildung 30) bzw. *R* (Abbildung 31).

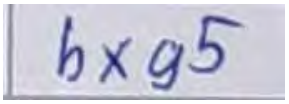


Abbildung 30: Visuelle Ähnlichkeit zwischen b und h

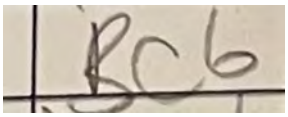


Abbildung 31: Visuelle Ähnlichkeit zwischen B und R

Verwechslung von *b* und *6*. Bei einem rund geschriebenen *b* (wie in Abbildung 32) liest sich dieses mehr wie eine *6*.

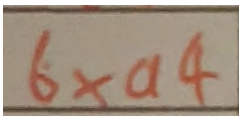


Abbildung 32: Visuelle Ähnlichkeit zwischen b und 6

9.4 Probleme der Validierungsdaten

Abschliessend wurden nochmals alle Validierungsdaten, welche nicht erkannt wurden, untersucht. Dabei zeigte sich, dass es gewisse Bilder gab, die nicht sauber zugeschnitten oder unlesbar bzw. unklar waren. Zudem gab es auch Bilder, bei denen das Label mindestens fragwürdig war.

Schlecht ausgeschnittene Bilder. Die Boxen gewisser Bilder beinhalten nicht den ganzen Zug oder beinhalten zusätzlich Text der nicht zum Zug gehört. Die Abbildungen 33 bis 36 zeigen Beispiele dafür und werden in der Tabelle 19 erklärt.

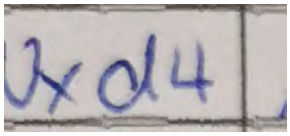


Abbildung 33: Nxd4

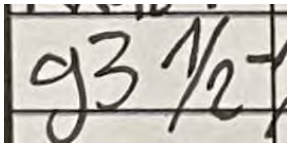


Abbildung 34: g3

Abbildung 35: Kh5

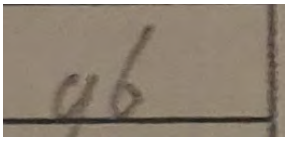


Abbildung 36: g6

Label	Problem
Nxd4	*N* ist nur teilweise sichtbar
g3	Es ist zusätzlicher Text sichtbar (½)
Kh5	*K* ist nur teilweise sichtbar
g6	*g* ist nur teilweise sichtbar

Tabelle 19: Beispiele schlecht ausgeschnittener Bilder

Mehrdeutige Zeichen. Teilweise ist die Schrift so schlecht, dass gewisse Zeichen nicht eindeutig zugeordnet werden können. Die Abbildungen 37 bis 40 zeigen Beispiele dafür und werden in der Tabelle 20 erklärt.

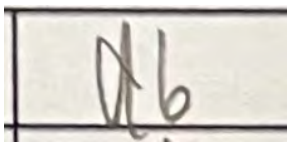


Abbildung 37: d6

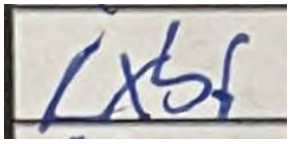


Abbildung 38: cxb6

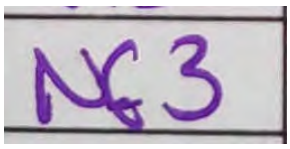


Abbildung 39: Nf3

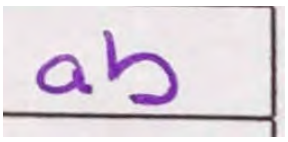


Abbildung 40: a6

Label	Problem
d6	Das *d* könnte auch ein *a* oder *Ol* sein
cxb6	Das *c* könnte auch ein *L* oder *I* sein
Nf3	Das *f* könnte auch ein *c* sein
a6	Die *6* könnte auch eine *5* oder ein *b* sein

Tabelle 20: Beispiele mehrdeutiger Zeichen

Unlesbare Zeichen. Bei einem Teil der Bilder sind einzelne Zeichen schlecht bis gar nicht lesbar, dies meist aufgrund von Nachkorrekturen. Die Abbildungen 41 bis 44 zeigen Beispiele dafür und werden in der Tabelle 21 erklärt.

Abbildung 41: Nbd2

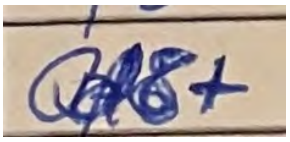


Abbildung 42: Qd8+

Abbildung 43: c5

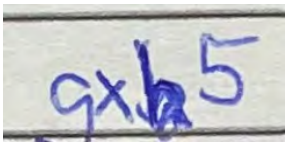


Abbildung 44: gxh5

Label	Problem
Nbd2	Das *d* ist nicht als solches erkennbar
Qd8+	Das *d8* ist stark überschrieben
c5	Das *c* ist stark überschrieben und es ist eine weitere *5* erkennbar
gxh5	Das *h* ist stark überschrieben

Tabelle 21: Beispiele unlesbarer Zeichen

Potenziell falsche Labels. Bei manchen Bildern ist das Label zumindest fragwürdig. Es ist wahrscheinlich der Zug der ausgeführt wurde, aber er ist nicht auf dem Scoresheet als solcher erkennbar. Die Abbildungen 45 bis 48 zeigen Beispiele dafür und werden in der Tabelle 22 erklärt.

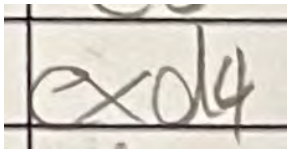


Abbildung 45: cxd4

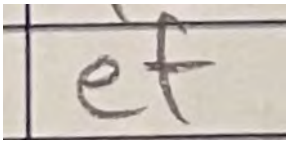


Abbildung 46: e4

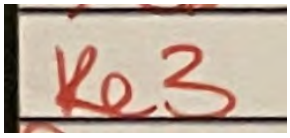


Abbildung 47: Re3

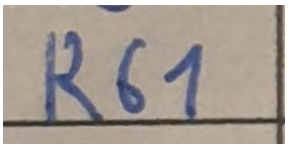


Abbildung 48: Rb1

Label	Problem
cxd4	Das *c* entspricht einem *e*
e4	Die *4* entspricht einem *f*
Re3	Das *R* entspricht einem *K*
Rb1	Das *b* entspricht einer *6*

Tabelle 22: Beispiele potenziell falscher Labels

10 Diskussion

10.1 Wichtigste experimentelle Erkenntnisse

Modellgrösse. Das Erhöhen der Parameteranzahl führt ab einem bestimmten Punkt nicht mehr zu einer Verbesserung der Modellleistung. Dies zeigt auch das gescheiterte Experiment mit dem Erhöhen der Anzahl Schichten auf jeweils fünf im Abschnitt 8.1.6. Daraus lässt sich schliessen, dass für die Aufgabe der SAN-Erkennung nicht primär grössere Modelle entscheidend sind, sondern andere Faktoren eine grössere Rolle spielen.

Transformer vs. CTC. Sowohl der Transformer als auch CTC erzielten ähnliche Ergebnisse. Dies deutet darauf hin, dass beide Architekturen grundsätzlich Potenzial für die Aufgabe des ChessReaders besitzen. Da der Transformer in den durchgeführten Tests jedoch leicht bessere Ergebnisse erzielte, wurde diese Architektur als Grundlage für das endgültige Modell dieser Arbeit ausgewählt.

Einfluss der Datensätze. Ein wichtiger Einflussfaktor der Modellresultate ist die Verfügbarkeit realer Trainingsdaten. Die Ergebnisse zeigen, dass für die Aufgabe des ChessReaders mehr reale Daten benötigt werden. Auch wenn es keine Garantie auf eine Verbesserung gibt, deutet vieles darauf hin. Diese Erkenntnis wird sowohl durch die Experimente zum Pretraining auf realen Daten (siehe Abschnitt 8.3.2), als auch durch das Finetuning mit unterschiedlichen Datenmengen (siehe Abschnitt 8.3.1) gestützt.

10.2 Limitationen

Eine zentrale Limitation des ChessReaders liegt in der grossen Vielfalt an Handschriften. Diese unterscheiden sich stark zwischen einzelnen Personen, wodurch gewisse Zeichen auch bei einem gut trainierten Modell schwierig eindeutig zu erkennen sind. Besonders problematisch sind Fälle, in denen ein Zeichen von einer Person so geschrieben wird, dass es der Schreibweise eines anderen Zeichens sehr ähnlich ist. Besonders wenn nur sehr wenige Personen das Zeichen auf diese Weise schreiben.

Solche Fehler lassen sich nicht vollständig vermeiden, da sie nicht ausschliesslich auf Schwächen des Modells zurückzuführen sind, sondern auch auf Mehrdeutigkeiten in den Eingabedaten.

10.3 Ausblick auf zukünftige Arbeiten

In zukünftigen Arbeiten könnten verschiedene Ansätze verfolgt werden, um die Qualität des ChessReaders weiter zu verbessern.

10.3.1 Überarbeitung des Fine-Tuning-Datensatzes

Im Verlauf dieser Arbeit wurde der *HCS*-Datensatz mehrfach überarbeitet, um die Qualität der Daten zu verbessern, wie bereits im Vorgehen Abschnitt 6.3.6 beschrieben wurde. Dabei zeigte sich, dass jede Überarbeitung zu einer Verbesserung der Modellresultate führte, was sich auch in den Ergebnissen widerspiegelte. Daher könnte es sich lohnen, den Datensatz in zukünftigen Arbeiten weiter zu bereinigen und gezielt auf verbleibende Fehler zu untersuchen. Obwohl alle Daten bereits von Hand überprüft wurden, ist nicht auszuschliessen, dass weiterhin einzelne Bilder enthalten sind, die schlecht zugeschnitten, unleserlich, uneindeutig oder falsch gelabelt sind.

10.3.2 Fokus auf problematische Zeichen

Ein weiterer Ansatzpunkt ergibt sich aus der Fehleranalyse. Die Auswertung der Edit-Distance zeigte, dass hauptsächlich einzelne Zeichen falsch erkannt wurden. Auch die Analyse der Confusion-Matrix deutete darauf hin, dass bestimmte Zeichen häufiger zu Fehlklassifikationen führen als andere. Zukünftige Arbeiten könnten daher gezielt auf diese problematischen Zeichen fokussieren. Eine Möglichkeit wäre, zusätzliche Scoresheets von Hand auszufüllen, mit Schachzügen welche besonders vielen dieser Zeichen enthalten. Dabei sollte darauf geachtet werden, dass die Daten möglichst viele unterschiedliche Handschriften, Schreibstile und Schreibmaterialien abdecken.

10.3.3 Überarbeitung des Testdatensatzes

Auch der Testdatensatz (*CR-v6-EN*) weist weiteres Überarbeitungspotenzial auf. In dieser Arbeit wurden darin nur wenige einzelne Bilder korrigiert, da der Datensatz zu Beginn als Vergleichsgrundlage festgelegt wurde. Deshalb wurden lediglich Bilder angepasst, die eindeutig fehlerhaft waren. Weitere potenzielle Schwächen wurden zwar analysiert, jedoch nicht korrigiert, um die Vergleichbarkeit der Experimente nicht zu beeinträchtigen. In zukünftigen Arbeiten könnte es sich daher lohnen, den Testdatensatz nochmals umfassend zu überprüfen und gegebenenfalls durch neue und qualitativ bessere Bilder zu ersetzen. Dadurch könnte die Aussagekraft der Evaluation weiter verbessert werden.

Ein konkreter Fall betrifft das *Scoresheet 39*, das in dieser Arbeit vollständig entfernt wurde, da es Probleme beim automatischen Zuschneiden gab. In einer zukünftigen Überarbeitung könnten die entsprechenden Koordinaten der Boxen manuell definiert werden oder das Bild neu aufgenommen werden.

10.3.4 Implementierung von Beam Search

Ausserdem wurde in der Fehleranalyse der Unterschied zwischen Greedy Decoding und Beam Search betrachtet. Dabei zeigte sich, dass Beam Search auf den ersten Blick keine besseren Ergebnisse als Greedy Decoding erzielte. Betrachtet man jedoch nicht nur das beste Ergebnis, sondern die fünf besten Ergebnisse der Beam Search Ausgabe, befindet sich in vielen Fällen das korrekte Ergebnis darunter, wie die Abbildung 23 zeigt. Für den aktuellen ChessReader ist diese Erkenntnis zwar nicht relevant, sie könnte jedoch bei einer Weiterentwicklung des Systems von Bedeutung sein.

Beispielsweise könnte der ChessReader künftig jeweils die fünf wahrscheinlichsten Ergebnisse ausgeben. Das wahrscheinlichste Ergebnis könnte direkt angezeigt werden, während die weiteren Vorschläge über ein Dropdown-Menü auswählbar wären. Dadurch wäre in diesen Fällen weiterhin eine Nachkorrektur nötig, jedoch müsste das Ergebnis nicht manuell eingegeben werden. Stattdessen könnte in vielen Fällen einfach der passende Vorschlag ausgewählt werden, was den Korrekturprozess deutlich vereinfachen und beschleunigen könnte.

11 Fazit

Ziel dieser Arbeit war die Entwicklung und Evaluation eines robusten, intern betreibbaren OCR-Modells zur Erkennung handschriftlicher SAN-Züge für das ChessReader-Projekt. Damit sollte die Abhängigkeit von externen Diensten reduziert werden, ohne dass die Genauigkeit der ChessReader-Software dadurch sinkt. Zur Untersuchung dieser Fragestellung wurden zwei Modellansätze betrachtet: Ein Transformer-Encoder-Decoder mit CNN-Backbone sowie eine Variante mit reinem Encoder und CTC-Decoding.

Die Experimente zeigen, dass synthetische Daten allein zwar eine brauchbare Ausgangsbasis liefern, für eine hohe Modellqualität jedoch nicht ausreichen. Der grösste Leistungssprung entstand durch das Einbeziehen realer handschriftlicher Daten im Finetuning. Ebenso wurde deutlich, dass die Qualität und Repräsentativität der Trainingsdaten wichtiger sind als eine möglichst grosse Menge synthetischer Bilder. Auch die Vergrösserung der CNN-Filter brachte einen klaren Gewinn, was darauf hindeutet, dass die lokale Feature-Extraktion für diese Aufgabe eine zentrale Rolle spielt.

Als insgesamt beste Variante erwies sich das Transformer-Encoder-Decoder-Modell mit CNN-Backbone und 7×7-Filtern. Es erreichte eine Ply Accuracy von 84.35 % und eine Character Accuracy von 94.02 %. Das CTC-Modell erzielte mit 83.80 % Ply Accuracy und 94.07 % Character Accuracy eine sehr ähnliche Leistung. Eine eindeutige architekturelle Überlegenheit lässt sich daraus nicht ableiten. Für den Einsatz im ChessReader-Projekt ist das Encoder-Decoder-Modell aktuell die geeignetere Wahl, da die leicht höhere Ply Accuracy für den praktischen Nutzen entscheidender ist.

Trotz der guten Resultate bleiben Grenzen sichtbar. Vor allem schwer lesbare oder uneindeutig geschriebene Schachzüge, sowie die Vielfalt individueller Handschriften stellen weiterhin eine Herausforderung dar. Zukünftige Arbeiten sollten deshalb vor allem die Datenbasis weiter verbessern und verbleibende Fehlerbilder gezielt adressieren. Insgesamt zeigt die Arbeit aber, dass ein robustes, intern nutzbares OCR-Modell für den ChessReader realisierbar ist und auf einer soliden Grundlage weiterentwickelt werden kann.

Glossar

BiLSTM Bidirectional Long Short-Term Memory beschreibt eine LSTM-Variante, welche eine Sequenz vorwärts und rückwärts verarbeitet, um Kontext aus beiden Richtungen zu nutzen.

Box Rechteckiger Bereich auf einem Scoresheet, in dem ein einzelner Schachzug notiert ist. Im Kontext des ChessReaders bezeichnet eine Box meist den erkannten oder ausgeschnittenen Bildbereich.

Dynamic Programming Eine Methode, bei der ein Problem in kleinere Teilprobleme zerlegt wird, deren Lösungen gespeichert und wiederverwendet werden.

en passant Spezielle Schlagregel im Schach, bei der ein Bauer einen gegnerischen Bauern schlagen kann, der unmittelbar zuvor zwei Felder vorgezogen ist.

Feed-Forward Teil eines neuronalen Netzes, bei dem Informationen nur in eine Richtung durch mehrere Schichten weitergegeben werden. In Transformer-Schichten bezeichnet es meist ein kleines voll verbundenes Netzwerk, das nach der Attention auf jede Token-Position angewendet wird.

LSTM Long Short-Term Memory bezeichnet eine rekurrente neuronale Netzarchitektur, die Sequenzen verarbeitet und Informationen über mehrere Zeitschritte hinweg speichern kann.

Ply Bezeichnung für einen einzelnen Schachzug einer Person.

Simple Chess Engine Eine einfache Python-Schachengine, die Schachzüge bewertet und auswählt.

Literatur

[1] Fédération Internationale des Échecs. „Laws of Chess," besucht am 30. Mai 2026. Adresse: https://www.fide.com/FIDE/handbook/LawsOfChess.pdf

[2] „ChessReader," besucht am 5. Juni 2026. Adresse: https://chessreader.org/about

[3] N. Majid und O. Eicher, „Digitization of Handwritten Chess Scoresheets with a BiLSTM Network," *Journal of Imaging*, Jg. 8, Nr. 2, 2022, issn: 2313-433X. doi: 10.3390/jimaging8020031 Adresse: https://www.mdpi.com/2313-433X/8/2/31

[4] O. Eicher, D. Farmer, Y. Li und N. Majid. „Handwritten Chess Scoresheet Dataset," besucht am 29. Mai 2026. Adresse: http://tc11.cvc.uab.es/datasets/HCS_1

[5] A. Graves, S. Fernández, F. Gomez und J. Schmidhuber, „Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks," in *Proceedings of the 23rd International Conference on Machine Learning*, Ser. ICML '06, 2006, S. 369–376. doi: 10.1145/1143844.1143891

[6] T. Onliner. „Schach: das älteste Brettspiel der Welt," besucht am 22. Mai 2026. Adresse: https://theonliner.ch/22-9-2022/schach-das-%C3%A4lteste-brettspiel-der-welt

[7] „Schach," besucht am 2. Juni 2026. Adresse: https://de.wikipedia.org/wiki/Schach

[8] R. Stone. „Bobby Fischer: Showdown in Reykjavik," besucht am 22. Mai 2026. Adresse: https://www.rollingstone.com/culture/culture-news/bobby-fischer-showdown-in-reykjavik-177551/

[9] „Recognised International Federations," besucht am 30. Mai 2026. Adresse: https://www.olympics.com/ioc/recognised-international-federations

[10] „Partieformular," besucht am 2. Juni 2026. Adresse: https://de.wikipedia.org/wiki/Partieformular

[11] Wikimedia Commons. „Bobby Fischer Scoresheet," besucht am 30. Mai 2026. Adresse: https://commons.wikimedia.org/w/index.php?curid=523458

[12] „Schachbrett," besucht am 31. Mai 2026. Adresse: https://de.wikipedia.org/wiki/Schachbrett

[13] Matheretter. „Koordinaten beim Schachbrett," besucht am 2. Juni 2026. Adresse: https://www.matheretter.de/wiki/schachbrett

[14] „Schachfigur," besucht am 31. Mai 2026. Adresse: https://de.wikipedia.org/wiki/Schachfigur

[15] K. Hamad und M. Kaya, „A Detailed Analysis of Optical Character Recognition Technology," *International Journal of Applied Mathematics Electronics and Computers*, Jg. 4, 2016. Adresse: https://www.researchgate.net/publication/311851325_A_Detailed_Analysis_of_Optical_Character_Recognition_Technology

[16] IBM. „Convolutional Neural Networks,“ besucht am 1. Juni 2026. Adresse: https://www.ibm.com/think/topics/convolutional-neural-networks

[17] A. Amidi und S. Amidi, *Convolutional Neural Networks Cheatsheet*, CS 230 Deep Learning, Stanford University, 2019. besucht am 30. Mai 2026. Adresse: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks

[18] A. Vaswani u. a., „Attention Is All You Need,“ in *Advances in Neural Information Processing Systems*, Bd. 30, 2017. Adresse: https://arxiv.org/abs/1706.03762

[19] A. Dosovitskiy u. a., „An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale,“ in *International Conference on Learning Representations*, 2021. Adresse: https://arxiv.org/abs/2010.11929

[20] M. Li u. a., „TrOCR: Transformer-Based Optical Character Recognition with Pre-trained Models,“ in *Proceedings of the AAAI Conference on Artificial Intelligence*, Bd. 37, 2023, S. 13 094–13 102. doi: 10.1609/aaai.v37i11.26538 Adresse: https://ojs.aaai.org/index.php/AAAI/article/view/26538

[21] A. Zhang, Z. C. Lipton, M. Li und A. J. Smola. „Beam Search,“ besucht am 30. Mai 2026. Adresse: https://d2l.ai/chapter_recurrent-modern/beam-search.html

[22] A. Hannun, „Sequence Modeling with CTC,“ *Distill*, 2017, https://distill.pub/2017/ctc. doi: 10.23915/distill.00008

[23] N. Moser und R. Meier, „Aufbau einer Evaluierungspipeline für die Schachspiel-Digitalisierungssoftware ChessReader,“ unveröffentlicht, Bachelorarbeit, Zürcher Hochschule für Angewandte Wissenschaften, 2024.

[24] A. George, A. Troussard und P.-H. Levieil, „Machine Learning for Chess Movement Recognition,“ CS-433 Project, EPFL, Switzerland. Unveröffentlicht.

[25] A. Büchi, D. Eymold und M. Gallati, „Optimizing OCR and UI for ChessReader: A Chess Scoresheet Digitization System,“ Unveröffentlicht, Bachelorarbeit, Zürcher Hochschule für Angewandte Wissenschaften, 2025.

[26] ZHAW School of Engineering. „ZHAW SoE High-Performance Computing (HPC) Cluster,“ besucht am 2. Juni 2026. Adresse: https://hpc.engineering.zhaw.ch/

Abbildungsverzeichnis

1 Bobby Fischer gegen Tigran Petrosjan 1971 in Buenos Aires [8] 12

2 Beispiel eines Scoresheets von Bobby Fischer [11] 13

3 Koordinatensystem des Schachbretts [13] 14

4 SAN Figuren (Deutsch & Englisch) [14] 15

5 CNN-Aufbau anhand eines Beispiels [17] 16

6 Convolutional Layer mit Filter-Verschiebung über das Eingabebild [17] 16

7 Pooling Layer Arten: Max-Pooling und Average-Pooling 17

8 Fully Connected Layer mit vollständigen Verbindungen zwischen Neuronen [17] . 18

9 Transformer-Encoder-Decoder-Architektur nach *Attention Is All You Need* [18]. . 19

10 Gesamtarchitektur des Transformer-Encoder-Decoder-Modells 33

11 Loss-Kurven des Basismodells (Training auf *SYN-200K*). 34

12 Loss-Kurven des Modells trainiert auf *SYN-FC*. 36

13 Character Confusion Matrix des auf *SYN-FC* trainierten Modells 37

14 Loss-Kurven des Modells (3 Encoder, 2 Decoder) trainiert auf *SYN-FC* 38

15 Loss-Kurven des Pretrainings auf *SYN-WC* 39

16 Character Confusion Matrix nach dem Pretraining auf *SYN-WC* 40

17 Loss-Kurven des Finetunings auf *HCS* 41

18 Character Confusion Matrix nach dem Finetuning auf *HCS* 42

19 Character Confusion Matrix nach dem Finetuning mit 7×7 Filtergrösse 44

20 Ply und Character Accuracy beim Finetuning im Vergleich (Transformer-Modell) 48

21 Ply und Character Accuracy beim Finetuning im Vergleich (CTC-Modell) 49

22 Verteilung der Edit-Distance von Zug-Vorhersagen 54

23 Kategorisierung der Beam Search Positionen für falsch erkannte Züge 55

24 Confusion Matrix vor dem Fine-Tuning 56

25 Confusion Matrix nach dem Fine-Tuning 56

26 Visuelle Ähnlichkeit zwischen 1 und 7 57

27 Visuelle Ähnlichkeit zwischen c und e 57

28 Visuelle Ähnlichkeit zwischen a und d 58

29 Visuelle Ähnlichkeit zwischen a und g 58
30 Visuelle Ähnlichkeit zwischen b und h 58
31 Visuelle Ähnlichkeit zwischen B und R 58
32 Visuelle Ähnlichkeit zwischen b und 6 58
33 Nxd4 59
34 g3 59
35 Kh5 59
36 g6 59
37 d6 59
38 cxb6 59
39 Nf3 59
40 a6 59
41 Nbd2 60
42 Qd8+ 60
43 c5 60
44 gxh5 60
45 cxd4 60
46 e4 60
47 Re3 60
48 Rb1 60

Tabellenverzeichnis

1 SAN Regeln [1] 14

2 CNN-Backbone Architektur 30

3 Transformer-Encoder Hyperparameter (Basismodell) 31

4 Transformer-Decoder Hyperparameter (Basismodell) 31

5 Training-Hyperparameter (Basismodell) 32

6 Parameteranzahl des Basismodells 32

7 Beispiele von problematischen Schriftarten und getauschten Zeichen 35

8 Parameteranzahl des Modells (3 Encoder, 2 Decoder) 38

9 Ergebnisse vor und nach Erhöhung der Filtergrösse auf 7×7. 43

10 Architektur-Änderungen vom Encoder-Decoder zum Encoder-Only Modell 46

11 Parameter-Vergleich Transformer vs. CTC 47

12 Vergleich Accuracy Transformer vs. CTC 47

13 Pretraining Resultate ohne Finetuning (Transformer-Modell) 50

14 Pretraining Resultate ohne Finetuning (CTC-Modell) 51

15 Gesamtübersicht aller Ergebnisse. Evaluation jeweils auf dem *CR-v6-EN*-Datensatz. 52

16 Beispiele typischer OCR-Fehler (Greedy Decoding vs. Beam Search) 53

17 Verteilung der Beam Search Positionen für falsch erkannte Züge 55

18 Top 10 der häufigsten Zeichenverwechslungen nach dem Fine-Tuning 57

19 Beispiele schlecht ausgeschnittener Bilder 59

20 Beispiele mehrdeutiger Zeichen 59

21 Beispiele unlesbarer Zeichen 60

22 Beispiele potenziell falscher Labels 60

24 Eingesetzte KI-Werkzeuge während der Erstellung der Arbeit 71

25 Übersicht über die Ablage der Endergebnisse 72

A KI-Nutzungsdeklaration

Während der Erstellung dieser Bachelorarbeit wurden KI-gestützte Werkzeuge für die in Tabelle 24 aufgeführten Zwecke eingesetzt. Alle KI-generierten Inhalte wurden sorgfältig geprüft, validiert und bei Bedarf angepasst. Die zentralen Ideen, Analysen und Schlussfolgerungen stammen vollständig von den Autoren.

KI-Werkzeug	**Verwendungszweck**
GitHub Copilot	Code-Vervollständigung und allgemeine Code-Generierung. Zusätzliche Unterstützung beim Verfassen, Korrigieren und Formulieren von Textabschnitten der Arbeit
ChatGPT	Recherche, Problemlösung und Hilfestellung bei fachlichen Fragestellungen
DeepL	Textverbesserung und Übersetzung (insbesondere beim Abstract)
Google Search AI Summary	KI-gestützte Zusammenfassungen, die bei der regulären Google-Websuche automatisch angezeigt werden

Tabelle 24: Eingesetzte KI-Werkzeuge während der Erstellung der Arbeit

B Ablage der Endergebnisse

Die Ablage der finalen Resultate der Arbeit werden zur besseren Nachvollziehbarkeit und Wiederverwendbarkeit in der Tabelle 25 aufgeführt. Alle Dateien befinden sich auf dem *ZHAW SoE Compute Cluster* [26] im Projektordner `/cluster/projects/2026_vode_BA_ChessCNN`.

Artefakt	**Ablageort (im Projektordner)**
Bestes OCR-Modell	`./abgabe/TR_12_Chess_OCR_hcs_v3`
Transformer-Modell Quellcode	`./repos/ChessReader_CNN_TR`
CTC-Modell Quellcode	`./repos/ChessReader_CNN_CTC`
Synthetischer Generator Quellcode	`./repos/synthetic-image-generator`

Tabelle 25: Übersicht über die Ablage der Endergebnisse

C Evaluationsreport

Nachfolgend ist der Evaluationsreport, welcher in der Bachelorarbeit von Nadine Moser und Robin Meier entwickelt wurde [23], eingefügt. Die darin ausgewiesenen Modelle werden als *TrCNN* und *TrCNNCTC* bezeichnet. Die Resultate können leicht von den in Kapitel 8 angegebenen Werten abweichen, da sie auf den in Abschnitt 6.4 beschriebenen Auswertungen beruhen.

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: 02. June 2026 11:30

Summary

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

provider	total character accuracy	total ply accuracy
CNN	78.4%	42.2%
Google	83.8%	54.7%
Google2	83.8%	54.8%
TrCNN	95.1%	83.6%
TrCNNCTC	93.0%	77.0%
ChessReader	90.3%	73.9%

Intersection over union average over entire corpus: **100.0%**

Providers

The following OCR provider were used to generate predictions for the plies in the scoresheets:
CNN, Google, Google2, TrCNN, TrCNNCTC, 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:

- **CNN** - Selfmade Convolutional Neural Network
- **Google** - Google Vision API Document Text Detection
- **Google2** - Google Vision API Text Detection
- **TrCNN** - Transformer OCR (Encoder-Decoder)
- **TrCNNCTC** - TrCNNCTC (CNN-Encoder + CTC Head)
- **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
has shadows	1
is sheet incomplete	3

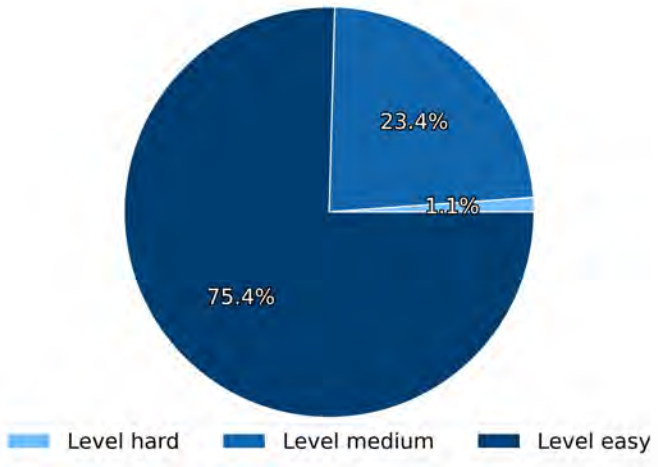
Handwriting Properties

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

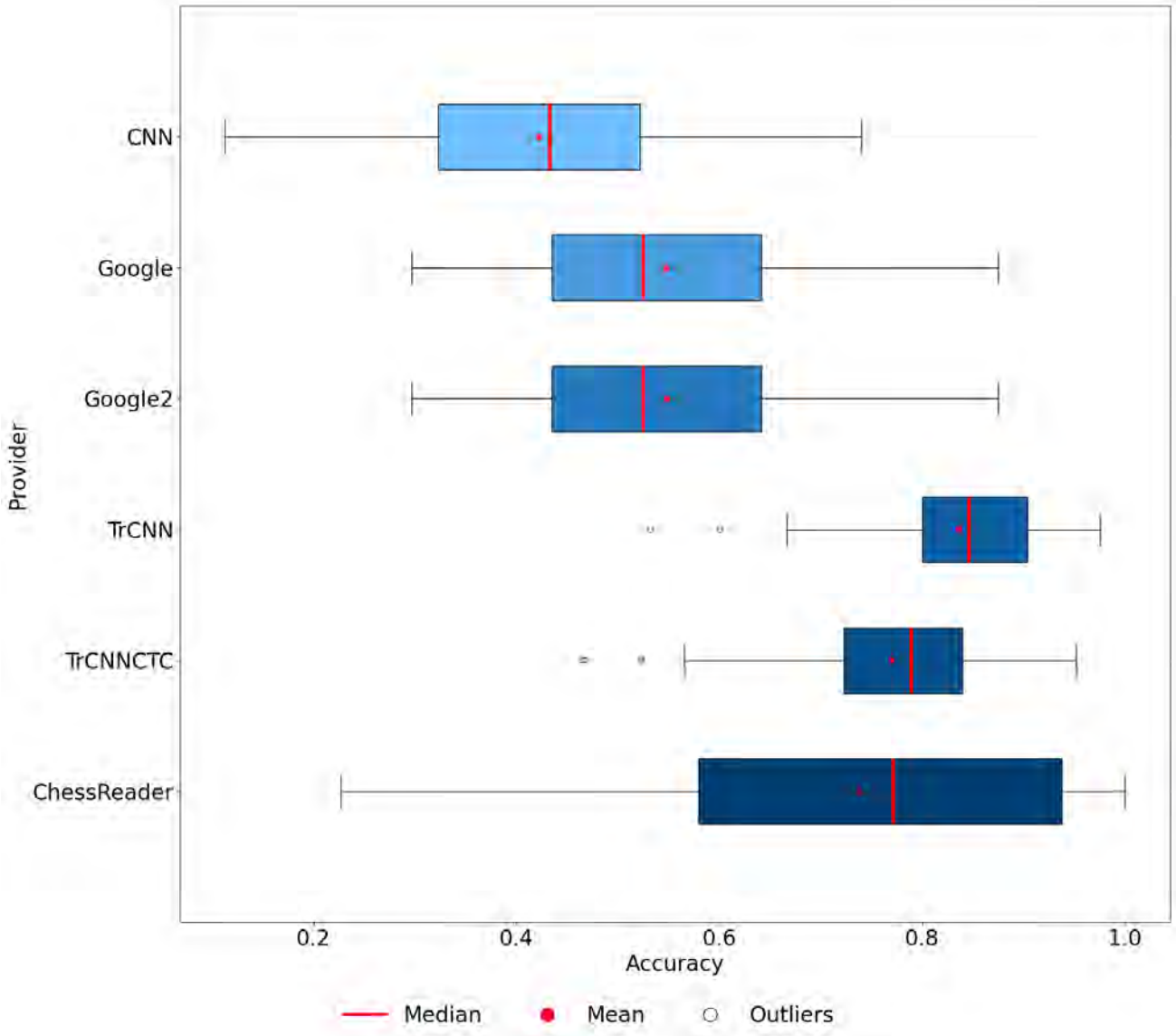
Corpus Legibility Levels



Accuracy

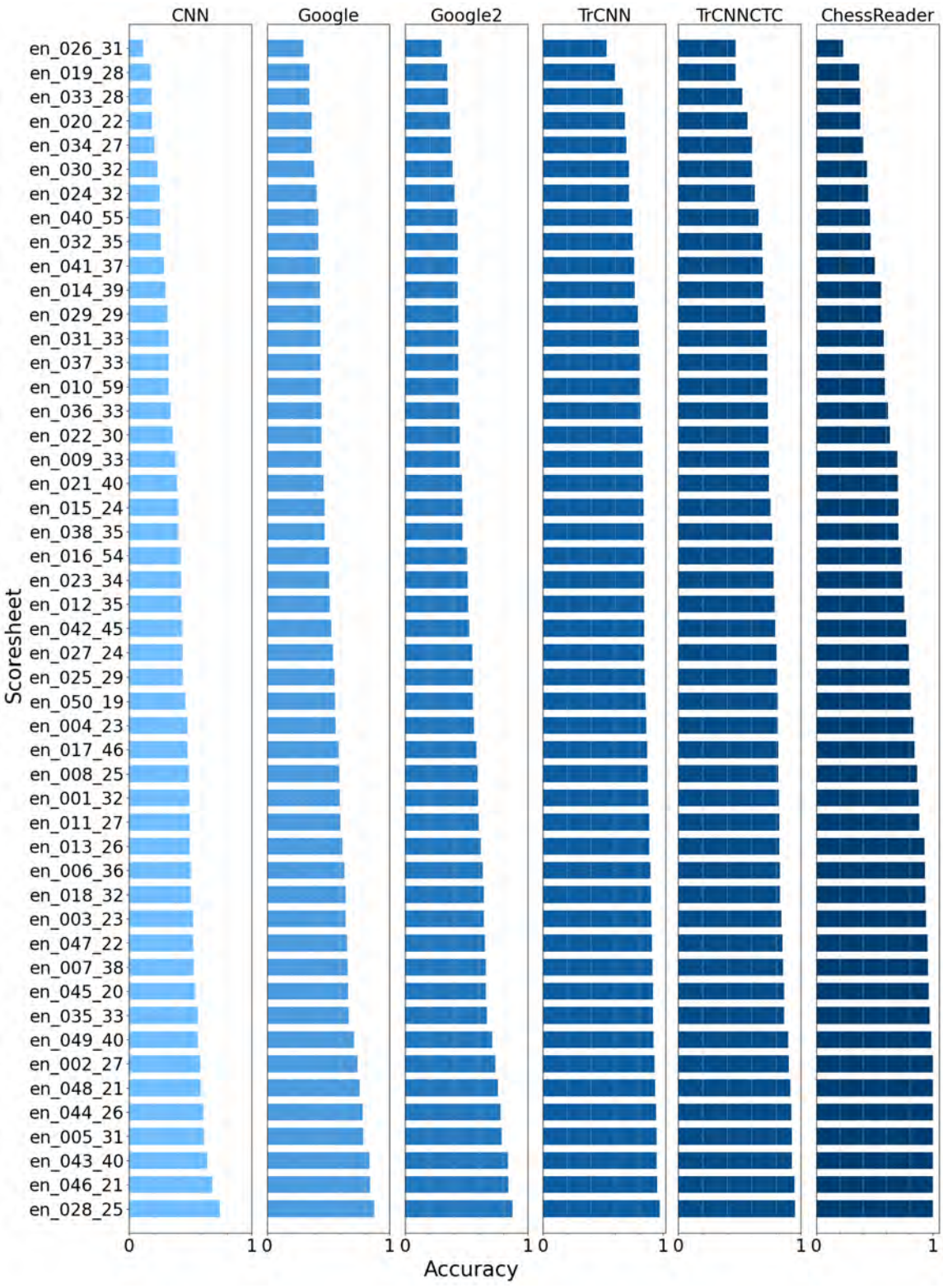
The following data describes the ply and character accuracy metrics respectively for each provider.

Ply Accuracy Per Provider

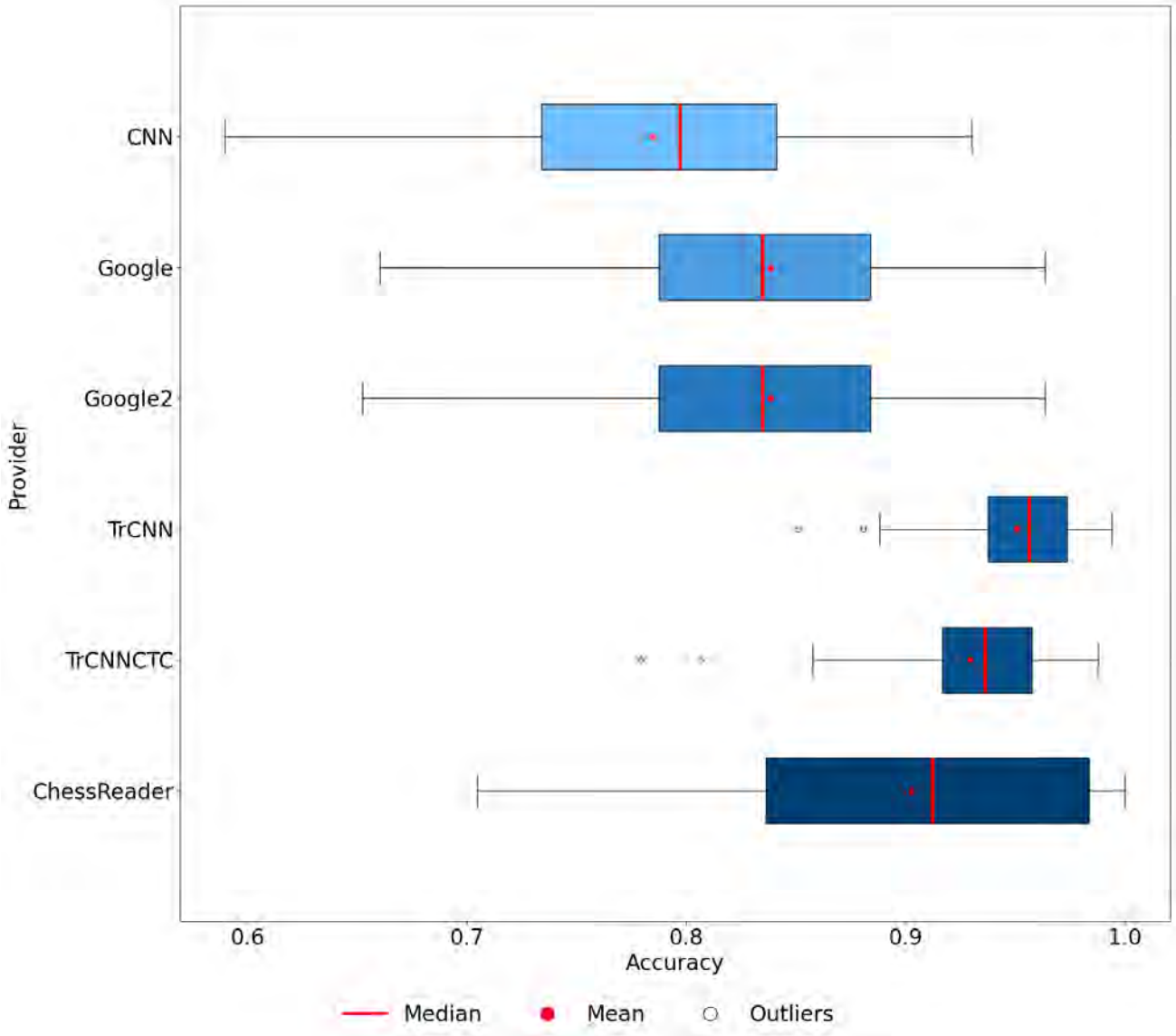


provider	total ply accuracy	total ply accuracy median	total ply accuracy std
CNN	42.2%	43.3%	14.2%
Google	54.7%	52.5%	14.3%
Google2	54.8%	52.5%	14.3%
TrCNN	83.6%	84.6%	9.3%
TrCNNCTC	77.0%	78.9%	11.6%
ChessReader	73.9%	77.2%	21.7%

Ply Accuracy Per Scoresheet Per Provider

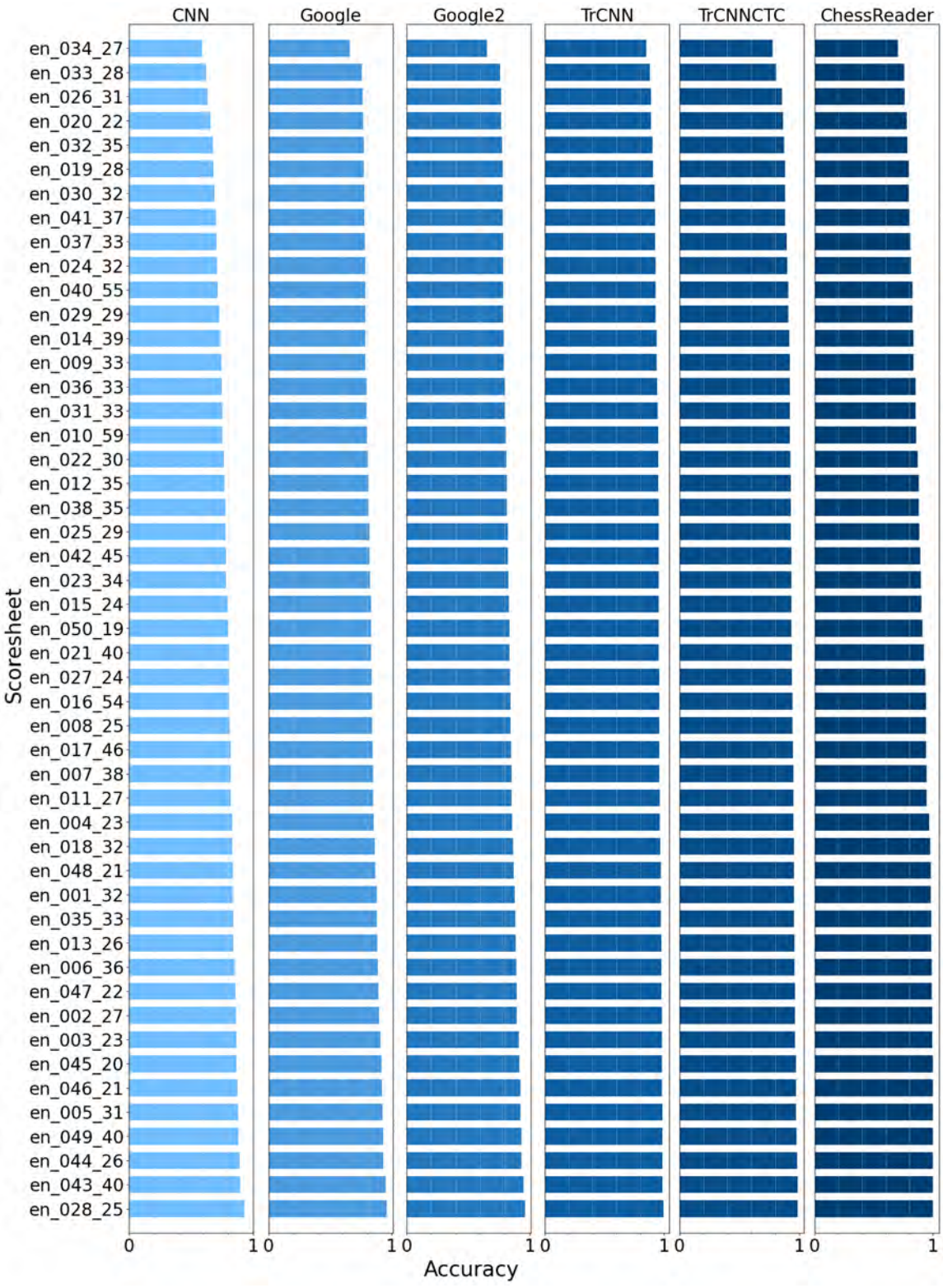


Character Accuracy Per Provider



provider	total character accuracy	total character accuracy median	total character accuracy std
CNN	78.4%	79.7%	7.7%
Google	83.8%	83.5%	6.0%
Google2	83.8%	83.5%	6.1%
TrCNN	95.1%	95.7%	3.1%
TrCNNCTC	93.0%	93.6%	4.3%
ChessReader	90.3%	91.2%	8.2%

Character Accuracy Per Scoresheet Per Provider



Box Accuracy

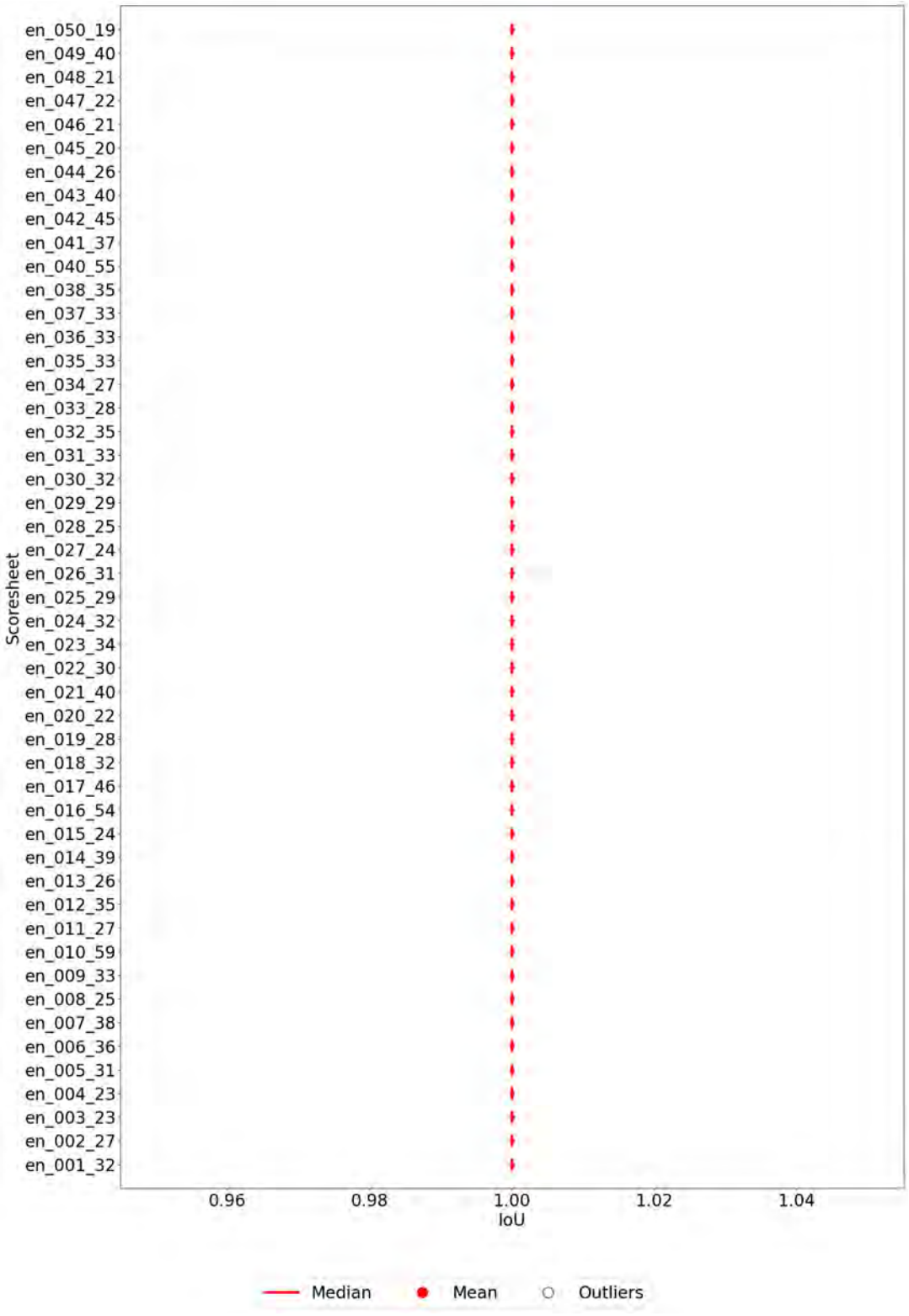
The following data describes the accuracy of the box predictions which is defined as the intersection over union of the box prediction and label.

The box accuracies were calculated using the following formula:

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

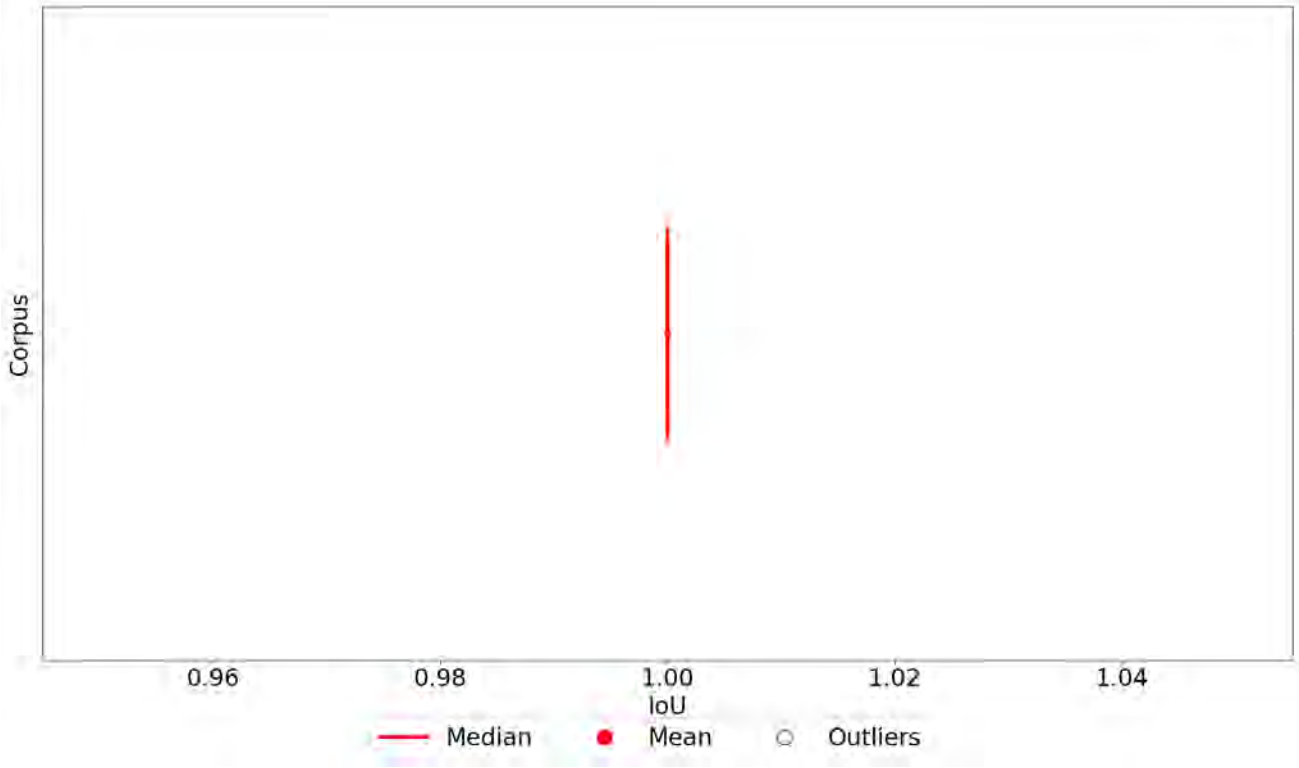
$$Box_l = \text{The box label from the corpus}$$
$$Box_p = \text{The box prediction from ChessReader}$$

Box Accuracy Per Scoresheet



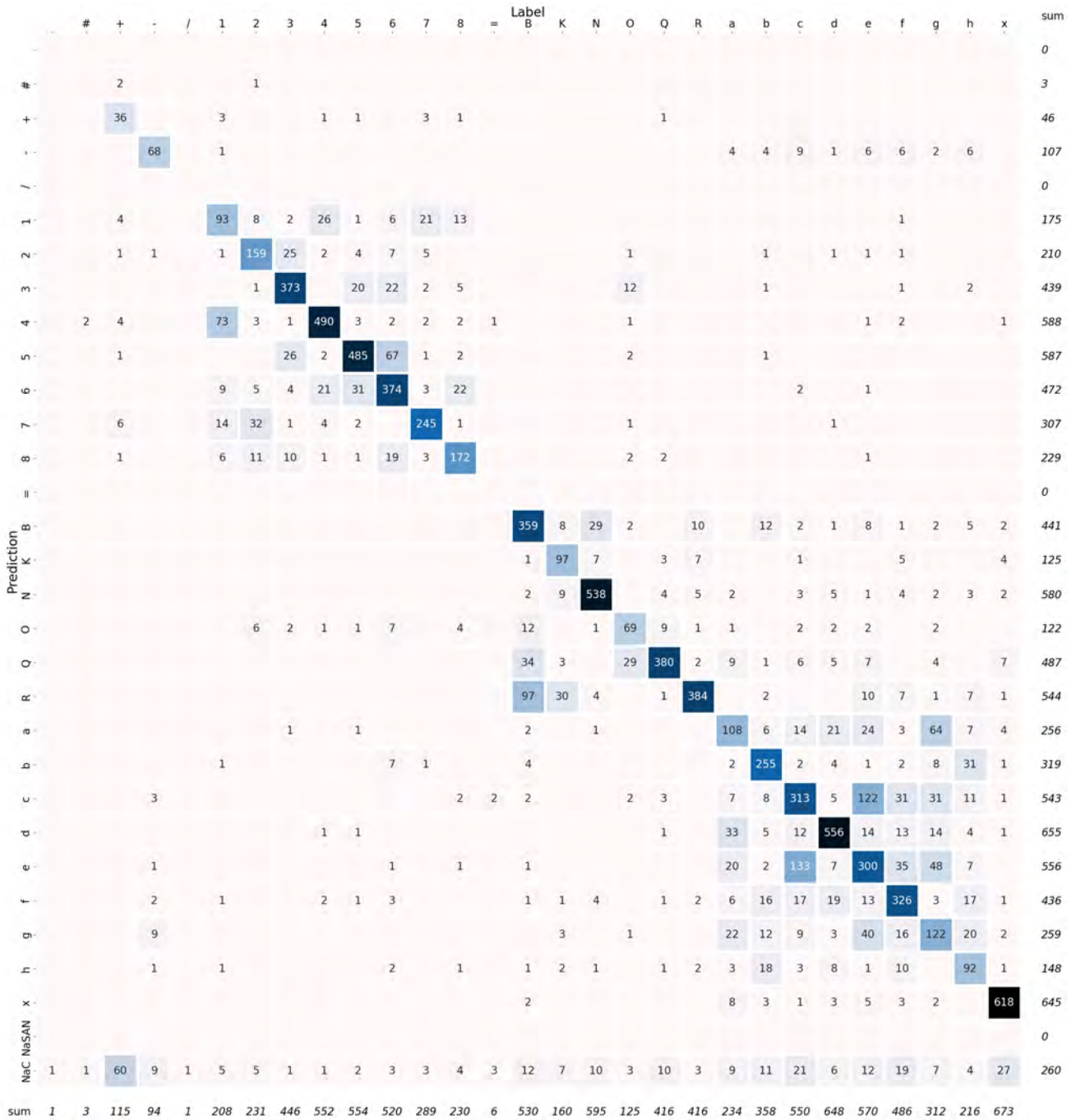
Box Accuracy Over Corpus

Intersection over union average over entire corpus: **100.0%**

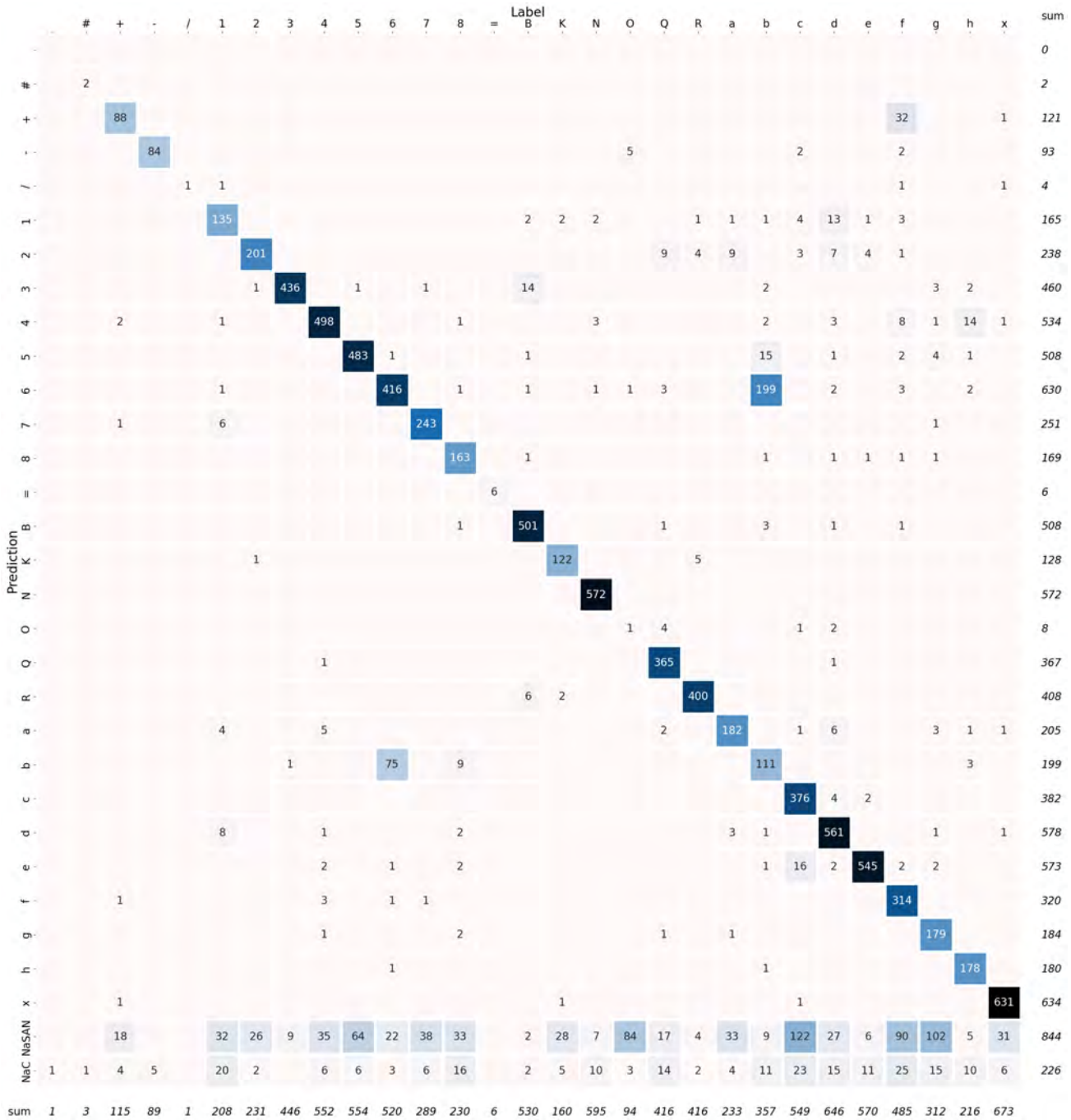


Character Confusion

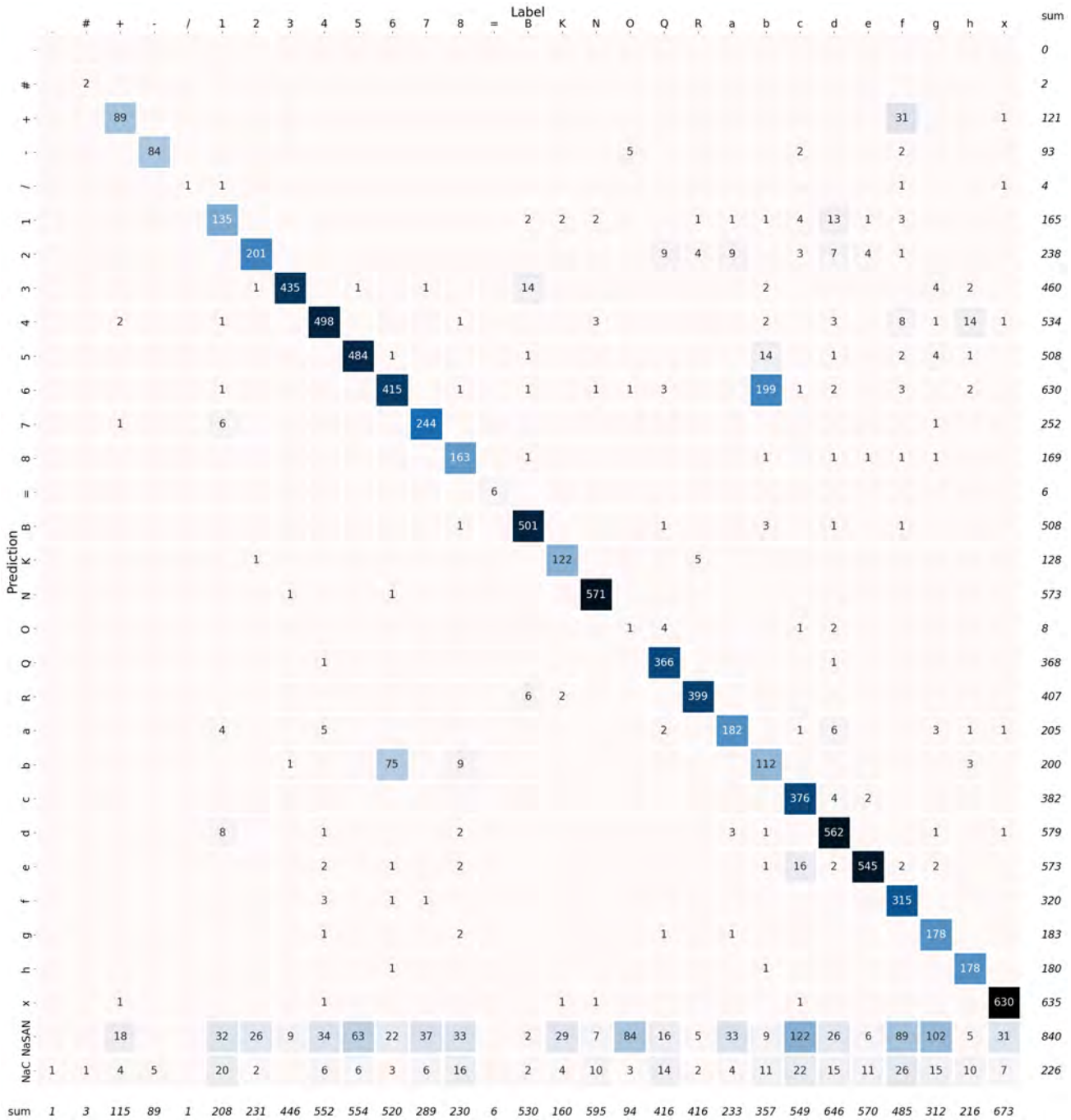
Cnn Character Confusions



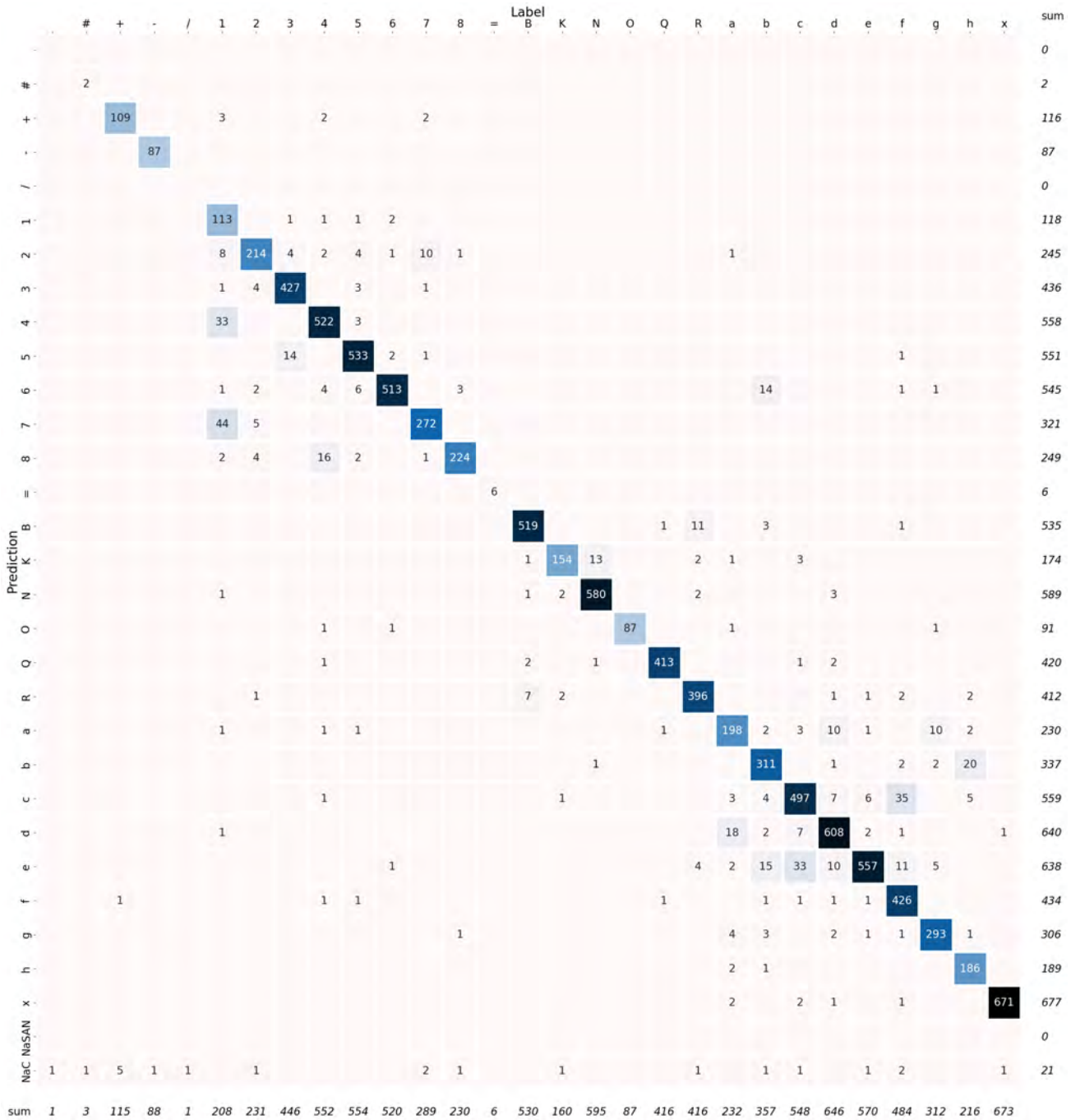
Google Character Confusions



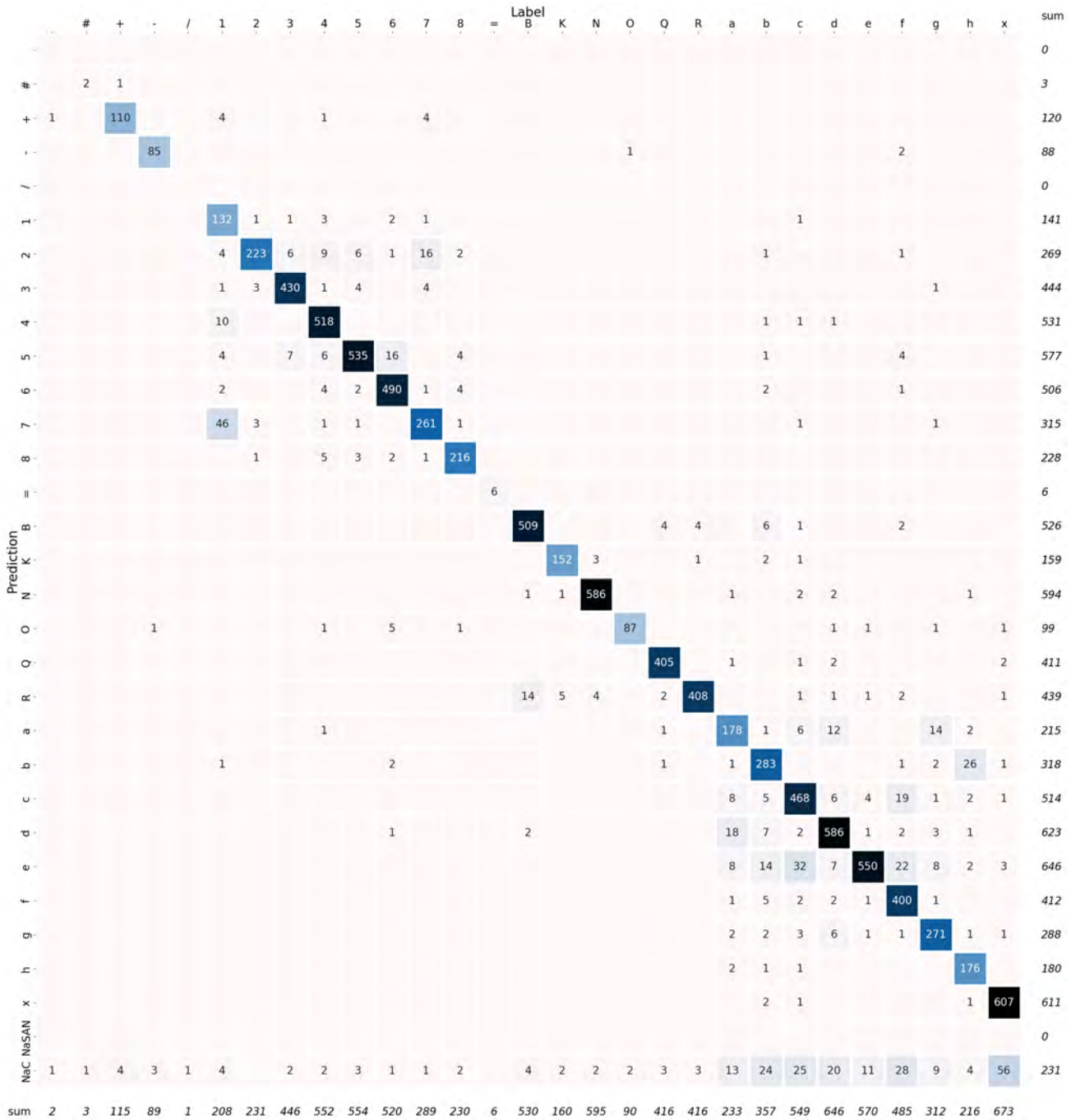
Google2 Character Confusions



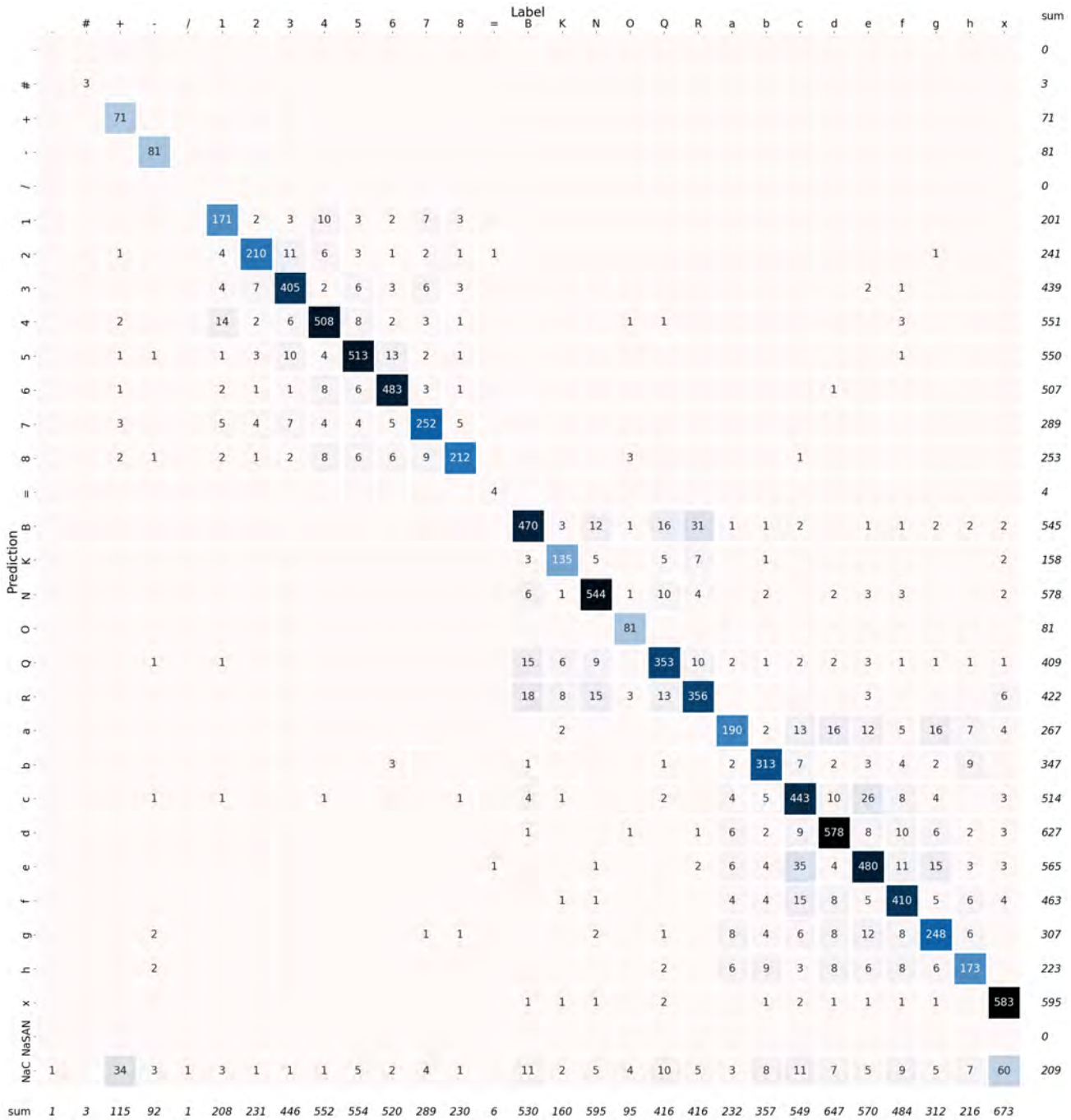
Trcnn Character Confusions



Trcnnctc 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:

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

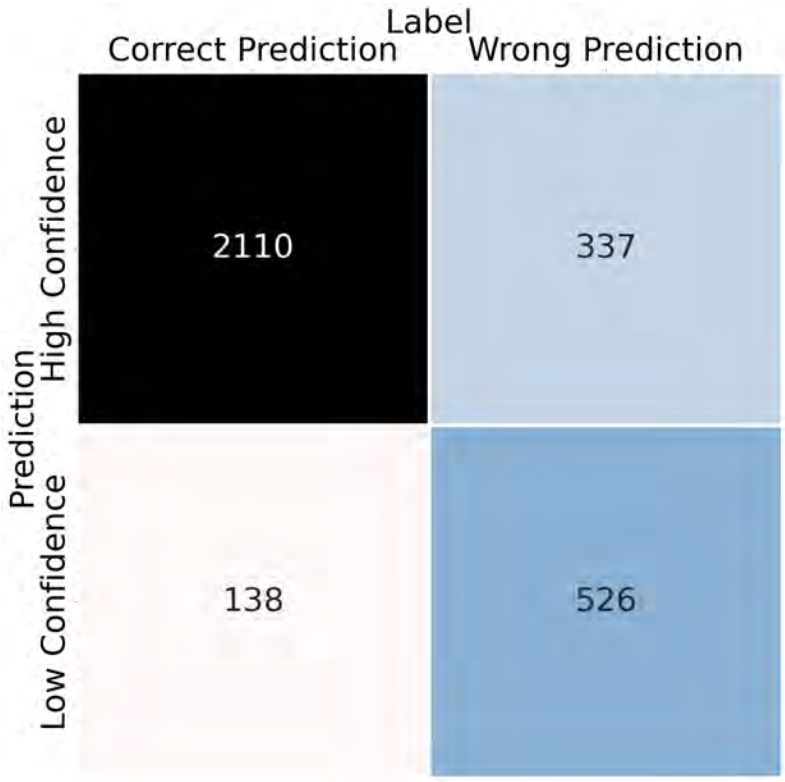
$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

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

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

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

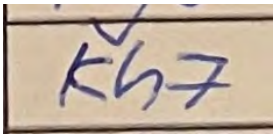
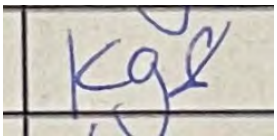
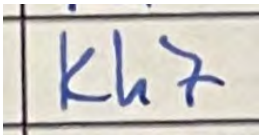
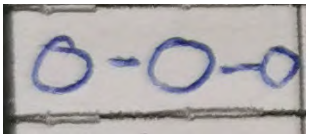
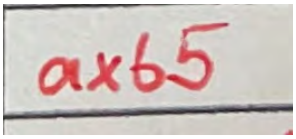
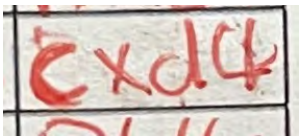
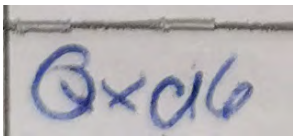
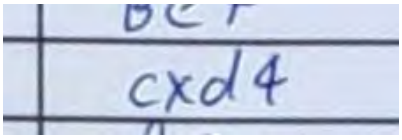
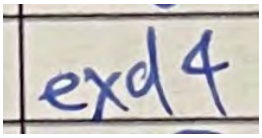
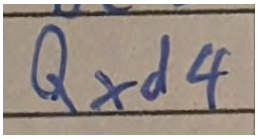
Confidence Confusion



accuracy	precision	recall	f1
84.7%	86.2%	93.9%	89.9%

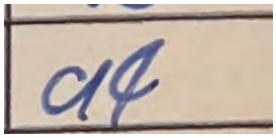
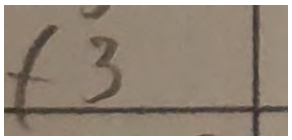
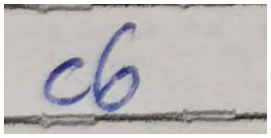
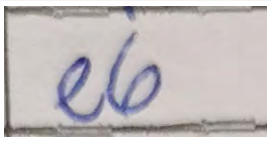
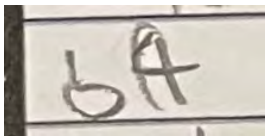
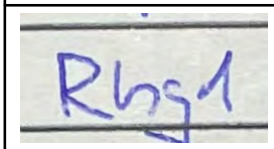
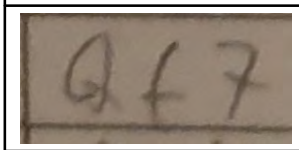
Top False Positive Results

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

ply box image	scoresheet	ply index	prediction	label	confidence
	english/en_007_38	75	Kxh8	Kh7	100%
	english/en_024_32	57	Kxg8	Kg8	100%
	english/en_038_35	43	Kg8	Kh7	100%
	english/en_041_37	22	O-O	O-O-O	100%
	english/en_008_25	36	Bxb5	axb5	100%
	english/en_034_27	13	Qxd4	cxd4	100%
	english/en_040_55	33	Qxd6	Qxa6	100%
	english/en_050_19	17	exd4	cxd4	100%
	english/en_038_35	63	exd2	exd4	100%
	english/en_017_46	14	Nxd4	Qxd4	100%

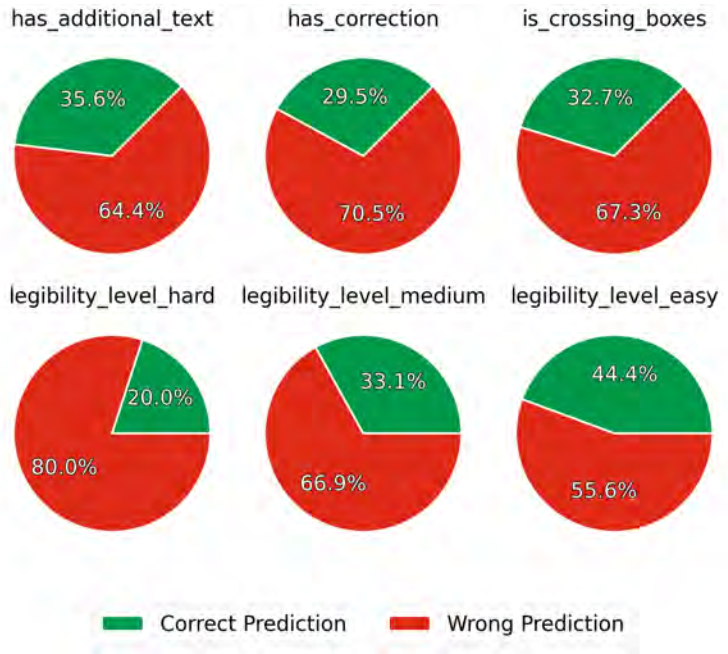
Top False Negative Results

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

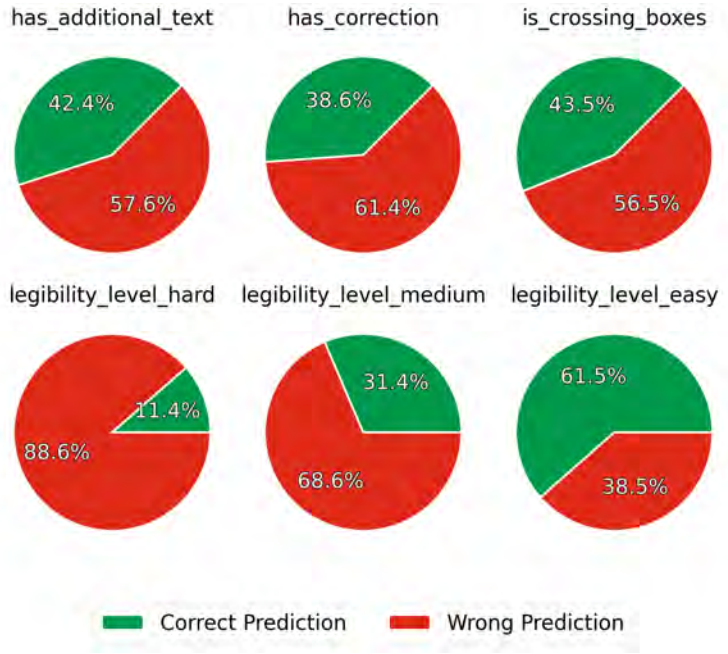
ply box image	scoresheet	ply index	prediction	label	confidence
	english/en_009_33	37	a4	a4	32%
	english/en_019_28	22	f3	f3	36%
	english/en_023_34	23	Nc2	Nc2	36%
	english/en_041_37	25	c6	c6	43%
	english/en_041_37	3	e6	e6	43%
	english/en_037_33	26	b4	b4	45%
	english/en_022_30	36	Qb5	Qb5	47%
	english/en_009_33	36	Re4	Re4	48%
	english/en_002_27	46	Rhg1	Rhg1	49%
	english/en_020_22	35	Qf7	Qf7	51%

Handwriting

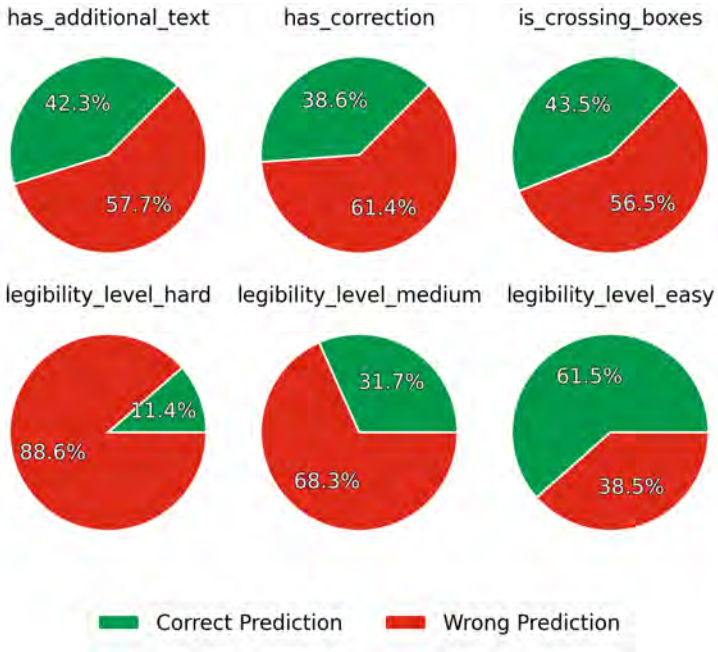
Handwriting Property Recognition For Cnn Provider



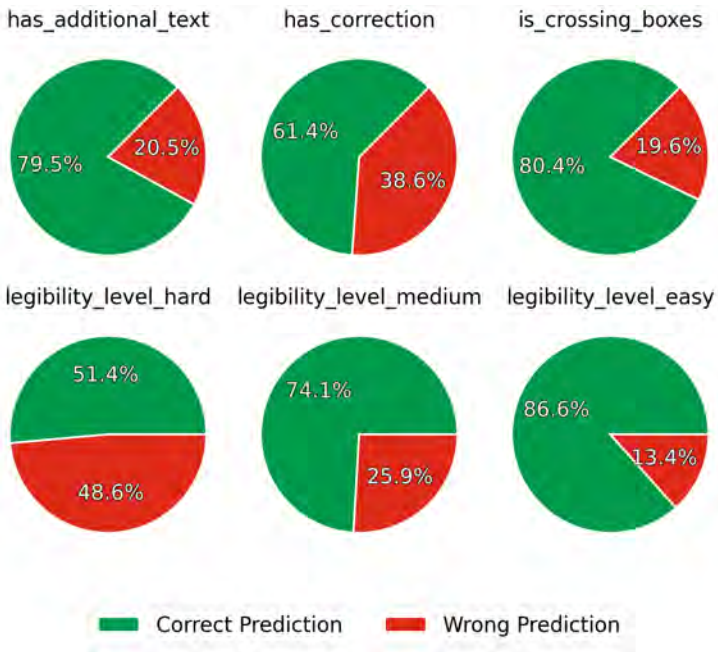
Handwriting Property Recognition For Google Provider



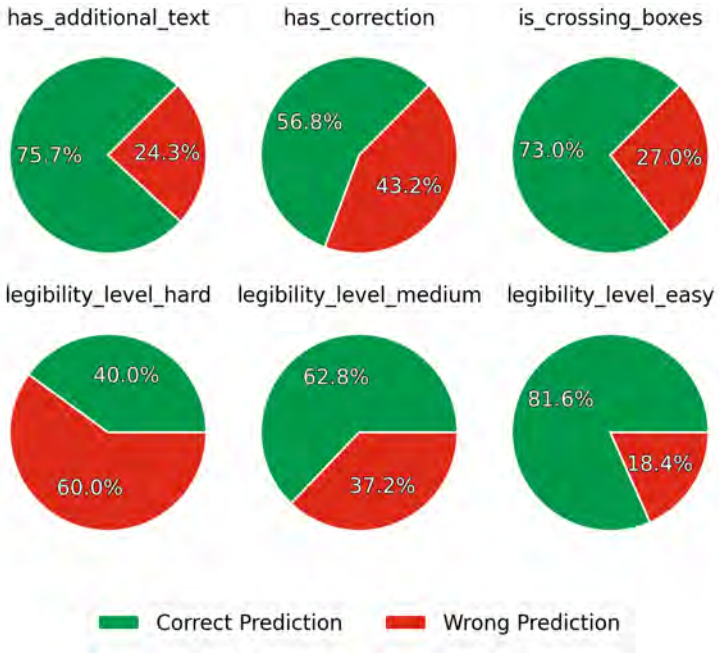
Handwriting Property Recognition For Google2 Provider



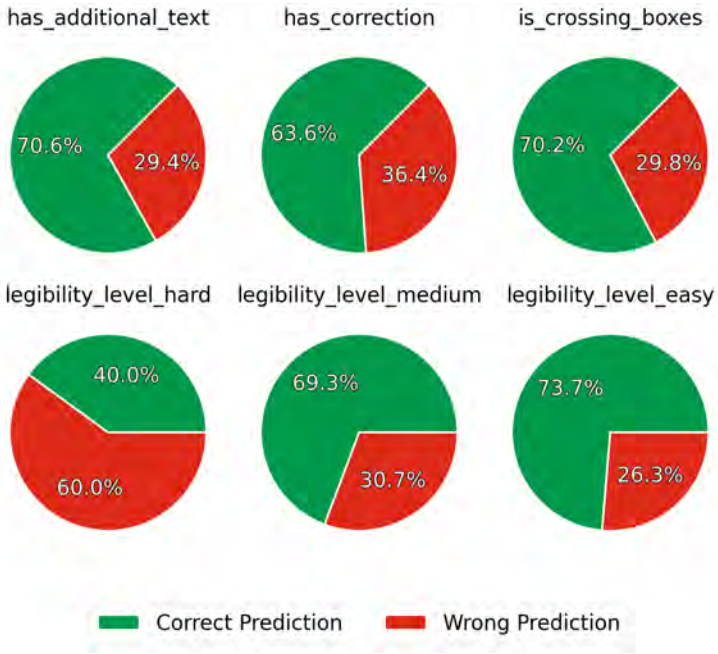
Handwriting Property Recognition For Trcnn Provider



Handwriting Property Recognition For Trcnnctc Provider

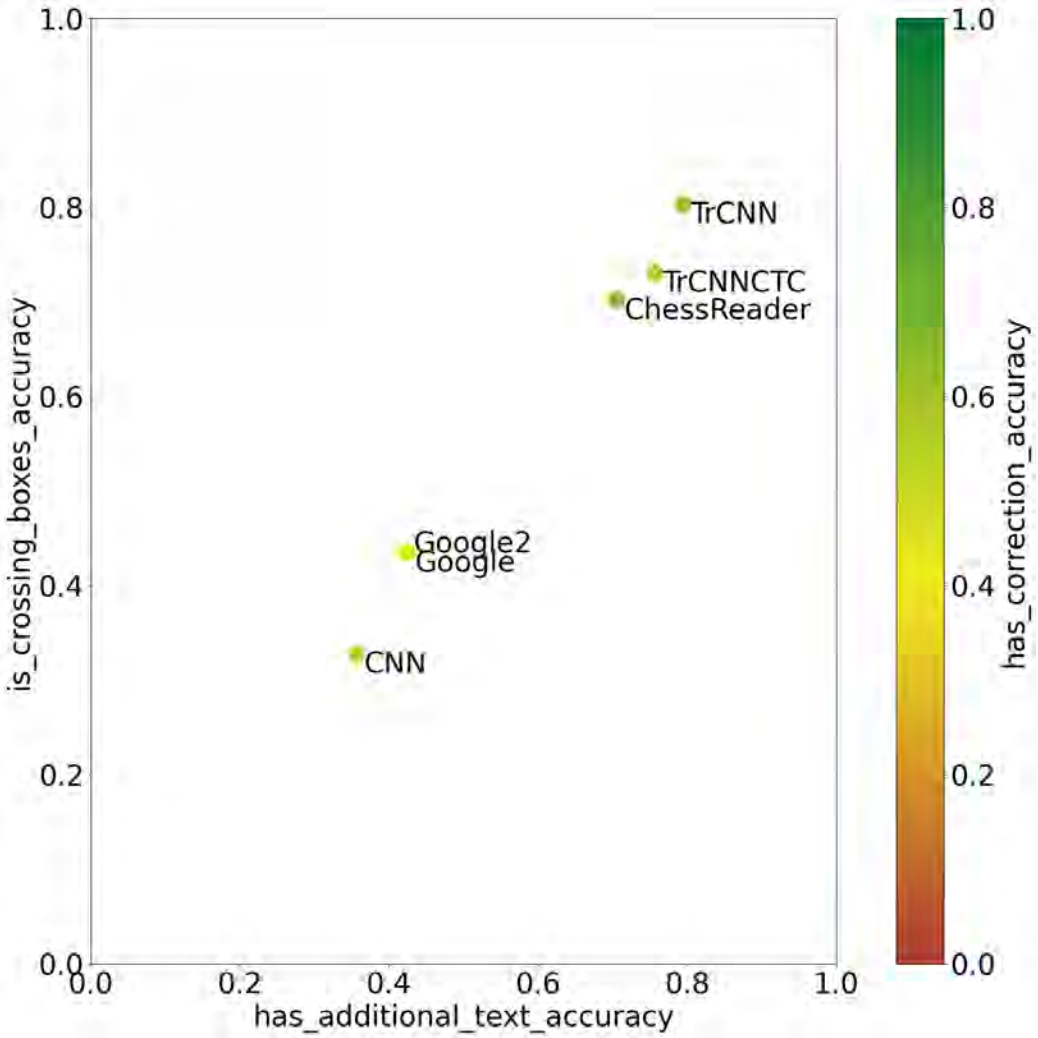


Handwriting Property Recognition For Chessreader Provider



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
CNN	35.6%	29.5%	32.7%	20.0%	33.1%	44.4%
Google	42.4%	38.6%	43.5%	11.4%	31.4%	61.5%
Google2	42.3%	38.6%	43.5%	11.4%	31.7%	61.5%
TrCNN	79.5%	61.4%	80.4%	51.4%	74.1%	86.6%
TrCNNCTC	75.7%	56.8%	73.0%	40.0%	62.8%	81.6%
ChessReader	70.6%	63.6%	70.2%	40.0%	69.3%	73.7%

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.