Institut für Mehrphasenprozesse
Prof. Prof. h.c. Dr.-Ing. Birgit Glasmacher, M.Sc.
An der Universität 1 | Gebäude 8143 | 30823 Garbsen

Studienarbeit

Automatisierte Erfassung und Extraktion von Daten aus Tabellen in Dokumenten

Hui Chen
Matrikel-Nr.: 10048521

Betreuung:
Tarek Deeb, M. Sc.
Dr. -Ing. Holger Zernetsch

Hannover, Dezember 2022

Institut für Mehrphasenprozesse
An der Universität 1, 30823 Garbsen, Prof. Prof. h.c. Dr.-Ing. Birgit Glasmacher

Leibniz Universität Hannover

Studienarbeit

für

Hui Chen

Matr.-Nr.: 10048521

Thema: Automatisierte Erfassung und Extraktion von Daten aus Tabellen in Dokumenten.

Aufgabenstellung

Daten sind die Grundlage für Informationen und somit die Generierung von Wissen. Große Teile unserer Daten werden nach wie vor in Form von Tabellen innerhalb eines Dokuments publiziert, um eine spezifische These zu bestätigen oder zu widerlegen - ein Beispiel hierfür sind wissenschaftliche Veröffentlichungen. Neben der Verwendung für die ursprüngliche Fragestellung lassen sich Daten auch für andere Thesen nutzen. Dazu müssen sie jedoch zunächst extrahiert, normalisiert und klassifiziert werden.

Im Rahmen dieser Arbeit soll daher ermittelt werden, welche Technologien zur Extraktion von Daten bzw. Tabellen aus Dokumenten existieren. Im nächsten Schritt soll zu diesem Zweck ein Python-Skript entwickelt und getestet werden. Hierbei sind bestehende Algorithmen bzw. Programmbibliotheken zur Bildverarbeitung zu nutzen. Nach dem erfolgreichen Einsatz mit einem Beispieldokument soll das Skript auf die Stapelverarbeitung mehrerer Dokumente erweitert werden. Die extrahierten Daten sollen in eine Datenbank geladen werden.

Es ist abschließend zu diskutieren wie eine Normalisierung und Klassifizierung dieser Daten durchzuführen ist. Ziel ist es, Daten aus verschiedenen Dokumenten vergleichen zu können.

Die Ergebnisse der Studienarbeit sind in vorgegebener Form zu dokumentieren, anhand von Literaturdaten zu diskutieren und im Rahmen einer Abschlusspräsentation am Institut für Mehrphasenprozesse vorzustellen.

Beginn der Arbeit: 01.08.2022

Institutsleitung	Betreuerin/Betreuer	Zweitbetreuerin/Zweitbetreuer

Der Zeitraum zur Bearbeitung der Studienarbeit beträgt sechs Monate ab dem Beginn der Arbeit. Für die Einhaltung der Bearbeitungszeit sowie des angefügten Zeitplans ist der Student/die Studentin selbst verantwortlich. Bei Abweichungen ist rechtzeitig Kontakt zur Betreuung zu suchen.

Erklärung

Hiermit erkläre ich, Hui Chen, dass diese Arbeit selbstständig und nur mit den angegebenen Hilfsmitteln und Quellen angefertigt wurde. Ich habe diese Arbeit weder im In- noch im Ausland in gleicher oder ähnlicher Form einer Prüfungsbehörde vorgelegt.

Ort, Datum

Unterschrift

Inhaltsverzeichnis

Aufgabenstellung II

Erklärung III

Abbildungsverzeichnis VII

Tabellenverzeichnis VIII

Formelzeichen IX

Abkürzungen X

1. Einleitung 1
   1.1. Motivation 1
   1.2. Ziel der Arbeit 2
   1.3. Struktur der Arbeit 2

2. Grundlagen 4
   2.1. Daten 4
   2.2. Datenbanken 5
   2.3. Python 6
   2.4. Digitale Bildverarbeitung 8
   2.5. Maschinelles Lernen und neuronale Netze 9

3. Stand der Technik 13
   3.1. Tabellenerkennung 13
   3.2. Texterkennung 17

4. Implementierung 21
   4.1. Vorverarbeitung 21
   4.2. Tabellenbereichserkennung 23
      4.2.1. Linienerkennungsbasierte Tabellenbereichserkennung 24
      4.2.2. Textblöckebasierte Tabellenbereichserkennung 25
      4.2.3. Tabellenbereichserkennung basierend auf tiefen, neuronalen Netzen . 27

4.3. Tabellenstrukturanalyse 33
4.3.1. Linienrekonstruktionsbasierte Tabellenstrukturanalyse 33
4.3.2. Textblöckebasierte Tabellenstrukturanalyse 35
4.3.3. Tabellenstrukturanalyse mit Unterstützung neuronaler Netze 40
4.4. Texterkennung und Tabellenrekonstruktion 42
4.5. Nachverarbeitung und Einschreibung in Datenbank 43

5. Ergebnisse und Auswertung 55
5.1. Auswertung von DenseNet Modell und U-net Modell 55
5.2. Tabellenbereichserkennung 58
5.3. Tabellenstrukturanalyse 59

6. Diskussion 62

7. Zusammenfassung 63

8. Ausblick 65

Literaturverzeichnis 66

A. Anhang XI

Abbildungsverzeichnis

4.1. Koordinatensystem auf Bild. 21
4.2. Workflow von Vorverarbeitung. 23
4.3. Beispiele der Tabellenvarianten. 24
4.4. Workflow linienerkennungsbasierter Tabellenbereichserkennung. 25
4.5. Workflow textblöckebasierter Tabellenbereichserkennung. 26
4.6. Filtern der Blöcke. 27
4.7. Beispielbild im Datensatz zusammen mit den entsprechenden Masken. 28
4.8. Einsetzung des besten Modells und die Regularisierung der Vorhersage. . . . 29
4.9. Struktur des geänderten, auf U-net[RFB15] basierten Modells. 30
4.10. Struktur des *DenseNet-121-BC (k=32)* Backbones nach Huang[HLVDMW17]. . 31
4.11. Modell mit TableNet-Architektur basierend auf dem *DenseNet-121-BC (k=32)* Backbone nach Sagar[Sag21]. 32
4.12. Umsetzungsprozess linienrekonstruktionsbasierter Tabellenstrukturanalyse auf der Beispieltabelle 4.1. 34
4.13. Beispielbild, das zufällig zu falschen Lückenerkennungsergebnissen führen kann, und das Filtern 35
4.14. Umsetzungsprozess textblöckebasierter Tabellenstrukturanalyse auf der Beispieltabelle 4.1. 36
4.15. Beispieltabelle mit linksbündigen, rechtsbündigen und zentrierten Zellen sowie die Lokalisierung der Zellen. 37
4.16. Workflow zur Bestimmung des besten k-Werts. 38
4.17. Visulalisierung des K-Means-Clusterings auf der Tabelle 4.15. 39
4.18. Korrektur der Ergebnisse von K-Means-Clustering auf der Tabelle *Wochenbericht 2022-01-06 24*. 40
4.19. Tabellenstrukturanalyse mit neuronalen Netzen auf der Tabelle 4.15. 41
4.20. Texterkennung und Tabellenrekonstruktion auf einer Beispieltabelle. 43
4.21. Vergleichung von einfachen und komplexen Tabellen und die Benennung der Teile. 44
4.22. Visualisierung der achten Regeln und die Schmelzung. 45
4.23. Visualisierung der Regeln. 45
4.24. Komplexe Tabelle wegen zu langen Text. 46
4.25. Komplexe Tabelle wegen mehrstufigen Köpfe. 47
4.26. Visualisierung vom vertikalen Schmelzen. 48

4.27. Horizontales Schmelzen der Zeilenköpfe. 49
4.28. Übliche komplexe Köpfe mit drei Zeilen. 50
4.29. Visualisierung der vorläufigen und endgültigen Eignungsprüfung. 50
4.30. Loops für die Diffusion der qualifizierten primären Köpfe. 52
4.31. Beispieltabelle, in der die leeren Zellen nicht alle ausgefüllt werden. 53
4.32. Vertikales Schmelzen der Tabellenköpfe. 53
4.33. JSON-Objekt der Abb. 4.32 (a). 54
4.34. Programmablaufplan der Nachverarbeitung. 54

5.1. Verlustkurven der Modelle im Training. 56
5.2. Visualisierung der Berechnung von Intersection Over Union. 56
5.3. Vergleich der Vorhersagen der Modelle und Darstellung der Nachverarbeitung. 57
5.4. Verwirrung in der linienbasierten Tabellenbereichserkennung. 59
5.5. Einfluss der Kerngröße auf die Dilatation. 59
5.6. Verarbeitung der Spalten unter dem spaltenübergreifenden Kopf auf Bild *Wochenbericht_2022-01-06_6*. 61
5.7. Ergebnisse bei Pipeline 1, 2 und 3 auf Bild *2023-01-02-de_2*. 61

Tabellenverzeichnis

4.1. Funktionen zur Binarisierung und Rauschunterdrückung in *OpenCV*. 22
4.2. Statistik der Textblöcke in der Tabelle 4.18 (a). 39

5.1. Mean Intersection Over Union (mIoU) des *DenseNet-121-BC* und *U-net* Modells bei Tabellenbereichserkennung und Spaltenbereichserkennung auf 85 Bildern. 57
5.2. Vergleich der drei Verfahren bei Tabellenbereichserkennung. 58
5.3. Durchschnittliche Zellgenauigkeitsrate nach drei Pipelines und der mittele Zeitaufwand. 60

Formelzeichen

Formelzeichen	Einheit	Erläuterung
X_l	1	Ausgabe der l^{ten} Schicht in einem Dense-Block von DenseNet-121-BC
H_l	1	Kombinationsfunktion (nicht-linearer Transformationsfunktion) der l^{ten} Schicht
$s_k(j)$	1	Silhouettenkoeffizient unter dem k-Wert in K-Means-Clustering
$a(j)$	1	Cohesion von Clustering
$b(j)$	1	Separation von Clustering
$\bar{s}_k$	1	Mittelwert $s_k(j)$ aller Daten im gesamten Datensatz
SC	1	Maximale $\bar{s}_k$
RZ	1	Anzahl der Zellen mit richtigen Inhalt im Values-Teil, Köpfe-Teil oder Zeilenköpfe-Teil
ZN	1	Anzahl der Zeilen im entsprechenden Teil
SN	1	Anzahl der Spalten im entsprechenden Teil
μ	1	Mittelwert statistischer Daten
N	1	Menge statistischer Daten
X_i	1	Jede statistische Data
s	1	Standardabweichung der Grundgesamtheit bei statistischen Daten
R	1	Mittelwert unter Three-Sigma Limit

Abkürzungen

Abkürzung	Bezeichnung
IMP	Institut für Mehrphasenprozesse
VDI	Verein Deutscher Ingenieure
CPU	Central Processing Unit
MDP	Markov-Entscheidungsprozess
KNN	künstliche neuronale Netze
RNN	rekurrente neuronale Netze
CNN	convolutionale neuronale Netze
MLP	Multi-Layer Perceptron
FC	Fully Connected Layers
LRD	Long-range dependence
LSTM	Long Short-Term Memory Network
GRU	Gated Recurrent Unit
DBMS	Database Management System
SQL	Structured Query Language
NoSQL	Not-only-SQL
ACID	Atomicity, Consistency, Isolation, Durability
KI	Künstliche Intelligenz
AutoMLP	Automatic Tunable Multilayer Perceptron
GNN	Graph Neural Networks
FCN	Fully Convolutional Networks
OCR	Optical Character Recognition
ICR	Intelligent Character Recognition
IWR	Intelligent Word Recognition
CTPN	Connectionist Text Proposal Network
EAST	Efficient and Accurate Scene Text Modell
PSENet	Progressive Scale Expansion Network
CTC	Connectionist Temporal Classification
BLSTM	Bidirectional Recurrent Neural Networks
STN	Spatial Transformer Network
CA-FCN	Character Attention Fully Convolutional Network

ResNet	Residual Neural Network
ROI	Interessengebiet
XML	Extensible Markup Language
JPL	Jet Propulsion Laboratory
CT	Computertomographie
SIFT	Scale-invariant Feature Transform
PCA	Principal Component Analysis
HoG	Histogram of oriented Gradients
SVM	Support Vector Machines
KI	Künstliche Intelligenz
MDP	Markov-Entscheidungsprozess
GFTE	Graph-based Financial Table Extraction
LSD	Line Segment Detector
FLD	Fast Line Detector
RKI	Robert Koch Institut
GPU	Graphics Processing Unit
Adam	Adaptive Moment Estimation
BCE	Binary CrossEntropy
Conv	Faltung Schicht
BN	Batchnormalisierung
ReLU	Rectified Linear Unit
AvgPooling	Average-Pooling
DenseNet-121-BC	DenseNet-121 mit Bottleneck Layers und Compression
T0	Anfängliche Transition-Layer im DenseNet-121-BC
T1, T2, ...	Erste, zweite, ... Transition-Layer im DenseNet-121-BC
D1, D2, ...	Erste, zweite, ... Dense-Block im DenseNet-121-BC
ROI	Interessengebiet
PA	Pixel Accuracy
mIoU	Mean Intersection Over Union
IoU	Intersection Over Union
RF	Rezeptives Feld

1. Einleitung

1.1. Motivation

Daten sind eine Kombination physikalischer Symbole, die die Eigenschaften, Zustände und Zusammenhänge eines Objekts festhalten. Als Grundlage für Informationen sowie die Wissensgenerierung[Bod06] sind Daten neben dem ursprünglichen Thema auch für andere Fragestellungen zu verwenden. Dazu müssen sie jedoch zunächst extrahiert, normalisiert und klassifiziert werden.
Um Informationen strukturiert formulieren zu können, wird ein großer Teil der Daten in Form von Tabellen veranschaulicht und in Form von Bildern publiziert. Die Daten lassen sich in diesem Fall nicht direkt vom Computer abrufen, sondern müssen mittels optischer Zeichenerkennung extrahiert werden. Deshalb ist die genaue Identifizierung des Bereichs von Text, Tabellen und Abbildungen eine erforderliche Voraussetzung für die korrekte Extraktion jedes Wortes in der Tabelle. Außerdem lässt sich der Zusammenhang der Daten innerhalb komplexer Tabellen nicht allein durch optische Erkennung rekonstruieren. Infolgedessen sind weitere Bearbeitungen, insbesondere das Erhalten der Koordinaten jeder Zelle und die Analyse auf Kopfzeilen, dafür auch von entscheidender Bedeutung.
Insgesamt liegt die wesentliche Problematik bei der Erfassung und Extraktion von Daten aus Tabellen in Bilddokumenten in der Tabellenbereichserkennung und der Tabellenstrukturanalyse[HKLW02]. Hierzu wurden zahlreiche Forschungen und Methoden veröffentlicht. Die traditionellen Methoden[CK93, Kie98, Han00] basierten auf a priori bekannten Regeln oder manuell definierte Merkmale, um präzise Ergebnisse auf Tabellen mit einer bestimmten Struktur zu erzielen, die Robustheit und Universalität sind jedoch eingeschränkt. Aufgrund der rasanten Entwicklung von Deep Learning sind viele neue Verfahren[PVR+19, LHY+21] in den letzten zehn Jahren entstanden. Dieser auf neuronalen Netzwerken basierende Ansatz ist nicht nur auf die Verarbeitung der bestimmten Tabellen mit a priori bekannter Struktur beschränkt, sondern kann mit vielfältigen Tabellentypen umgehen. Fortgeschrittene Modelle von Deep Learning sind jedoch in der Regel rechenintensiv und wird daher langsam auf Central Processing Unit (CPU) durchgeführt[Sto22]. Darüber hinaus haben neuronalen Netzen eine begrenzte Interpretierbarkeit[GAZ19].

1.2. Ziel der Arbeit

Ziel dieser Arbeit ist es, ein *Python*-Skript zur Erfassung und Verarbeitung von Daten, die in tabellarischer Form dargestellt sind, zu entwickeln. Das Skript soll effizient offline auf CPU laufen können. Dazu konzentriert sich diese Arbeit auf drei Unteraufgaben: Tabellenbereichserkennung, Tabellenstrukturanalyse und Einschreibung in eine Datenbank. Die Tabellenbereichserkennung befasst sich mit dem Erkennen und Ausschneiden des Tabellenbereichs im Bilddokument. Bei der Tabellenstrukturanalyse ist es notwendig, ein geeignetes Verfahren zu finden, um jede Zelle zu lokalisieren und die Pixelinformationen in der Zelle in digitale Daten umzuwandeln, während die Positionsbeziehungen zwischen den Zellen beibehalten werden. Anhand intensiven Blicks auf den Stand der Technik sollen Verfahren zur Tabellenbereichserkennung und Tabellenstrukturanalyse erstellt werden, indem sie entweder auf traditionellen Methoden basieren oder sich auf neuronale Netzen stützen. Die Implementierungsprozesse dazu sollen detailliert beschrieben. Außerdem sollen die Ergebnisse und Limitierung diskutiert werden. In der Einschreibungsphase wird es erwartet, die Daten der Tabelle zu normalisieren und in einer Datenbank zu speichern. Darüber hinaus sollen entsprechende neuronale Netze für die Tabellenbereichserkennung und Tabellenstrukturanalyse trainiert werden.

1.3. Struktur der Arbeit

Nach einer Einleitung in das Thema werden im Kapitel 2 die Grundlagen dieser Arbeit vorgestellt und erläutert. Darin werden zunächst die allgemeine Bedeutung von Daten und Datenbanken sowie die Programmierung mit *Python* erklärt. Weiterhin werden die theoretischen Grundlagen der digitalen Bildverarbeitung und des maschinellen Lernens vorgestellt. Das Kapitel schließt mit einer Beschreibung von Deep Learning bzw. neuronalen Netzen, die in dieser Arbeit eine wichtige Rolle spielen.
Im 3. Kapitel wird ein intensiver Blick auf den Stand der Technik zur Erkennung von Formulardokumenten geworfen. Im Abschnitt 3.1 werden zuerst traditionelle Tabellenerkennungsmethoden vorgestellt, die auf a priori bekannten Regeln basieren, und dann werden neuere, auf Deep Learning basierende Methoden aufgezählt. Abschließend werden im Abschnitt 3.2 verschiedene Methoden zur Erkennung optischer Zeichen beobachtet.
Das Kapitel 4 befasst sich mit dem praktischen Teil und beschreibt verschiedene Algorithmendesigns und -implementierungen für die Vor- und Nachverarbeitung sowie die entsprechenden drei Unteraufgaben, die die vollständigen Tabellenerkennungsverfahren bilden. Dort wird erklärt, wie die Konzepte und Algorithmen in der Praxis umgesetzt werden können und welche Einschränkungen dabei vorgenommen werden müssen. Zu der Tabellenbereichserkennung und der Tabellenstrukturanalyse werden mehrere Alternativen vorgestellt und diskutiert, die jeweils auf angegebene Quellen zurückzuführen sind, aber teilweise geändert und optimiert werden. In der Nachverarbeitung werden durchdachte Regeln und Annahmen erstellt, um die extrahierten Tabellen zu normalisieren und sie vollständig in die Datenbank einzuschreiben.

Im Kapitel 5 befindet sich die Ergebnisse und die Bewertung. Hierbei werden die alle im Kapitel 4 genannten Verfahren durchgeführt, um sie an Beispielszenarien auf Tauglichkeit zu überprüfen. Die Laufzeit und die Genauigkeit des Algorithmus werden auf Testbeispielen untersucht und in Tabellen 5.2 und 5.3 verglichen. Der Leistungsunterschied der vier neuronalen Netze, die für die Tabellenbereichserkennung und Tabellenstrukturanalyse trainiert werden, ist in Abbildung 5.1 und 5.1 zu finden.
Eine Diskussion der Limitierung dieser Arbeit ist dann im Kapitel 6 zu finden. Am Ende dieser Arbeit steht im Kapitel 7 die Zusammenfassung bzw. das Fazit des Erarbeiteten. Weiterhin wird ein Ausblick auf weitere Arbeiten und mögliche Erweiterungen des Algorithmus im Kapitel 8 gegeben.

2. Grundlagen

Das vorliegende Kapitel behandelt die Grundlagen dieser Arbeit. Im ersten Abschnitt wird die allgemeine Bedeutung von Daten behandelt. Da die extrahierten und normalisierten Daten in einer Datenbank gespeichert werden sollten, stellt im zweiten Abschnitt einige heute gemeingebräuchliche Datenbanken vor. Im dritten Abschnitt wird die Programmierung mit *Python* vorgestellt, die in dieser Arbeit Verwendung findet. Die theoretischen Grundlagen der digitalen Bildverarbeitung werden im vierten Abschnitt erklärt. Der letzte Teil dieses Kapitels befasst sich mit maschinellem Lernen und neuronalen Netzen.

2.1. Daten

Daten als Träger von Informationen und Wissen[Bod06] werden aus Beobachtungen, Messungen, Experimenten oder Berechnungen gewonnen. Sie sind oft die Ergebnisse von Forschungen und können für weitere Arbeiten oder Widerlegungen zitiert werden. Im Zeitalter von Big Data beziehen sich Daten nicht nur auf Zahlen, sondern umfassen auch Sprache, Bilder, Grafiken, Animationen, Videos, Tonaufnahmen und andere Ausdrucksformen[HPT22].
Laut Statista Research Department betrug die Menge der erstellten, kopierten und verwendeten Daten im Jahr 2021 rund 79 Zettabyte, das Vierfache der Menge von vor fünf Jahren. Prognosen zufolge wird dieses Datenvolumen im Jahr 2025 181 Zettabyte erreichen[Dep22b]. Diese Datenexplosion erschwert die Datenerfassung und -analyse. Eine kleine Menge nützlicher Daten ist oft mit einer großen Menge chaotischer und nutzloser Daten durchsetzt. Zunächst sollte deshalb eine Klassifizierung der Daten vorgenommen werden, um klare, logische Beziehungen aus chaotischen Daten zu ermitteln.
Es gibt im Allgemeinen viele Kriterien, die für die Datenklassifizierung verwendet werden können. Ein klassisches Kriterium ist der Strukturierungsgrad der Daten. Somit werden die Daten in drei Kategorien unterteilt, nämlich strukturierte Daten, semistrukturierte Daten und unstrukturierte Daten[TaloJ]. Strukturierte Daten werden für einen effizienten Zugriff und Verwaltung in einem vorgegebenen Format strukturiert und verknüpft, wie zum Beispiel Daten in Excel-Tabellen und in SQL-Datenbanken. Unstrukturierte Daten, wie zum Beispiel Sprache, Bilder, Bürodokumente, Audio und Video, haben keine einheitliche Struktur und können daher nicht in einer relationalen Datenbank gespeichert werden. Semistrukturierte Daten sind zwar selbst unstrukturiert, aber mit Metadaten versehen, die auf bestimmte Merkmale verweisen. Diese Metadaten enthalten ausreichend Informationen, sodass eine effizientere Katalogisie-

rung, Suche oder Analyse vorgenommen werden kann[TaloJ]. Ein typisches Beispiel dafür ist Extensible Markup Language (XML).
Außerdem lassen sich die Daten auch nach dem Grad der Verarbeitung in originäre oder abgeleitete Daten unterscheiden. Originäre Daten sind erst- und einmalig vorhandene Daten, diese enthalten oft fehlende Werte, Rauschen und wiederholte Werte. Abgeleitete Daten zur Unterstützung von Schlussfolgerungen oder Entscheidungen müssen also zuvor aus den Originaldaten generiert werden. Darüber hinaus werden die Daten in der Regel nach dem Grad der Abstraktion bzw. Kapselung in Metadaten und Datenobjekt eingeteilt. Metadaten beschreiben die Attribute des Datenobjekts und legen fest, was mit dem Objekt passiert, wenn es aufgerufen wird[Ril17]. Ein Datenobjekt ist eine Sammlung von Datenelementen mit denselben Attributen. Durch Analyse und Extraktion der Metadaten und Datenobjekte innerhalb der zuvor erwähnten unstrukturierten und semistrukturierten Daten kann ein Datenmodell erstellt werden, um sie in strukturierte Daten umzuwandeln[Sar22].

2.2. Datenbanken

Eine Datenbank bezieht sich auf eine organisierte Sammlung von strukturierten Informationen oder Daten[Ora18]. Der Zugriff auf diese Daten wird typischerweise durch ein Database Management System (DBMS) bereitgestellt, um Benutzern die Interaktion mit einer oder mehreren Datenbanken zu ermöglichen. Die früh entwickelten traditionellen Datenbanken basierten auf der engen Beziehung von Daten, somit werden sie relationale Datenbanken[Cod70] genannt. Als eine andere Datenbankform existiert heute die nicht relationale Datenbank[HJ11], die auf einer losen Datenstrukturbeziehung basiert.
Relationale Datenbanken sind so organisiert, dass Daten in Zeilen und Spalten, die als Tupel und Attribute bekannt sind, gespeichert werden, damit sie von Benutzern leicht verstanden werden können. Diese Reihe von Zeilen und Spalten wird als Tabelle bezeichnet, eine Gruppe von Tabellen und die die Beziehungen dazwischen beschreibenden Verknüpfungen, die durch relationale Referenzen unter Verwendung von Primary und Foreign hergestellt werden, bildet die Datenbank. Daten werden in einer Datenbank über die Structured Query Language (SQL) abgerufen. Dabei handelt es sich um einen Ausführungscode, der bestimmte Bereiche der Datenbank definiert. Das relationale Modell kann also einfach als zweidimensionales Tabellenmodell verstanden werden, und eine relationale Datenbank ist eine Datensammlung, die aus zweidimensionalen Tabellen und den Beziehungen zwischen ihnen besteht.
Im Gegensatz dazu lassen sich in einer nicht relationalen Datenbank kein tabellarisches Schema von Zeilen und Spalten verwenden, sondern ein Speichermodell, das für die spezifischen Anforderungen des gespeicherten Datentyps optimiert ist und in dem die unstrukturierten oder halbstrukturierten Daten gespeichert werden als einfache Schlüssel-Wert-Paare, Spaltenfamilie, JSON-Dokumente oder Diagramme mit Kanten und Scheitelpunkten[HJ11]. Da für die Datenverwaltung kein SQL verwendet wird, werden solche Datenbanken auch als Not-only-SQL

(NoSQL) Datenbanken bezeichnet. Schlüsselwertspeicher ähneln assoziativen Karten oder Wörterbüchern. Hierbei identifiziert der Schlüssel den Wert eindeutig und wird zum Speichern und Abrufen des Werts verwendet. Die Spaltenfamilienspeicherung wird als zweidimensionaler Schlüsselwertspeicher betrachtet, in dem Daten spaltenorientiert gespeichert werden und ein Datensatz typischerweise aus mehreren Zeilen besteht, die jeweils durch einen eindeutigen Zeilenschlüssel adressiert werden[CDG+06]. Ein weiteres Derivat des Schlüsselwertspeicher-Datenmodells ist der Dokumentenspeicher, der einen Schlüssel verwendet, um ein Dokument im Format von wie zum Beispiel JSON eindeutig zu identifizieren, um Operationen mit Daten zu implementieren[Ser11]. Durch die Verwendung eines Graph-Datenmodells, das sich vollständig von Schlüsselwert-, Spaltenfamilien- und Dokumentspeichermodellen unterscheidet, können zusätzlich die Beziehungen zwischen stark miteinander verbundenen Datenknoten effizient in Graph-Datenbanken speichert werden[YKK17].
Relationale Datenbanken wie MySQL[Kof04], die ein festes Schema haben, verwenden SQL zum Verwalten der Daten und entsprechen den ACID-Spezifikationen (Atomicity, Consistency, Isolation, Durability)[Lan22], um sicherzustellen, dass die Daten im System zuverlässig und konsistent sind. Im Zeitalter von Big Data können sich herkömmliche relationale Datenbanken jedoch nicht an die durch groß angelegte digitale Interaktionen entstehenden Anforderungen wie beispiellose Datenskalierung, schnelle Reaktionsgeschwindigkeit und flexible sowie verteilte Datenspeicherung anpassen. Im Gegensatz dazu sind moderne NoSQL-Datenbanken, zum Beispiel MongoDB[BGBV16] und Elasticsearch[GT15], für Cloud-Dienste konzipiert und weisen eine hohe Leistung in Bezug auf Benutzerfreundlichkeit, Skalierbarkeit, Elastizität und Verfügbarkeitsmerkmale auf, sind jedoch häufig nicht ACID-konform[Lan22] und nicht SQL-konform. Einige Graphdatenbanken wie Neo4J sind zwar vollständig ACID-kompatibel, aber nicht so effizient und flexibel wie andere NoSQL-Datenbanken in nicht-graphen und relationalen Szenarien[GHTC13]. Daher wurde in den letzten 10 Jahren oft eine Mischform der beiden, nämlich NewSQL Datenbanken[GHTC13], diskutiert, die sich durch die Online-Skalierbarkeit von NoSQL-Datenbanken und Konformität ähnlich relationalen Datenmodellen auszeichnet, einschließlich die ACID-Konformität und der Verwendung von SQL als primäre Schnittstelle. Die Popularität einiger NewSQL-Datenbanken wie zum Beispiel VoltDB[SW13], CockroachDB[TSM+20] und NuoDB[KS17] hat innerhalb von 2013 bis 2022 insgesamt einen Aufwärtstrend gezeigt[sI22].

2.3. Python

Python wurde Anfang der 1990er-Jahre von Guido van Rossum am Centrum Wiskunde & Informatica als Nachfolger der Programmiersprache ABC entwickelt[Pra22]. Als eine Programmiersprache bietet *Python* stark strukturierte Konstrukte wie zum Beispiel Funktionen, Klassen, Module und Pakete, womit eine einfache und effiziente objektorientierte Programmierung ermöglicht wird. *Python*-Syntax und dynamische Typisierung sowie die Natur einer interpre-

tierten Sprache machen es zu einer Programmiersprache für Skripterstellung und schnelle Anwendungsentwicklung auf den meisten Plattformen[Swa13]. Mit der kontinuierlichen Aktualisierung der Version und dem Hinzufügen neuer Funktionen wurde *Python* zunehmend für die Entwicklung unabhängiger Großprojekte verwendet.
Weit von „Maschinensprache“ entfernt ist *Python* eine Sprache, die die Idee der Einfachheit repräsentiert, was es für Anfänger einfacher zu lesen und zu verstehen macht. Das Lesen eines guten *Python*-Programms fühlt sich an, als würde man die englische Programmbeschreibung im Klartext lesen. Diese Kernphilosophie ist in dem Dokument „The Zen of Python“[Pet10] zusammengefasst. Es wird durch diese pseudocode-artigen Wesen von *Python* ermöglicht, sich auf die Lösung von Problemen zu konzentrieren, anstatt die Sprache selbst herauszufinden.
Python ist so konzipiert, dass es erweiterbar ist. Nicht alle Features und Funktionen sind in den Sprachkern integriert[Ven03]. *Python* bietet reichhaltige APIs und Tools, sodass Programmierer problemlos *C*, *C++* und *Cython* verwenden können, um Erweiterungsmodule zu schreiben. Der *Python*-Compiler selbst lässt sich auch in andere Programme integrieren, die eine Skriptsprache benötigen. Daher wird *Python* auch als „Leimsprache“ verwendet, um in anderen Sprachen geschriebene Programme zu integrieren und zu kapseln. In Google Engine werden beispielweise die entsprechenden mit C++ geschriebenen Module mittels *Python* aufgerufen[Zah09].
Mit den Attributen „kostenlos“, „quelloffen“ und „leicht zu erlernen“ hat *Python* eine große und schnell wachsende Benutzergemeinschaft gebildet und viele Entwickler dazu gebracht, Bibliotheken bereitzustellen. Viele quelloffene Softwarepakete für wissenschaftliche Berechnungen bieten *Python*-Aufrufschnittstellen[Fou22] wie die Computer-Vision-Bibliothek *OpenCV*, die 3D-Visualisierungsbibliothek *VTK* und die medizinische Bildverarbeitungsbibliothek *ITK*. Gleichzeitig wird *Python* von einigen spezifischen Erweiterungsbibliotheken für wissenschaftliches Rechnen, wie *NumPy* und *Matplotlib*, mit schneller Array-Verarbeitung, numerischen Operationen bzw. Plotfunktionen versorgt[Fou22]. Der hochgradig interaktive Charakter und das wachsende Ökosystem machen *Python* zur beliebtesten Befehlsschnittstelle der Bibliotheken für maschinelles Lernen und andere numerische Algorithmen[Ras15].
Python als die Sprache mit dem größten Benutzerwachstum (9,6 %) in den letzten 5 Jahren hat sich zu einer der beliebtesten Programmiersprachen entwickelt. Die Häufigkeit der Suche nach *Python*-Tutorials bei Google hat seit 2006 linear zugenommen[Car22]. Aufgrund der Einfachheit, Lesbarkeit und Skalierbarkeit eignet sich die aus der *Python*-Sprache und ihren vielen Erweiterungsbibliotheken bestehende Entwicklungsumgebung optimal für technische und wissenschaftliche Forscher, um experimentelle Daten zu verarbeiten, Diagramme zu erstellen und sogar wissenschaftliche Computeranwendungen zu entwickeln. Einige Kurse, wie *Fundamentals of Programming and Computer Science* an der Carnegie Mellon Universität[Dep22a], werden bereits in *Python* unterrichtet.

2.4. Digitale Bildverarbeitung

Als Teilgebiet der digitalen Signalverarbeitung greift die Bildverarbeitung auf die diskrete mathematische Theorie und klassische Signalverarbeitungsmethoden zurück, um bildrepräsentative Signale zu verarbeiten, wie zum Beispiel Einzelbilder in Fotografien oder Videos. Der Zweck der frühen Bildverarbeitung bestand in Bildverbesserung, was bedeutet, die Qualität des Bildes zu verbessern. Das Jet Propulsion Laboratory (JPL) der Vereinigten Staaten wendete beispielsweise die Bildverarbeitungstechniken wie geometrische Korrektur, Graustufentransformation und Rauschunterdrückung auf Tausende von Mondfotos an, die 1964 von der Raumsonde „Ranger VII“ zurückgesendet wurden[Jaw05].
Mit der Entwicklung der Computergrafik entstehen und verbessern sich ständig die Technologien zur Bildverbesserung. Moderne Bildverbesserungsoperationen werden normalerweise in drei Arten unterteilt: Punktoperationen, lokale Operationen und globale Operationen. Punktoperationen, die durch Luminanzkorrektur und verschiedene Schwellwertoperationen dargestellt werden, betreffen einzelne Pixel des diskretisierten und quantisierten Bildes. Im Vergleich dazu werden bei lokalen Operationen das einzelne Pixel und seine Nachbarschaften betrachtet, um beispielsweise Glättung und Unschärfe zu erreichen oder um morphologische Operationen wie Dilatation und Erosion durchzuführen. Eine übliche Art der lokalen Operationen ist Faltungsfilter. Weder Punkt- noch Nachbarschaftsoperationen ändern die Größe oder Grundstruktur des Bildes. Dies wird durch geometrische globale Operationen wie Skalierung, Rotation oder Translation des Bildes erreicht. Eine weitere Darstellung der globalen Operationen ist die Fourier-Transformation, bei der das Bild im Frequenzraum transformiert wird, wobei die Anwendung linearer Filter weniger Arbeit bedeutet, um eine schnelle Erkennung von Merkmalen wie Linien und Kreisen oder eine Rauschunterdrückung zu ermöglichen.
Ein weiteres klassisches Anwendungsgebiet der Bildverarbeitung ist die Bildrekonstruktion. Diese Technologie wurde erstmals in radiologischen medizinischen Geräten eingesetzt, um mittels der Computertomographie (CT) dreidimensionale Gebilde mit Hilfe unendlicher zweidimensionaler Bilder zu rekonstruieren[Buz11]. Neben der tomographischen Rekonstruktion sind seit Ende des 20. Jahrhunderts viele neue 3D-Rekonstruktionsverfahren entstanden, wie beispielsweise die Shading-Rekonstruktion[IH81] und die Defokus-Rekonstruktion[FS05], um genauere Ergebnisse zu generieren.
Darüber hinaus ist ein anderes Hauptanwendungsgebiet der Bildverarbeitung die Objekterkennung und -Segmentation aufgrund von Features. Vor der Objekterkennung sollte zunächst eine Beschreibung der Objekteigenschaften erstellt werden. Die durch Snakes-Modell[KWT88] und Level-Set-Modell[OS88] dargestellte Kantenerkennung, Harris-Corner-Eckenerkennung[HS+88] sowie die Größen- und Farberkennung gelten häufig als die verwendete Verfahren zur Detektion der Features, die die Objekteigenschaften beschreiben sollen. Die weitere Etablierung der Objektmerkmalsdeskriptoren kann durch beispielsweise Scale-invariant Feature Transform (SIFT)[Low99] oder Histogram of oriented Gradients (HoG)[DT05] erreicht werden. Um unwichtige Merkmale aus einer Vielzahl von Merkmalen zu filtern, ist Principal Component

Analysis (PCA)[AW10] notwendig. Für komplexe Objekterkennungsaufgaben ist es zusätzlich notwendig, zuerst Merkmale mit einer oder mehrerer der oben genannten Methoden zu definieren und dann Technologien von klassischem maschinellem Lernen wie Support Vector Machines (SVM)[CV95] zur Klassifizierung zu verwenden. Außerdem lässt sich Deep Learning als Methode nutzen, um eine Objekterkennung durchzuführen, ohne bestimmte Merkmale vorher manuell zu definieren.
Mit der schnellen Entwicklung der Rechenleistung von Computern kann zusätzlich die Merkmalsextraktion von Bildsequenzen auf der Grundlage einer großen Anzahl von Matrixberechnungen auch für die Bewegungsschätzung in Echtzeit verwendet werden. Basierend auf dem oben erwähnten Erkennungsverfahren von Features kann das Geschwindigkeitsvektorfeld des sich bewegenden Objekts in der Bildfolge modelliert werden, um ein Bewegungsmodell des Objekts für die Bewegungsschätzung und -verfolgung zu erstellen[HS81].

2.5. Maschinelles Lernen und neuronale Netze

Künstliche Intelligenz (KI) ist die Simulation menschlicher Entscheidungsprozesse durch Maschinen bzw. Computersysteme[Edu20]. Als ein Zweig von KI ist maschinelles Lernen ein datenbasiertes Lernen, die Maschinen helfen kann, Gesetze aus vorhandenen komplexen Daten automatisch zu lernen, um zukünftige Verhaltensergebnisse und -trends vorherzusagen. Im Gegensatz zu regel-gesteuerten Algorithmen verwendet maschinelles Lernen datengesteuerte generische Algorithmen[MM97], was bedeutet, dass maschinelle Lernalgorithmen Logiken auf der Grundlage von Daten erstellen können, anstatt sich auf die vor der Implementierung festgelegten Regeln zu verlassen.
Maschinelles Lernen wird hauptsächlich in überwachtes Lernen, unüberwachtes Lernen und verstärkendes Lernen unterteilt. Überwachtes Lernen kann weiter unterteilt werden in Klassifikationsprobleme zur Vorhersage von Zugehörigkeit und Regressionsprobleme zur Vorhersage von Werten aus Eingaben. Beim überwachten Lernen passen Algorithmen, geleitet von Vorkenntnissen der Modellausgabewerte, automatisch die Modellparameter an, um die Abweichung zwischen erhaltenen und erwarteten Ergebnissen zu reduzieren. Ein klassischer überwachter Algorithmus ist der K-nächster-Nachbar-Algorithmus[FH89], wobei die k nächsten Trainingsbeispiele im Datensatz zur Überwachung verwendet werden, um Klassifizierung oder Regression über einen einzelnen Datenpunkt zu treffen. Darüber hinaus sind Algorithmen wie Decision Trees, Random Forest, Support Vector Machines und Naive Bayes ebenfalls häufig verwendete Methoden des überwachten Lernens[WKRQ+08].
Im Vergleich dazu sind die in einen unüberwachten Lernalgorithmus eingespeisten Daten unmarkiert, und der Algorithmus versucht die Daten selbst zu verstehen, indem er nach Merkmalen und Mustern sucht, um verschiedene Probleme bei der Mustererkennung zu lösen. Unüberwachtes Lernen wird hauptsächlich zum Lösen der Probleme wie Clustering, Datendimensionsreduktion, Anomalieerkennung, Bildsegmentierung und Rauschunterdrückung verwendet.

Für Clustering und Bildsegmentierung wird häufig der K-Means-Clustering Algorithmus erwähnt. Dies abzielt darauf, die gegebenen Beobachtungen in k Cluster zu unterteilen, in denen jede Beobachtung zu dem Cluster mit dem nächsten Mittelwert gehört[HW79]. Darüber hinaus kann Principal Component Analysis (PCA) jeden Datenpunkt auf die ersten Hauptkomponenten in einen niedriger-dimensionalen Raum projizieren, um die Dimensionalität der Daten auf ihre wichtigsten Merkmale zu reduzieren[Pea01]. Daher lässt sich dieses Verfahren häufig zur Reduzierung der Datendimensionalität und zur Reduzierung des Bildrauschens verwenden. Anomalieerkennungsalgorithmen werden verwendet, um Ausreißer in Datensätzen zu erkennen. Es kann mittels vieler Methoden durchgeführt werden, wie zum Beispiel Support Vector Machines[SPP10].
Verstärkendes Lernen besteht hauptsächlich aus Agent, Environment, State, Action und Reward[SB18]. Nachdem der Agent eine Aktion ausführt, geht die Umgebung in einen neuen Zustand über, für den ein Belohnungssignal (positive Belohnung oder negative Belohnung) gegeben wird. Anschließend führt der Agent neue Aktionen gemäß einer bestimmten Strategie und dem neuen Zustand sowie der Belohnung des Umgebungsfeedbacks aus. Durch verstärkendes Lernen weiß der Agent, in welchem Zustand er sich befindet und welche Maßnahmen er ergreifen sollte, um die maximale Belohnung zu erhalten. Darüber hinaus wird hier eine Wertfunktion benötigt, um langfristige Gewinne zu bewerten, anstatt sich nur auf unmittelbare Belohnungen zu verlassen. Verstärkendes Lernen versucht also, ein Gleichgewicht zwischen Exploration und Exploitation aufrechtzuerhalten, indem es Lerninformationen aus der Umgebung für Aktionen erhält und Modellparameter aktualisiert, um die Wertfunktion zu maximieren. Das Entscheidungsproblem des Agenten, die optimale Handlungssequenz zu finden, ist eigentlich ein Markov-Entscheidungsprozess (MDP)[OW12]. Um die Lösung dieses Prozesses zu finden, lassen sich Monte-Carlo-Methoden[Ham13] zur Verfügung stellen.
Es gibt vielfältige und eine stetig wachsende Anzahl von Methoden, die im Bereich des maschinellen Lernens eingesetzt werden. Zu den heute am häufigsten genannten Methoden gehört sogenannte Deep Learning. Bei klassischen maschinellen Lernmethoden sind die Eingaben für den Algorithmus manuell definierte Datenmerkmale wie beispielsweise die Farbe und Textur von Objekten in einem Bild. Stattdessen lernt Deep Learning, Objekte während des Trainings zu charakterisieren, spezifische Merkmale auszuwählen und damit zu berechnen, um die besten Ergebnisse zu erzielen. Diese Merkmale können intuitive Merkmale wie Farbe und Textur sein oder hochdimensionale Kombinationen von Merkmalen, die Menschen nicht wahrnehmen können. Jede Ebene des neuronalen Deep-Learning-Netzwerks werden mit einer überwachten Lerntechnik namens Backpropagation für Gewichtungsaktualisierungen trainiert[LBH15], um die Netzwerkausgabe näher an die Labels zu bringen, wobei die Gewichtungsaktualisierung auf einem Gradientenabstieg beruht, um den Wert der Verlustfunktion zu reduzieren. Um eine Überanpassung zu vermeiden, werden normalerweise einige Regularisierungsmethoden wie Dropout zur zufälligen Inaktivierung verwendet[SHK+14a].
Die verschiedenen neuronalen Netze im Deep Learning lassen sich im Allgemeinen in künstliche neuronale Netze (KNN), rekurrente neuronale Netze (RNN) und convolutionale neuro-

nale Netze (CNN) unterteilen. Künstliche neuronale Netze werden auch Multi-Layer Perceptron (MLP)[RL14] genannt. Sie besteht aus Eingabe- und Ausgabeschicht und versteckten Zwischenschichten, wobei jede Schicht außer der Eingabeschicht eine Reihe von mehreren Perzeptronen enthält, die auch als Neuronen bekannt sind. Normalerweise sind diese Schichten Fully Connected Layers (FC), daher wird MLP manchmal auch als Fully Connected Neural Network bezeichnet. Mit Ausnahme des Eingangsknotens ist jeder Knoten ein Neuron, das eine nicht lineare Aktivierungsfunktion verwendet, um zu vermeiden, dass es zu einem einschichtigen Netzwerk degeneriert, wodurch nicht-lineare trennbare Probleme behandelt werden. Bei der Verwendung eines KNN zur Lösung eines Bildklassifizierungsproblems besteht der erste Schritt darin, das zweidimensionale Bild in einen eindimensionalen Vektor umzuwandeln, wodurch das KNN nicht in der Lage ist, die Auftragsinformationen in den für die Verarbeitung erforderlichen Sequenzdaten zu erfassen. Die Anzahl der trainierbaren Parameter nimmt mit zunehmender Bildgröße stark zu. Anstatt also MLPs zu verwenden, lassen sich normalerweise Recurrent Connection Layers oder Convolutional Layers zur Merkmalsextraktion einsetzen, und dann werden eine oder mehrere FCs, die ähnlich wie die Ausgabeschicht vom MLP sind, für die Klassifizierung oder Regression genutzt. Die auf diese Weise aufgebauten Netze werden rekurrente neuronale Netze[MKB+10] oder konvolutionale neuronale Netze[LBBH98] genannt.
Rekurrente neuronale Netze fügen Rückkopplungen zwischen Neuronen innerhalb von gleicher Schicht oder unterschiedlichen Schichten des MLP hinzu, wodurch die Fähigkeit erlangt wird, zeitlich codierte Informationen in den Eingabedaten zu erfassen[SL19]. Der Zustand der verborgenen Schicht bezieht sich nicht nur auf die Eingabe im aktuellen Moment, sondern auch auf den Zustand des vorherigen Moments, sodass die Daten mit Zeitreiheninformationen wie Text- und Audiodaten verarbeitet werden kann. RNN teilen Parameter über verschiedene Zeitschritte hinweg, sodass weniger Parameter trainiert werden müssen, um die Rechenkosten zu reduzieren. Der Gradient dieser einfachen RNN-Struktur ändert sich jedoch exponentiell mit der Zeit, wodurch die Fähigkeit verloren geht, an den Informationen vor mehreren Zeitschritten zu erinnern, was zu einem Ausfall des Langzeitgedächtnisses und einem Problem aufgrund von Long-range Dependence (LRD) führt. Dieses Problem kann durch zum Beispiel die Einführung eines Long Short-Term Memory Network (LSTM)[HS97] oder einer Gated Recurrent Unit (GRU)[CVMBB14] bis zu einem gewissen Grad gelindert werden.
Als eine andere Variante von MLP lässt sich convolutionale neuronale Netze in verschiedenen Domänen einsetzen und sind besonders weit verbreitet in Bild- und Videoverarbeitungsprojekten. Im Gegensatz zu MLP muss CNN das Eingabebild nicht in eine Dimension erweitern, sondern verwendet abwechselnd Convolutional Layers und Pooling Layers als Hidden Layers, um Features und räumliche Elemente, die die Anordnung von Pixeln in einem Bild und ihre Beziehung zueinander widerspiegeln, zu erfassen[GBC16]. Somit lassen sich die Position von Objekten und ihre Beziehung zu anderen Objekten im Bild genau identifizieren. CNNs folgen auch dem Konzept der gemeinsamen Nutzung von Parametern, indem ein einzelner Filter auf verschiedene Teile der Eingabe zum Generieren der Merkmalkarte angewendet wird, wo-

durch räumliche Korrelationen in den Daten ermittelt werden, um die Anzahl der trainierbaren Parameter im Netzwerk zu reduzieren.
Im modernen Deep Learning werden mehrere Schichten gestapelt, um hoch-dimensionale Merkmale zu extrahieren. Beispielweise hat DenseNet-121 120 Faltungsschichten, 4 Average Pooling layers (AvgPool) und 1 FC[HLVDMW17]. Mit der Etablierung wichtiger Deep-Learning-Frameworks wie TensorFlow[ABC+16] und Pytorch[PGM+19], dem Erstellung großer Datensätze wie COCO[LMB+14] und der Entwicklung von Computerhardware hat Deep Learning große Erfolge bei der Erkennung, Segmentierung und Erkennung von Objekten in Bildern, wie z. B. Verkehrsschildererkennung[TS19], Segmentierung biologischer Bilder[LKB+17] und Straßen- und Fußgängererkennung[TNS20]. Darüber hinaus hat Deep Learning enorme Fortschritte in der Verarbeitung natürlicher Sprache[KLW19], der Fernerkundungserkennung[KM20] und anderen Bereichen ermöglicht. In Chess, Shogi und Go hat AlphaZero[DS18] mehrere Weltmeister geschlagen, und seine kreativen Gewinnzüge werden von Spielern aller Niveaus studiert. Im Bereich der Vorhersage der Proteinfaltung kann AlphaFold[JEP+21] bereits Vorhersagen für mehr als 200 Millionen Proteinstrukturen liefern. Es ist erwähnenswert, dass im modernen NLP-Bereich die traditionellen RNN-basierten Modelle schrittweise durch das sogenannte Transformer-Modell[VSP+17] ersetzt wurde, das auf dem Attentionmechanismus[Hu19] basiert. Die Leistung des auf der Grundlage dieser Architektur erweiterten Bert-Modells ist weitaus besser als die von RNN und seinen Varianten basierende Modelle[DCLT18]. Darüber hinaus werden Aufmerksamkeitsmechanismen auch im Bereich der Bildverarbeitung zur Verbesserung der Modellgenauigkeit eingeführt. Beispielsweise führt ein auf U-net[RFB15] basierendes neuronales Netzwerk namens RIC-Unet[ZXZL19] einen Kanalaufmerksamkeitsmechanismus ein, um die Leistung der Kernsegmentierung in Histologiebildern zu erhöhen.

3. Stand der Technik

In verschiedenen Dokumenten ist eine Tabelle ein Objekt, dessen strukturierte Organisation dem Extrahieren und Verstehen der Informationen dient. Da die Verarbeitungszeit einer Vielzahl elektronischer Dokumente im Zeitalter von Big Data durch die Tabellenerkennungstechnologie effektiv verkürzt werden kann, gilt die Tabellenextraktion als wichtiges Forschungsthema im Bereich des Dokumentenverständnisses. Es gibt jedoch verschiedene Variationen der Tabellenstruktur aufgrund komplexer Rahmen, Hintergrundpolsterung und Zellenverschmelzung, was dazu führt, dass keine Einheitslösung für alle Tabellenextraktionsaufgaben existiert. Tabellenextraktion besteht aus zwei Unteraufgaben: Tabellenerkennung und Texterkennung, wobei die Tabellenerkennungsaufgaben weiter in Tabellenbereichserkennung und Tabellenstrukturanalyse unterteilt werden können. Um verschiedene Methoden zur Tabellenextraktion zu vergleichen und auszuwählen, wird in diesem Kapitel ein intensiver Blick auf den Stand der Technik zur Tabellenerkennung und Texterkennung geworfen. Traditionelle regelbasierte Methoden und neuere Deep-Learning-Methoden, die auf neuronalen Netzwerken basieren, werden hierbei aufgezählt.

3.1. Tabellenerkennung

Unabhängig davon, ob es sich um ein traditionelles Verfahren oder ein auf Deep Learning basierendes Verfahren handelt, können Tabellenerkennungsverfahren nach der Reihenfolge von der Tabellenbereichserkennung und der Tabellenstrukturanalyse grob in Top-Down- oder Bottom-Up-Verfahren eingeteilt werden.
Beim Top-down-Verfahren wird zuerst den Tabellenbereich erfasst, danach lässt sich kontinuierlich der Tabellenbereich basierend auf a priori Regeln, wie zum Beispiel der leere Trennbereich[CK93, Han00] oder die Tabellenrahmen[GDPP05], zerschneiden, um jeden Zellenbereich zu erhalten und dabei die Tabellenstruktur zu analysieren. Im Gegensatz dazu basiert das Bottom-up-Verfahren darauf, dass am Anfang die Tabellenstruktur analysiert wird. Alle Textblöcke werden hierbei zunächst positioniert, dadurch lassen sich der Tabellen- und Textbereich entsprechend der räumlichen Anordnung der Textblöcke unterscheiden[MCDC06, Kie98].
Das traditionelle Tabellenerkennungsverfahren basiert auf einigen heuristischen Regeln, die sich hauptsächlich auf die Tabellenrahmen oder den leeren Trennbereich zwischen Tabellenzeilen und Textblöcken stützen, indem Bildverarbeitungsverfahren wie Erosion, Dilatation, Linien- und Schnittpunkteerkennung verwendet werden, um Zellen beziehungsweise die Text-

blöcke zu identifizieren.
Zur Erkennung der Tabellen mit vollständigen Linien, wie das Beispiel in Abb. 4.3 (a), stehen normalerweise den traditionellen Verfahren auf der Grundlage der Linienerkennung zur Verfügung. Yuan et al.[YLW+20] verwenden beispielsweise einen typischen traditionellen Top-Down-Ansatz für die Tabellenerkennung in Bildern basierend auf Bildverarbeitungsalgorithmen, die von der *OpenCV*-Bibliothek bereitgestellt wurden. Hierbei lassen sich zunächst die Umwandlung in Graustufen und die Binarisierung des Bilds durchführen und dann morphologische Operationen wie Erosion und Dilatation, um nacheinander horizontale und vertikale Linien zu extrahieren. Verglichen mit beispielsweise der Hough-Linienerkennung können diese morphologischen Operationen zuverlässiger horizontale und vertikale Linien extrahieren. Der Schnittpunkt wird durch Zusammenführen der extrahierten horizontalen und vertikalen Linien bestimmt, wodurch die gesamte Tabelle und jede einzelne Zelle lokalisiert und extrahiert werden. Die Ergebnisse werden dann unter Verwendung a priori bekannter Regeln wie etwa Zellenhöhe oder -breite weiter gefiltert. Obwohl diese Methode Tabellen in Bildern relativ vollständig identifizieren kann und leicht zur Erkennung der sich über Seiten erstreckenden Tabellen erweitert werden kann, lassen sich auch einige Probleme feststellen: Beispielweise können die Illustrationen, die ähnlich groß wie eine Zelle sind, die Erkennung stören. Darüber hinaus könnte dieser einfache Algorithmus keine Tabellen erkennen, die nicht-vollständigen Linien haben (in Abb. 4.3 (b), (c) und (d)) oder die sich in schiefen oder verrauschten Bildern befinden.
Ein mögliches Verfahren zur Erkennung von Tabellen mit nicht-vollständigen Rahmen könnte durch die zusätzliche Erkennung von weißen Lücken erfolgen. Anders als bei anderen Ansätzen ist die Vorgehensweise von Chandran und Kasturi[CK93] nicht auf das Vorhandensein von Linien zur Kennzeichnung der Elementblöcke angewiesen. Weiße Streams werden sowohl in horizontaler als auch in vertikaler Richtung als Ersatz für fehlende Demarkationslinien erkannt. Das Strukturinterpretationsverfahren verwendet die extrahierten Abgrenzungsinformationen, um jeden der Elementblöcke in der Tabelle zu identifizieren. Anschließend werden die Beziehungen dieser Elementblöcke verwendet, um die Struktur der tabellarischen Daten zu erkennen.
Zusätzlich besteht eine übliche Bottom-up-Lösung zur Erkennung der Tabellen mit unvollständigen Linien darin, zuerst Textblöcke zu lokalisieren, um Tabellenbereiche zu bestimmen und dann eine Strukturanalyse gemäß der Regelmäßigkeit der Positionsverteilung von Textblöcken durchzuführen. Kieninger et al.[Kie98] schlagen demgemäß das T-Recs-System vor, das auf der Idee des Textblock-Clustering für die Tabellenerkennung basiert. Beginnend mit einem beliebigen Wort als Blockstartwert erweitert hierbei der Algorithmus diesen Block rekursiv auf alle Wörter, die mit ihren vertikalen Nachbarn verschachtelt sind. Da selbst die kleinste Lücke von Tabellenspalten verhindert, dass sich ihre Wörter verflechten, ist diese anfängliche Segmentierung in der Lage, diese Spalten zu identifizieren und zu isolieren. Es gibt auch Einschränkungen bei diesem Ansatz. Beispielweise führen die zellübergreifenden Überschriften dazu, dass alle Textblöcke unter der Überschrift zusammengeführt werden. Zusätz-

lich werden isolierte Textzeilen in viele einzelne Blöcke aufgeteilt. Daher besteht der größte Teil der späteren Arbeit des T-Recs-Systems darin, Nachbearbeitungsregeln für diese Einschränkungen festzulegen, um eine bessere Allgemeingültigkeit zu erhalten. Darüber hinaus schlägt Handley[Han00] ein Tabellenanalysesystem vor, das Formatinformationen wie Tabellenteiler aus Tabellenbildern rekonstruiert, indem Pixelwert-Histogrammstatistiken in vertikaler und horizontaler Richtung durchgeführt werden, unabhängig davon, ob die Teiler explizit oder implizit im Originalbild enthalten sind. Diese Methode verwendet eine Reihe von sorgfältig ausgearbeiteten Regeln, um Zellen und ihre Wechselbeziehungen zu finden, selbst wenn keine expliziten Trennzeichen sichtbar sind.
Herkömmliche Methoden können genaue Ergebnisse in Tabellen mit bekannter Struktur identifizieren, aber unabhängig davon, wie komplexe Regeln zur Analyse der Tabelle verwendet werden, gibt es keine einheitliche Methode, um mit allen Varianten umzugehen. Beim Umgang mit Bildern in schlechter Qualität ist außerdem die erwartete Leistung schwer zu erhalten.
In den letzten zehn Jahren hat sich die Technologie der künstlichen Intelligenz (KI) rasant entwickelt. Es werden viele auf maschinellem Lernen basierende Verfahren zum universellen Erkennen verschiedener Tabellen vorgeschlagen. Somit schlägt Farrukh et al.[FFRC+17] eine Bottom-up-Argumentationsmethode vor, die Random Forests, Markov-Logik-Netze und K-Means-Clustering kombiniert, um zellbezogene Hierarchien zu finden. In der Arbeit von Rashid et al.[RAA+17] wird ein Automatic Tunable Multilayer Perceptron (AutoMLP), das die Ideen genetischer Algorithmen und stochastischer Optimierung kombiniert, verwendet, um basieren auf einige künstliche Merkmale jeden Textblock als „gehört zur Tabelle“ oder „gehört nicht zur Tabelle“ zu klassifizieren. Die Ergebnisse werden durch eine kontextbezogene Nachbearbeitung weiter verbessert.
Zusätzlich lassen sich die ausgereiften neuronalen Netze von Deep Learning wie CNN, RNN und Graph Neural Networks (GNN)[KW16] auf Tabellenerkennungsaufgaben mit guten Ergebnissen einsetzen. Zu den gängigen Methoden der auf tiefen neuronalen Netze basierenden Tabellenerkennungsverfahren gehören semantische Segmentierung, Objekterkennung oder Graph-Neural. Die berühmten Deep-Network-Modelle wie VGG19 von Simonyan et al.[SZ14], Faster R-CNN von Ren et al.[RHGS15], FCN von Long et al.[LSD15], Mask R-CNN von He et al.[HGDG17] und die YOLO Serien von Redmon et al.[RDGF16], funktionieren effektiv bei Objekterkennungs- und semantischen Segmentierungsaufgaben gemäß ihren Experimenten an mehreren Datensätzen. Die semantische Segmentierung bezieht sich auf die Klassifizierung jedes Pixels im Bild, d. h. das Zuordnen eines Etiketts zu jedem Pixel. Die meisten Modelle, die üblicherweise bei semantischen Segmentierungsaufgaben verwendet werden, basieren auf der Fully Convolutional Networks (FCN)[LSD15], die die FCs am Ende von traditionellem CNN durch Faltungsschicht ersetzen, sodass die Ausgabe des Netzwerks gleich wie eine Heatmap ist. Gleichzeitig wird die Bildgröße durch Upsampling wiederhergestellt, um die durch Faltung und Pooling verursachte Reduzierung der Bildgröße zu lösen. Abhängig von der Aufgabe können semantische Segmentierungsmodelle trainiert werden, um nur tabellarische Masken oder eine größere Vielfalt von Masken auszugeben. Siddiqui et al.[SKDA19] verwenden ein FCN

Modell mit einer Zwei-Kopf-Architektur, indem sie die anfänglichen Feature-Maps aus dem Encoder-Decoder-Modell gemeinsam nutzten, um Vorhersagen für Zeilen und Spalten gleichzeitig unter Verwendung der letzten zwei Faltungsschichten zu generieren, wobei durch die Schnittmenge von Zellspalten und Zellzeilen jede Zelle sowie die endgültige Tabellenstruktur bestimmt wird. Die CV-Anwendungsforschungsgruppe des Tencent QQ R&D Center[Yon19] schlägt eine detailliertere Klassifizierungsmethode vor, entwirft ein semantisches Segmentierungsmodell mit U-net[RFB15] als Backbone und unterteilte Pixel weiter in horizontale Linien, vertikale Linien, unsichtbare horizontale Linien und unsichtbare vertikale Linien, um Zelleninformationen genauer zu erhalten.
Darüber hinaus kann die Positionierung des Tabellenbereichs auch durch das Objekterkennungsmodell vervollständigt werden. Das von Schreiber et al.[SAW+17] vorgeschlagene DeepDeSRT Modell verwendet zuerst Faster R-CNN zur Erkennung der Tabellenregionen. Danach werden die Tabellenzeilen bzw. -spalten mittels FCN segmentiert, somit lässt sich schließlich die Tabellenstruktur festlegen. Das LGPMA-Schema von Qiao et al.[QLC+21] unterteilt die Tabellenerkennung in drei Teile: Textzeilenerkennung, Texterkennung und Tabellenstrukturerkennung. Hierbei steht ein einzeiliger Textdetektor zum Textzeilenerkennen und einen aufmerksamkeitsbasierten Erkenner zum Texterkennen zur Verfügung, um Textinformationen in Tabellenbildern zu erhalten. Der Teil zur Tabellenstrukturanalyse basiert auf Mask-RCNN mit zwei semantischen Segmentierungsköpfen zum Lernen lokaler Ausrichtungsgrenzen bzw. globaler Ausrichtungsgrenzen. Um das Problem der ungenauen Positionierung von Tabellengrenzen anzustreben, schlägt Sun et al.[SZH19] ein schnelles Tabellenerkennungsverfahren vor, das auf dem Faster R-CNN Modell in Kombination mit Eckenpositionierung basiert. Neben der Erkennung der Tabellenobjekte werden zusätzlich die Eckpunkte der Tabellen erkannt, sodass die Tabellengrenzen durch entsprechende Eckgruppen angepasst und optimiert werden.
In den letzten Jahren haben immer mehr Arbeiten die Graphentheorie auf das Gebiet des Deep Learning zur Analyse der Tabellenstruktur eingesetzt. Die Textblöcke werden hierbei als die Knoten des Graphen angesehen, und die Beziehung zwischen den Textblöcken als die Kanten. Die Knoten und Kanten des Graphen werden mit bestimmten Merkmalen beschrieben, die entweder manuell definiert oder automatisch durch maschinelles Lernen oder Deep Learning generiert werden könnte, somit lassen sich mittels GNN die interaktive Fusion von Features durchführen, um das repräsentative Feature jedes Knotens zu generieren. Ein Klassifikator wie MLP wird dann vewendet, um die Beziehung zwischen zwei Knoten und die Kategorie jedes Knotens gemäß den verschmolzenen Merkmalen vorherzusagen. Darauf aufbauend wird dann eine Adjazenzmatrix generiert, um die Tabellenstruktur wiederherzustellen. Typische Arbeiten sind zum Beispiel GraphTSR[CHX+19] und Graph-based Financial Table Extraction (GFTE)[LHY+21].
Es ist erwähnenswert, dass traditionelle Methoden und auf maschinelles Lernen sowie Deep Learning basierende Methoden nicht antagonistisch sind, sondern sich ergänzen können. Während Huang et al.[HYL+19] das bekannte YOLOv3 Modell auf die Tabellenerkennungsaufgabe einsetzt, wird K-Means-Clustering dabei gleichzeitig verwendet, um die Anker der

Objekterkennung zu optimieren, wodurch es für das Modell einfacher wird, die genaue Position der Tabelle zu finden. Beim von Paliwal et al.[PVR+19] erstellten TableNet Modell lässt sich VGG19[SZ14] als Backbone zur Merkmalenextraktion und zwei semantische Segmentierungsköpfe zur Tabellenbereichssegmentierung und Tabellenspaltensegmentierung anwenden. Im Vergleich dazu war hierbei die Tabellenzeilensegmentierung unter Verwendung a priori bekannter Regeln zu erreichen. In dem von Prasad et al.[PGK+20] vorgeschlagenen CascadeTabNet werden Eingangsbilder zuerst in Tabellen mit vollständigen oder unvollständigen Linien klassifiziert. Die Tabellen mit unvollständigen Linien werden mittels des zweiten Teils des Modells zellsemantische segmentiert, während traditionelle Methoden zur Identifizierung der Zellen in Tabellen mit vollständigen Linien verwendet wurden.
Insgesamt ist es nötig, den Tabellenbereich zu bestimmen und die Tabellenstruktur zu analysieren. Die Verwendung von Deep Learning zum Extrahieren der Merkmale für die Tabellenerkennung scheint ein robusterer Ansatz zu sein, aber es ist auch möglich, traditionelle regelgesteuerte Methoden zu verwenden, um das Parsing weiter zu verbessern.

3.2. Texterkennung

Im Allgemeinen sollte der Text in den Zellen erkannt werden, nachdem der Tabellenbereich auf irgendeine Weise extrahiert und die Struktur analysiert wurde. Diese Technik zum Erkennen von Zeichen in Bildern wird als Optical Character Recognition (OCR) bezeichnet.
Die traditionellen Texterkennungen erfolgen meist durch ausgeführte OCR einzelner Buchstaben. Der Standardverarbeitungsablauf umfasst Bildvorverarbeitung, Textzeilenerkennung, Einzelzeichensegmentierung, Einzelzeichenerkennung und Nachbearbeitung. Dabei sollten die Buchstaben zunächst korrekt segmentiert werden und anschließend die Einzelzeichenerkennung basierend auf Musterabgleich oder Merkmalsextraktion durchgeführt werden[OCR09].
Im Musterabgleich wird eine Reihe sorgfältig entworfener Vorlagen verwendet, um eine eingeschränkte Zeichenerkennung zu ermöglichen. Dieser Ansatz funktioniert gut für Standarddokumente mit bestimmten Schriftarten, die Leistung ist jedoch eingeschränkt, wenn die Eingabezeichen zu weit von der gespeicherten Vorlage abweichen. Bei der Merkmalsextraktion werden die eingegebenen Zeichen durch eine Reihe von manuell definierten Merkmalen wie Stroke Width Transforms[EOW10] oder HOG-Features[WBB11, WB10] beschrieben, danach werden sie durch einige maschinelle Lernverfahren wie Sliding Window Klassifikatoren[MAJ12], Ensembles von SVMs[YBSL14]oder k-nächste Nachbarn Klassifikatoren mit HoG-Features[WB10] klassifiziert oder geclustert. Obwohl dieser Ansatz die Robustheit im Vergleich zum Vorlagenabgleich verbessert, stützt er sich zur Erkennung nur auf manuell festgelegte Merkmale auf niedriger und mittlerer Ebene, was in extremen Anwendungsszenarien wie der handschriftlichen Texterkennung oft zu einer langwierigen Nachbearbeitung in der Hoffnung auf die Verbesserung der Leistung führt[THH+16].
Eine logische Erweiterung der traditionellen OCR-Technologie ist die Intelligent Character

Recognition (ICR)[Moh20], die die Erkennung von handschriftlichen Zeichen in verschiedenen Schriftarten und Stilen ermöglicht. Nach dem Erfassen des Bildinhalts bestimmt es anhand des semantischen Kontexts, ob die identifizierten Buchstaben sinnvoll sind, um verwirrende Zeichen wie „8“ und „B“ zu korrigieren. Eine weitere Erweiterung der OCR-Technologie ist Intelligent Word Recognition (IWR), bei der statt einzelner Zeichen ganze Wörter erkannt werden, somit ermöglicht die bessere Erkennung von unstrukturierter oder freier Handschrift (z. B. Schreibschrift)[Bis15].

Mit der rasanten Entwicklung von Deep Learning sind allmählich die auf Deep Learning basierenden Texterkennungen gereift, die flexibel mit unterschiedlichen Szenarien umgehen und dabei Buchstaben statt einzeln in ganzen Zeilen oder Seiten zusammen verarbeiten können. Gegenwärtig umfassen die auf Deep Learning basierenden Texterkennungen hauptsächlich zwei Verfahren: Eine ist in zwei Phasen der Textbereichsdetektion und Inhaltserkennung unterteilt[ZYW+17, LPH+20], während die andere darin besteht, die Textbereichsdetektion und Inhaltserkennung gleichzeitig durch ein End-to-End-Modell abzuschließen[LLY+18, LSB18].

Die Textbereichsdetektion wird verwendet, um Textbereiche in Bildern zu lokalisieren. Die Genauigkeit der Lokalisierung wirkt sich direkt auf die Ergebnisse der nachfolgenden Erkennung aus. Es gibt bereits viele Verfahren zur Textbereichsdetektion: Connectionist Text Proposal Network (CTPN)[THH+16] ist ein Texterkennungsalgorithmus, der von Tian et al. vorgeschlagen wurde. Der Algorithmus ist gegenüber Faster-RCNN[RHGS15] verbessert und verwendet VGG16[SZ14] als Backbone, um Merkmale zu extrahieren. Dabei werden CNN und bidirektionales LSTM Deep Network zur effektiven Lokalisierung von horizontal verteiltem Text kombiniert. Das von Zhou et al.[ZYW+17] vorgeschlagene Efficient and Accurate Scene Text Modell (EAST) erreicht eine Optimierung in Genauigkeit und Effizienz, indem viele unnötige Zwischenschritte verworfen werden. Dieses Modell entwirft ein FCN basierend auf der Idee von U-net[RFB15], um Wörter oder Textzeilen in jeder Richtung und viereckigen Form im vollständigen Bild direkt vorherzusagen. Der hierbei vorgeschlagene Algorithmus kann die Leistung erheblich verbessern und gleichzeitig schneller laufen. Anders als die anchor-basierten[Mat22] Verfahren, die durch oben genannten Modelle CTPN und EAST repräsentiert werden, verwenden Wang et al.[WXL+19] ein segmentierungsbasiertes Verfahren und schlägt ein neuartiges Progressive Scale Expansion Network (PSENet) vor. Es kann Textinstanzen mit beliebigen Formen wie zum Beispiel gekrümmten Text genau erkennen und gleichzeitig falsche Erkennungen vermeiden, die zwei Instanzen abdecken, die durch zwei nahe beieinander liegende Textinstanzen verursacht werden.

Die Inhaltserkennung ist der Prozess der Umwandlung von Textbereichen in computerlesbare und bearbeitbare Symbole auf der Grundlage der Texterkennung. Die tiefen neuronalen Netze verwenden zunächst ein faltendes Netz zur Lokalisierung jeder Zeichen, und dann eine Reihe von Softmax-Klassifikatoren[Woo22] zur Klassifizierung, sowie zusätzlich ein Softmax-Klassifikator zur Vorhersage der Textlänge, sodass es kurze und einfache Textsequenzen verarbeiten kann, wie z. B. die Erkennung von Hausnummern[GBI+13]. Ein breiter anwendbares tiefes Texterkennungsverfahren ist das Decodierungssequenzverfahren. Es kann verwen-

det werden, um eine große Anzahl ganzer Wörter ohne Lokalisierung einzelner Buchstaben vorherzusagen. Die Hauptschwierigkeit bei der Decodierung von OCR-Texterkennung ohne Einzelzeichenpositionierung liegt darin, wie die Eingabe und Ausgabe ausgerichtet werden, was bedeutet, dass dasselbe Eingabezeichen möglicherweise mehrmals decodiert und wiederholt werden kann. Daher ist es notwendig, einen Decodierungsalgorithmus für die Texterkennung zu entwerfen, um das Ausrichtungsproblem von Eingabe und Ausgabe zu lösen. Häufig verwendete Decodierungsmethoden zum Auffinden genauer Zuordnungsbeziehungen umfassen die Connectionist Temporal Classification (CTC)[GFGS06] und Aufmerksamkeitsmechanismen. Eine typische Arbeit ist eine von Shi et al.[SBY16] vorgeschlagene neuartige neuronale Netzwerkarchitektur, die Merkmalsextraktion, Sequenzmodellierung und Transkription in ein einheitliches Framework integriert. Zuerst wird dabei CNN zur Extraktion der Bildfaltungsmerkmale verwendet. Um weitere Sequenzmerkmale zu extrahieren, wird dann LSTM hinzugefügt. Es führt schließlich CTC zum Lösen des Problems ein, so dass die Eingaben und Ausgaben während des Trainings nicht ausgerichtet werden können. Darüber hinaus schlagen Gao et al.[GCW+19] ein vollständig konvolutionelles Netz mit gestapelten Faltungsschichten vor, das effizienter als Bidirectional Recurrent Neural Networks (BLSTM) ist, um die verkettete Struktur von RNNs zu ersetzen. Dadurch wird langfristige Abhängigkeiten zwischen Szenentext und Bildelementen effektiver erfasst.
Es ist erwähnenswert, dass für den gekrümmten unregelmäßigen Text, wenn der gesamte Textbereich direkt in das Inhaltserkennungsmodell eingespeist wird, eine große Anzahl ungültiger Bereiche zu einem schlechten Erkennungseffekt führt. Eine Lösung namens RARE[SWL+16] besteht darin, das Bild zuerst in das Spatial Transformer Network (STN)[JSZ+15] einzuspeisen, um sich ändernde Parameter für die geometrische Korrektur zu lernen, bevor es zur Inhaltserkennung in das Sequenzerkennungsnetz eingespeist wird. Zusätzlich kann auch das Erkennungsproblem von gekrümmtem Text basierend auf der Klassifizierungsidee gelöst werden. Liao et al.[LZW+19] schlagen Character Attention Fully Convolutional Network (CA-FCN) vor, wobei einen semantischen Segmentierungszweig im Netzwerk verwendet wird, um die relative Position jedes Zeichens zu erhalten und dann die Zeichen zu klassifizieren und auszugeben. Obwohl das zweistufige Verfahren mit Textbereichsdetektion und Inhaltserkennung die Erkennung von Szenentext realisieren kann, erfordert es immer noch viel manuelles Wissen, wenn die Ergebnisse der beiden Schritte kombiniert werden, und erhöht es den Zeitaufwand, während die End-to-End Texterkennung gleichzeitig Textbereichsdetektion und Inhaltserkennung abschließen kann, wodurch die Echtzeitleistung der Texterkennung erheblich verbessert wird. In dem von Liu et al.[LLY+18] vorgeschlagenen FOTS-Modell wird zuerst ein gemeinsames Faltungsmodul zum Extrahieren der Merkmalskarten eingesetzt. Anschließend werden zwei Zweige eingerichtet, einer basierend auf FCN für die Vorsage von orientiertem Bounding Box und der andere für die Inhaltserkennung. Der Inhaltserkennungszweig verwendet eine Operation namens ROIRotate, um geeignete Merkmale aus der Merkmalskarte gemäß dem vorhergesagten Bounding Box mit Richtung zu extrahieren und dieses Merkmal in den nachfolgenden RNN- und CTC-Decoder zur Textinhaltserkennung einzugeben. Da Detektion und

Erkennung dasselbe Merkmalsextraktionsmodul teilen, kann das Netzwerk Text gleichzeitig in Echtzeit detektieren und erkennen, ohne dass die Rechenkosten erhöht werden. Ein weiteres interessantes Modell ist das STN-OCR-Modell[BYM17], das halbüberwacht trainiert wird. STN-OCR verwendet ein Spatial Transformer Network (STN) in der Lokalisierungsphase, um n Regionen aus dem ursprünglichen Eingangsbild zu generieren, wobei n der Anzahl der Textblockbereiche im Eingabebild entspricht. Die Regionen werden mittels einer Reihe von affinen Transformationen generiert, was das nachfolgende Erkennungsnetzwerk zum Umgehen mit gekrümmtem Text vermöglicht. In der Texterkennungsphase werden die Inhalte in den n Regionen separat identifiziert, was von einem auf der Residual Neural Network (ResNet)[HZRS16] basierenden CNN durchgeführt wird.

4. Implementierung

Wie im dritten Kapitel beschrieben, kann die Tabellenextraktion in Tabellenerkennung und Texterkennung aufgeteilt werden, wobei die Tabellenerkennung weiter in Tabellenbereichserkennung und Tabellenstrukturanalyse unterteilt werden kann. In diesem Kapitel werden zuerst die Vorverarbeitung dargestellt, die dem Entfernen von Neigung und Rauschen aus dem Bild dient. Danach sind verschiedene Algorithmen für die Tabellenbereichserkennung, Tabellenstrukturanalyse implementiert. Anschließend ist die Beschreibung von Texterkennungsweise. Am Ende wird in der Nacharbeitung die mittels OCR identifizierten Informationen in den Tabellen auf der Grundlage einer Reihe sorgfältig entwickelter Regeln verarbeitet, somit diese Informationen vollständig in die Datenbank eingeschrieben werden können.
Um Mehrdeutigkeiten bei der Beschreibung des Tabellenbereichs und des Zellenbereichs zu vermeiden, wird das in dieser Arbeit auf Bild verwendete Koordinatensystem wie in Abbildung 4.1 definiert. Der Koordinatenursprung befindet sich in der oberen linken Ecke des Bildes. Daher wird eine hellgraue Zelle mit einer Breite von 800 Pixeln und einer Höhe von 400 Pixeln, dessen obere linke Ecke mit dem Koordinatenursprung zusammenfällt, als ein Array $[0,0,800,400]$ in Form von $[x,y,w,h]$ dargestellt, wobei x die Abszisse der oberen linken Ecke der Zelle, y die Ordinate, w die Breite und h die Höhe darstellt.

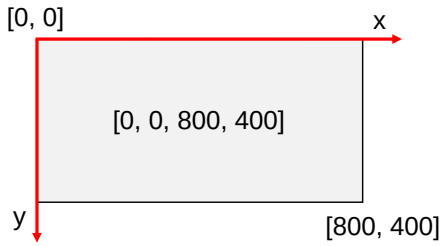


Abbildung 4.1. Koordinatensystem auf Bild.

4.1. Vorverarbeitung

Als Anfangsschritt der meisten Dokumentenanalysen dient die Vorverarbeitung hauptsächlich dazu, das Rauschen im Bild zu unterdrücken, wodurch die durch Rauschen verursachte Effizienzminderung im nachfolgenden Analyseprozess verringert wird. Außerdem erfordern die gescannte Dokumente oft eine Neigungskorrektur, um sicherzustellen, dass die Linien innerhalb der Tabelle horizontal oder vertikal sind. Schließlich haben Bilder mit stark schiefem Text

Tabelle 4.1. Funktionen zur Binarisierung und Rauschunterdrückung in *OpenCV*.

Algorithmus	Funktion in *Python*[1]
Otsus Methode	*cv2.threshold()*
Adaptive Binarisierung	*cv2.adaptiveThreshold()*
Medianfilter	*cv2.medianBlur()*
Gaußscher Filter	*cv2.GaussianBlur()*
Boxfilter	*cv2.boxFilter()*

[1] Detaillierte Parameter der Funktionen sind in den referenzierten offiziellen Dokumentationen von *OpenCV* zu finden.

nach der Neigungskorrektur tendenziell zu breite weiße Ränder, die entfernt werden müssen, um wichtige Teile hervorzuheben. Der entsprechende Workflow der Vorverarbeitung wird in Abbildung 4.2 visualisiert.

1) **Binarisierung und Rauschunterdrückung.** Die üblicherweise verwendeten Algorithmen zur Rauschunterdrückung und Linienextraktion werden an Binärbildern durchgeführt, daher sollte das Eingangsbild zuerst in ein Graustufenbild und dann weiter in ein Binärbild umgewandelt werden. Nach der Binarisierung kann es zu Geräusch in Form von auffälligen schwarzen Punkten kommen. Diese Geräusche können durch Rauschunterdrückung bis zu einem gewissen Grad entfernt werden. In der Tabelle 4.1 werden die entsprechenden Funktionen in *OpenCV* zur Binarisierung wie Otsus Methode und Adaptive Binarisierung[Vis22b] und Rauschunterdrückung wie Medianfilter, Gaußscher Filter und Boxfilter[Vis22a] aufgelistet.

2) **Vorläufige Neigungskorrektur mittels Liniendetektion.** Die allgemeine Neigungskorrektur basiert auf der Erkennung gerader Linien im Bild. Alle geraden Linien, die länger als eine bestimmte Länge sind, werden erkannt. Der Neigungswinkel wird somit entsprechend berechnet, um die Neigung durch Generieren der Rotationsmatrix des Bildes zu korrigieren. Zur häufig verwendeten Algorithmen zur Erkennung gerader Linien gehören Hough-Transform[DH72], Line Segment Detector (LSD)[VGJMR12] und Fast Line Detector (FLD)[Vis22c]. In Abbildung 4.2 (a) befindet sich ein Beispiel der Neigungskorrektur basierend auf der leistungsstärksten FLD-Linienerkennung dargestellt.

3) **Ergänzende Neigungskorrektur mittels rechteckigen Rahmens.** Die Rotationskorrekturmatrizen von Bildern, auf den sich keine gerade Linie oder nur Linien kurzer als ein gegebener Längenschwellenwert befinden, können nicht auf der Grundlage der Liniendetektion erhalten werden, was die Neigungskorrektur unmöglich macht. In diesem Fall ist es ein notwendiges zusätzliches Mittel, den Textbereich mittels der Funktion

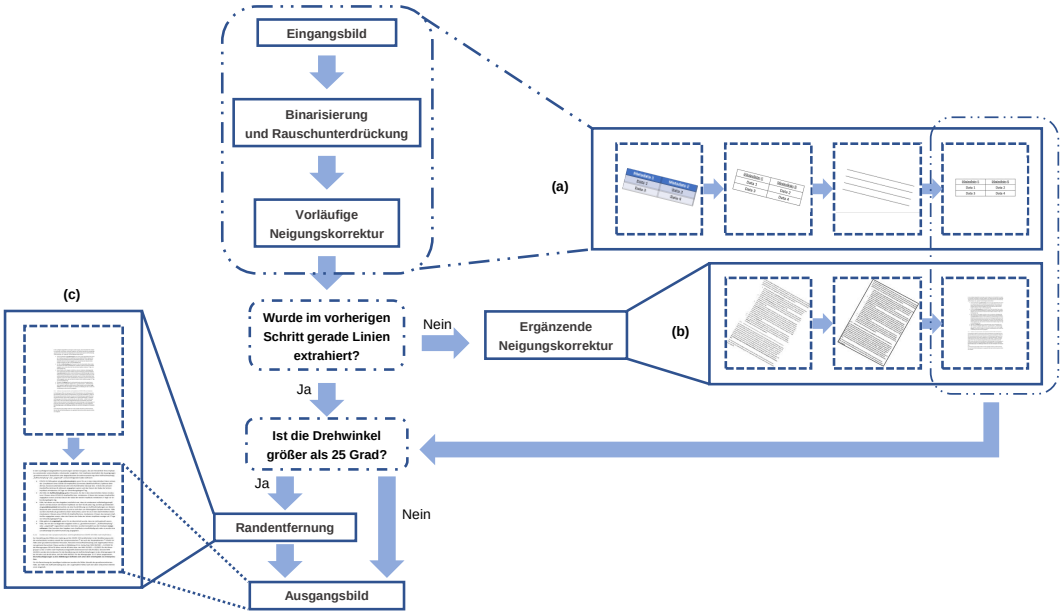


Abbildung 4.2. Workflow von Vorverarbeitung.

cv2.minAreaRect() mit einem minimal neigbaren rechteckigen Rahmen einzurahmen, um einen Neigungswinkel zu erhalten. Ein Beispielbild ohne Linien ist in Abbildung 4.2 (b) gezeigt. Dabei sind es auch zu sehen, wie die Neigungskorrektur in diesem Fall unter Verwendung eines minimal neigbaren rechteckigen Rahmens durchgeführt werden kann.

4) **Randentfernung.** Bei Bildern nach starker Neigungskorrektur sind die Textbereiche oft mit breiten weißen Rändern versehen, die dazu führen, dass ein Teil hochfrequenter Informationen innerhalb des Textbereichs durch den Auflösungsabfall bei der anschließenden Bildgrößennormierung verloren geht. Daher wird in diesem Schritt der weiße Rand entfernt, der größer als die Breitenschwelle ist, um die aussagekräftigeren Bereiche hervorzuheben, die Text und Tabellen enthalten. Der Vorgang des Entfernens der Umrandung ist in Abbildung 4.2 (c) dargestellt.

4.2. Tabellenbereichserkennung

In diesem Abschnitt werden die Implementierungsprozesse von drei Erkennungsverfahren des Tabellenbereichs im Detail beschrieben. Das erste ist ein traditionelles, regelbasiertes Verfahren, das die Tabellenbereichserkennung durch Erfassen von Tabellenrahmenlinien realisiert. Dann gibt es ein weiteres regelbasiertes Verfahren, bei dem die Erkennung des Tabellenbereichs auf dem offensichtlichen Verteilungsunterschied zwischen den Textblöcken im Text-

	column1	column2	column3
row1	value1	value2	value3
row2	value4	value5	value6
row3	value7	value8	value9

(a) Beispiel: Tabelle mit vollständigen Linien

	column1	column2	column3
row1	value1	value2	value3
row2	value4	value5	value6
row3	value7	value8	value9

(b) Beispiel: Tabelle mit vollständigem, äußerem Rahmen

	column1	column2	column3
row1	value1	value2	value3
row2	value4	value5	value6
row3	value7	value8	value9

(c) Beispiel: Tabelle mit wenigen Linien

	column1	column2	column3
row1	value1	value2	value3
row2	value4	value5	value6
row3	value7	value8	value9

(d) Beispiel: Tabelle mit rahmenlosen Zellen

Abbildung 4.3. Beispiele der Tabellenvarianten.

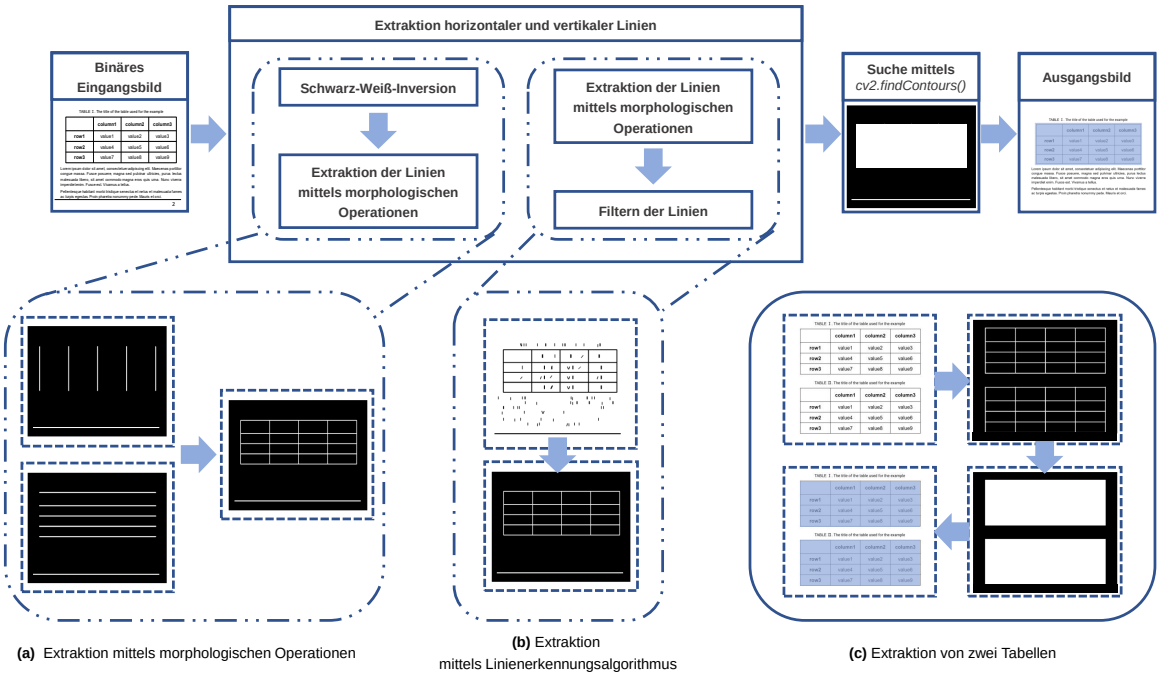
bereich und dem Tabellenbereich beruht. Das heißt, dass die Textblöcke in der Tabelle eine offensichtliche, spalten-förmige Verteilung zeigen. Abschließend wird ein Verfahren zur Erkennung von Tabellenregionen basierend auf tiefen, neuronalen Netze vorgestellt, wobei die Modellstruktur und die verwendeten Trainingsparameter detailliert erläutert werden.

4.2.1. Linienerkennungsbasierte Tabellenbereichserkennung

Die Erkennung des Bereichs von Tabellen mit vollständigem, äußerem Rahmen wie Abb. 4.3 (b) kann durch Erfassen aller horizontalen und vertikalen Linien und ihrer Schnittpunkte erreicht werden. Die auf dieser Idee basierenden Implementierungsschritte sind wie folgt. Darüber hinaus sind die vollständigen Prozessdetails in Abbildung 4.4 zu finden.

1) **Extraktion horizontaler und vertikaler Linien.** Die Extraktion horizontaler und vertikaler Linien kann auf zwei Wegen, wie in Abb. 4.4 (a) und (b), erfolgen: durch Algorithmen zur Erkennung gerader Linien oder durch morphologische Operationen wie Erosion und Dilatation. Es sollte beachtet werden, dass beim Extrahieren gerader Linien unter Verwendung morphologischer Operationen die Eingabe nicht neigen darf, was höhere Anforderungen an die Neigungskorrektur stellt. Im Gegensatz dazu können Linienerkennungsalgorithmen gerade Linien mit Neigung erkennen, jedoch sind einen zusätzlichen Screening-Schritt erforderlich, um gerade Linien innerhalb des Textes herauszufiltern.

2) **Suche nach den zusammenhängenden Linien.** Nach der Extraktion werden, werden die zusammenhängenden Linien mittels der Funktion *cv2.findContours()* bestimmt. Zu diesem Zeitpunkt wird der Satz sich schneidender Linien in denselben rechteckigen



(a) Extraktion mittels morphologischen Operationen

(b) Extraktion mittels Linienerkennungsalgorithmus

(c) Extraktion von zwei Tabellen

Abbildung 4.4. Workflow linienerkennungsbasierter Tabellenbereichserkennung.

Bereich unterteilt. Auf diese Weise ist es möglich, zwei Tabellen im Bild (zum Beispiel in Abb. 4.4 (c)) zu unterscheiden.

3) **Bestimmung der Tabellenbereich.** Alle erhaltenen rechteckigen Bereiche werden gefiltert, und der Bereich mit der größeren Fläche wird als Tabellenbereich ausgewählt.

4.2.2. Textblöckebasierte Tabellenbereichserkennung

Die Tatsache, dass die Tabellen unterschiedliche Spalten haben, impliziert, dass Lücken zwischen den Feldern wesentlich größer sind als die Lücken zwischen den Wörtern in Textzeilen. Nach Durchführung morphologischer Operationen mittels geeigneten Kernen werden daher die Wörter in derselben Zeile des Textbereichs zu einem Ganzen zusammengeführt, während die Wortblöcke, die zu unterschiedlichen Spalten in der Tabelle gehören, getrennt werden. Signifikante Unterschiede zwischen Tabellen- und Textbereich sind auf diese Weise zu beobachten. Dementsprechend wird ein einfacher, aber effizienter Algorithmus implementiert. Die graphische Darstellung des Prozesses wird in Abbildung 4.5 veranschaulicht. Es ist erwähnenswert, dass fast alle Tabellen, ob sie vollständigen Linien (Abb. 4.3 (a)) oder wenigen Linien (Abb. 4.3 (b) und (c)) aufweisen, nach dem Entfernen gerader Linien zu einer Tabelle mit rahmenlosen Zellen (Abb. 4.3 (d)) normalisiert werden können, sodass diese Methode theoretisch auf die meisten Arten von Tabellen anwendbar ist.

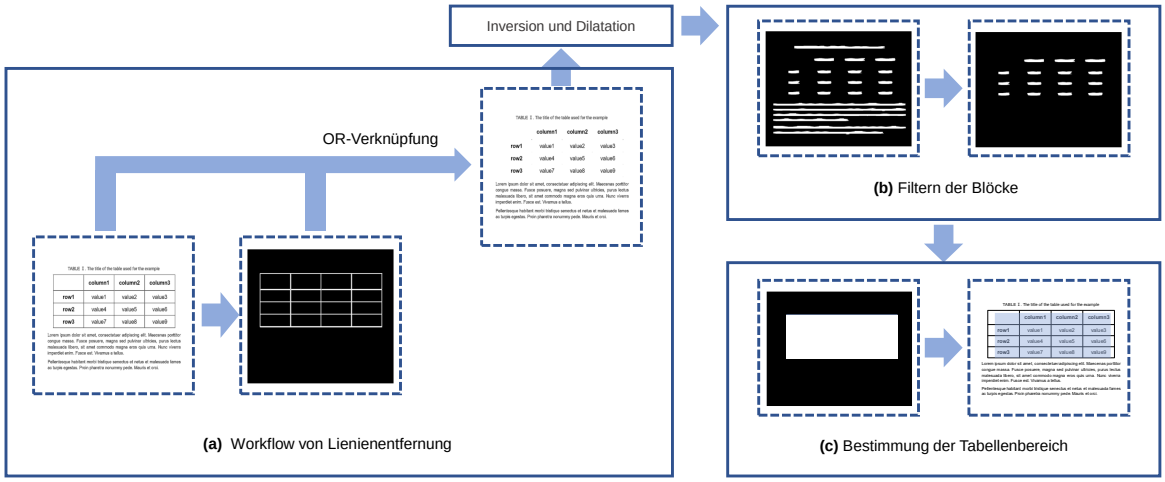


Abbildung 4.5. Workflow textblöckebasierter Tabellenbereichserkennung.

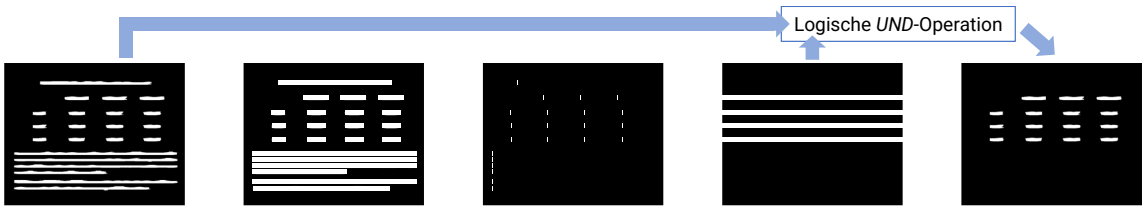
1) **Linienentfernung.** Vertikale Linien zwischen Tabellentextspalten führen manchmal dazu, dass Wörter, die zu unterschiedlichen Spalten gehören, irrtümlicherweise miteinander verschmolzen werden. Das führt dazu, dass der Tabellenbereich fälschlicherweise als Textbereich eingeschätzt wird. Außerdem werden horizontale Linien fälschlicherweise als Textzeilen erkannt. Daher werden alle geraden Linien entfernt, bevor morphologische Operationen zur horizontalen Zusammenführung von benachbarten Buchstaben durchgeführt werden. Linien werden zuerst lokalisiert, entweder durch Liniendetektionsverfahren oder durch morphologische Manipulation. Dann wird ein schwarzes Hintergrundbild, das nur weiße gerade Linien enthält, mit dem Originalbild mit der Funktion *cv2.bitwise_or()* OR-verknüpft, um die geraden Linien zu entfernen. Die Visualisierung dieses Prozesses ist in Abbildung 4.5 (a).

2) **Inversion und Dilatation.** Nachdem alle geraden Linien entfernt werden, wird eine Schwarz-Weiß-Inversion auf das Bild durchgeführt und dann alle weißen Pixel horizontal mittels eines Kerns mit geeigneter Größe dilatiert. Zu diesem Zeitpunkt werden Wörter bzw. Buchstaben, die näher beieinander liegen, zu einem Ganzen verschmolzen, während Buchstaben, die weit auseinander liegen, nicht verschmolzen werden. Der in dieser Implementierung verwendete Kernel ist folgend.

$$Kernel = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.1}$$

3) **Filtern der Blöcke.** Nach Dilatation ist die Anzahl der Textblöcke in jeder Zeile des Tabellenbereichs und des Textbereichs signifikant unterschiedlich: Jede Zeile im Textbereich enthält nur einen langen Textblock, während sich in jeder Zeile des Tabellenbereichs mindestens zwei Textblöcke befinden. Dadurch werden Zeilen gelöscht, die

nicht zur Tabelle gehören (Abb. 4.5 (b)). Das spezifische Implementierungsverfahren ist in der Abbildung 4.6 gezeigt: Zuerst wird die Funktion *cv2.findContours()* auf dem linken Bild in Abb. 4.5 (b) durchgeführt, um jeden Textblock zu regularisieren. Dann wird jeden Farbblock nach links zu einer weißen Linie mit einer Breite von einem Pixel komprimiert. Danach werden die Pixel des Bildes zeilenweise aufsummiert. Da der schwarze Pixelwert 0 und der weiße Pixelwert 1 ist, ist der Wert des Pixels gleich der Anzahl von Textblöcken in der Zeile. Auf eine neue, schwarze Basiskarte werden die Zeilen mit Pixelwerten größer als 1 im Originalbild als reines Weiß markiert. Auf dieser neuen Basiskarte und dem originalen Bild wird dann die logische *UND*-Operation durchgeführt, um das rechte Bild in Abb. 4.5 (b) zu erhalten.



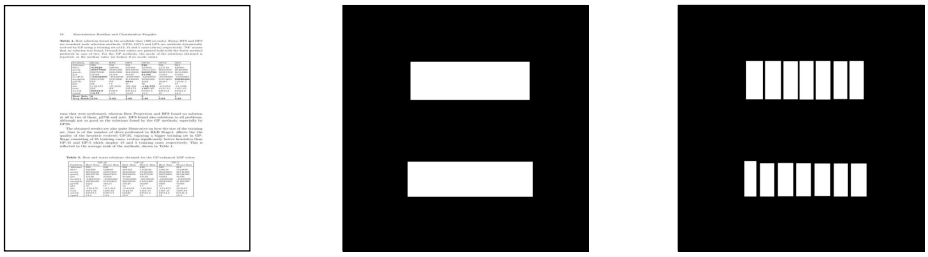
(a) Regularisierung. **(b)** Komprimierung. **(c)** Statistik und Screening.

Abbildung 4.6. Filtern der Blöcke.

4) **Bestimmung der Tabellenbereich.** Da nun nur noch die zur Tabelle gehörenden weißen Textblöcke in der Abbildung übrig sind, wird darauf morphologische Closing-Operation[Goy11] durchgeführt. In der Abbildung 4.5 (c) wird der auf diese Weise bestimmte Tabellenbereich dargestellt. Textblöcke, die zu derselben Tabelle gehören, werden zusammengeführt, um den Umfang der Tabelle anzuzeigen. Unter Nutzung entsprechender Kernelgröße beim morphologischen Closing können zwei Tabellen im selben Graphen voneinander unterschieden werden.

4.2.3. Tabellenbereichserkennung basierend auf tiefen, neuronalen Netzen

Die Aufgabe des Identifizierens von Tabellenbereichen in Bildern kann durch die Verwendung neuronaler Netze zur Objektdetektion oder Objektsegmentierung erreicht werden. Da die Rauschtoleranz bei der Tabellenbereichserkennung viel geringer ist als bei der Erkennung der natürlichen Objekte, ist es angemessener, Tabellenbereiche Pixel für Pixel durch Objektsegmentierung vorherzusagen, anstatt Tabellenpositionen durch Objekterkennung zu regressieren[PVR+19]. Der Einsatz neuronaler Netze führt nicht nur zu einem robusteren Erkennungsmodell, sondern erspart auch langwierige Zwischenschritte und kann nach dem Trainieren des Modells offline verwendet werden. Während dieses Prozesses werden zwei unterschiedliche semantische Segmentierungsmodellarchitekturen ausprobiert, eine basierend auf U-net[RFB15] und eine basierend auf DenseNet-121[HLVDMW17] und TableNet[PVR+19]. Details



(a) Originales Bild (b) Tabellenbereichsmaske (c) Spaltenbereichsmaske

Abbildung 4.7. Beispielbild im Datensatz zusammen mit den entsprechenden Masken.

wie die Struktur dieser beiden Modelle werden am Ende dieses Abschnitts in Abbildungen 4.9, 4.10 und 4.11 vorgestellt. Die Schritte zum Trainieren sind wie folgt:

1) **Erstellen des Datasets.** Das hierbei verwendete originelle Dataset kommt aus Marmot Dataset[FTT+12], wird weiterhin von Sagar[Sag21] erweitert. Dieser erweiterte Datensatz enthält 1004 Seiten aus chinesischen oder englischen Publikationen mit entsprechenden Tabellen- und Spaltenbereichsmasken. Zusätzlich wurden dem Datensatz insgesamt 162 Bildseiten und die dazugehörigen Tabellen- und Spaltenbereichsmasken aus den vom Robert Koch Institut (RKI) zwischen April und Juli 2022 veröffentlichten deutschen Covid-19-Wochenberichten[Ins22] hinzugefügt. Nach dem subjektiven Filtern von Bildern schlechter Qualität umfasst der verwendete endgültige Datensatz insgesamt 1155 Originalbilder und seine Tabellenbereichs- und Spaltenbereichsmasken, darunter 993 Bilder aus dem erweiterten Datensatz und 162 Bilder aus RKI. Alle Bilder sind im JPG-Format mit drei Kanälen (Bittiefe: 24). Ein Beispiel ist in Abbildung 4.7 darzustellen.

2) **Unterteilung des Datasets.** Um den Fortschritt des Modelltrainings zu beurteilen und eine Überanpassung zu verhindern, muss der Datensatz in Trainingssatz und Validierungssatz aufgeteilt werden[Sha17]. In dieser Arbeit werden 80 % der Bilder (insgesamt 955 Bilder) in den Trainingssatz und 20 % der Bilder (insgesamt 200 Bilder) in den Validierungssatz unterteilt. Zusätzlich besteht der Testsatz, der statt zum Training nur zur Bewertung der Modellleistung nach dem Training dient, aus 85 Seiten der Covid-19-Wochenberichte des RKI[Ins22] im Januar 2022.

3) **Erstellen des Datenloaders.** Vor dem Training sollte eine Class *ReadDataset()* zum Datenlesen eingerichtet werden. Diese Unterklasse erbt die Oberklasse *Dataset* im *torch.utils.data* Paket. Außerdem werden die Datenaugmentation beim Lesen der Bilder verwendet, die als Regularisierer fungieren[SK19]. Somit wird die Datenmenge erhöht, indem leicht modifizierte oder neu erstellte synthetische Daten aus vorhandenen Daten generiert werden, was zur Reduzierung der Überanpassung beiträgt. Für die Datenaugmentation wird das Paket *albumentations* eingesetzt[BIK+20]. Schließlich wird die Class

ReadDataset() in der Funktion *GetTrainVal()* aufgerufen, und dann lädt die Class *Dataloader()* die Daten stapelweise als iterierbares Objekt. Hiebei steuert der Parameter *batch_size* die Anzahl der Bilder, die in einen Datenstapel geladen werden. Zu beachten ist, dass die geladenen Masken dem originalen Bild entsprechen sollte.

4) **Training.** Das zu trainierende Modell sollte auf Graphics Processing Unit (GPU) geladen werden. Anschließend werden die Gesamtzahl der Training-Epochs und die Stapelgröße festgelegt und die anfängliche Lernrate angegeben. Hier wird auch der Adaptive Moment Estimation (Adam) Optimierer verwendet, um die Lernrate dynamisch zu optimieren, und die Funktion *ReduceLROnPlateau* im Paket *torch.optim.lr_scheduler* wird basierend auf der Verlust des Modells auf dem Validierungssatz für den Lernratenabfall zur effizienten Suche nach dem minimalen Verlust verwendet. Der Verlust wird mit der Funktion Binary CrossEntropy (BCE)[Com12] Loss berechnet. Aufgrund der Beschränkung von Google Colab werden je 4 Bilder (und entsprechende Tabellenbereichsmasken wie die Abb. 4.7 (b) zur Berechnung von Loss) in diesem Training als ein Batch zur Aktualisierung der Modellparameter mittels Batch-Gradientenabstieg geladen. Alle 1155 Bilder werden deswegen in einem Epoch des Trainings in zufälliger Reihenfolge in 289 Batchs geteilt, und der Datensatz wird für insgesamt 150 Epochs zum Training des Modells geloopt.

5) **Bewertung des Modells.** Nach dem Training wird das Modell mit wenigstem Verlust basierend auf den Visualisierungsergebnissen des Verlusts im Trainings- und Validierungssatz ausgewählt. Nachdem die Modelle mit Unet- und TabletNet-Architekturen erhaltet werden, die ihre jeweiligen minimalen Verluste zeigen, wird der Vergleich zwischen den beiden Modellen an den 85 Bildern im Testsatz durchgeführt, die nicht am Training teilgenommen haben, wobei Mean Intersection Over Union (Miou)[Tiu19] berechnet wird, um die Leistung des Modells zu charakterisieren.

(a) originale Bild. **(b)** Vorhersage des Modells. **(c)** Regularisierung der Vorhersage. **(d)** Bestimmung des Tabellenbereichs.

Abbildung 4.8. Einsetzung des besten Modells und die Regularisierung der Vorhersage.

6) **Einsetzung des besten Modells.** Schließlich wird das beste Modell auf die Tabellenstrukturerkennung angewendet. Zu diesem Zeitpunkt wird die vom Modell ausgegebene Vorhersagekarte nicht direkt zur Bestimmung des Tabellenbereichs verwendet. Im Ta-

bellenbereich werden weiter die morphologische Öffnungsoperation und die *cv2.findcontour()* Funktion durchgeführt, um ein regelmäßiges Rechteck zu bekommen. Dieser Vorgang ist in der folgenden Abbildung 4.8 dargestellt.

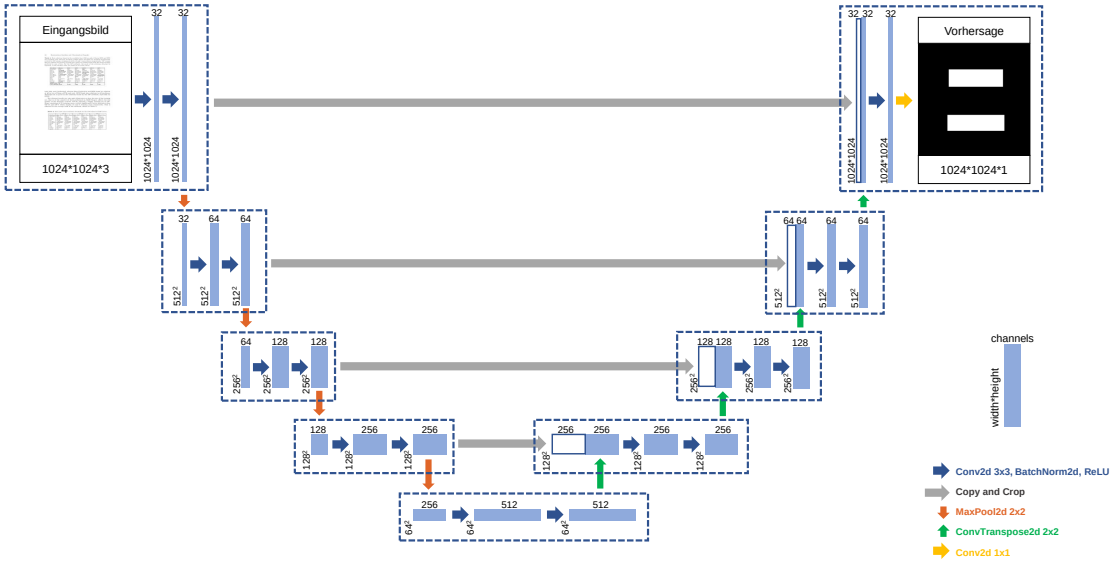


Abbildung 4.9. Struktur des geänderten, auf U-net[RFB15] basierten Modells.

Ein der in dieser Arbeit verwendetes Modelle basiert auf dem originalen U-Net-Modell[RFB15]. Die Struktur wird jedoch geändert, um sich an der Tabellenbereichserkennung anzupassen. Wie in Abbildung 4.9 hat das Modell eine symmetrische Struktur basierend auf der Encoder-Decoder-Idee. Das Ganze lässt sich in drei Teile unterteilen: der Merkmalsextraktionsteil (Encoder) links, der Upsampling-Teil (Decoder) rechts und der Merkmalsspleißungsteil in der Mitte. Der Merkmalsextraktionsteil ist ein herkömmliches CNN, das aus fünf Merkmalsextraktionsblöcken besteht. In jedem Block wird der Durchgang von 3×3 Faltung (Conv) mit Stride 1, Batchnormalisierung (BN)[IS15] und Rectified Linear Unit (ReLU)[GBB11] zweimal zur Merkmalsextraktion verwendet, wobei in dem ersten Durchgang sich die Anzahl der Kanäle verdoppelt und in dem zweiten die gleiche Anzahl beibehält. Die Merkmale werden dann nach Max-Pooling in den nächsten Block eingespeist. Nach jedem Merkmalsextraktionsblock verdoppelt sich die Anzahl der Kanäle, was bedeutet auch, dass die Anzahl der extrahierten Merkmale verdoppelt wird. Im Gegensatz dazu ist die Größe des Bilds halbiert. Die nach viertem Max-Pooling extrahierten Merkmale werden als Eingabe für den Decoder-Teil verwendet. Die Eingabe jedes Decodierungsblocks in der rechten Hälfte verkettet der entsprechenden Merkmalskarte, die vom Merkmalspleißungsteil in der Mitte gespeist wird, und der Ausgabe der 2×2 Transposed Convolution[Mis20] mit Stride 2 und einem Kanalkomprimierungsverhältnis 0,5. Die Eingabe wird dann in jedem Dekodierungsblock nach zwei Durchgängen von 3×3 Conv, BN und ReLU mit halbierten Kanalanzahl ausgegeben und nach der nächsten

Transposed Convolution in den nächsten Decodierungsblock eingespeist. Im letzten Dekodierungsblock wird eine 1×1 Conv anstelle des zweiten Durchgangs von 3×3 Conv, BN und ReLU eingesetzt. Somit wird eine Karte mit Größe 1024×1024 generiert, um den Tabellenbereich vorherzusagen. Darüber hinaus wird während alle Faltungsoperationen ein Padding mit der Größe 1 hinzugefügt, um die Bildgröße vor und nach der Faltung unverändert zu lassen.

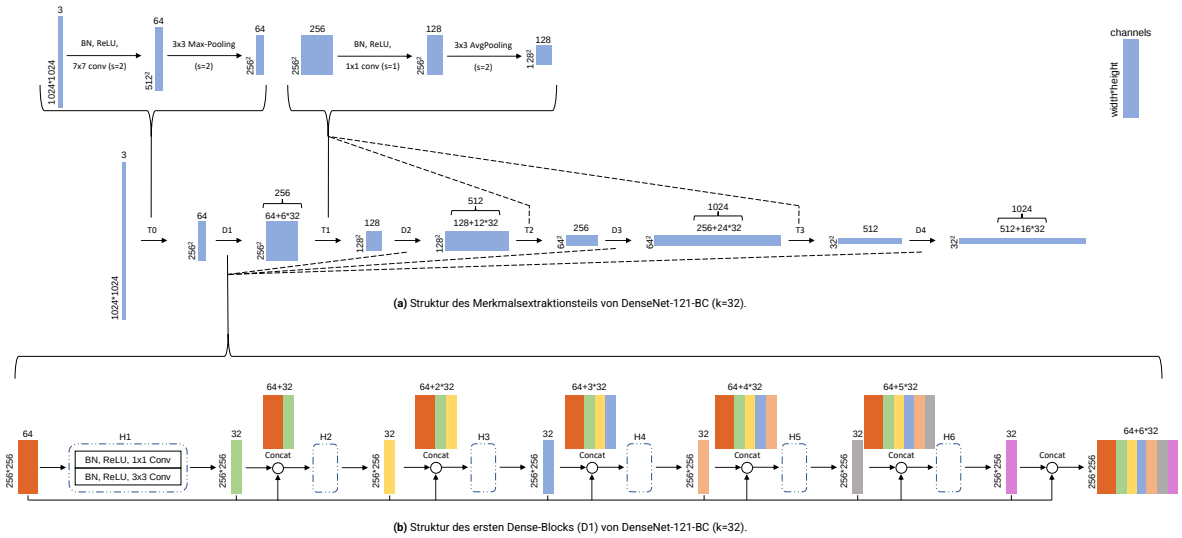


Abbildung 4.10. Struktur des *DenseNet-121-BC (k=32)* Backbones nach Huang[HLVDMW17].

Außerdem wird ein weiteres semantisches Segmentierungsnetz mit TableNet-Architektur trainiert, wobei DenseNet-121 mit Bottleneck Layers und Compression[HLVDMW17] (DenseNet-121-BC) als Backbone zur Merkmalsextraktion eingesetzt. DenseNet-121-BC besteht aus vier Dense-Blöcke, ein anfänglicher Transition-Layer, drei Transition-Layers und ein Classification-Layer. In jedem Block werden alle vorherigen Schichten dicht mit den folgenden Schichten verbindet, um die Merkmale wiederzuverwenden. Somit wird das Problem gelöst, dass der Gradient während der Backpropagation in Deep-CNN verschwindet zu sein. Spezifisch ist die Eingabe jeder Schicht die Concatenation aller vorherigen Schichten in der Kanaldimension, was als Formel 4.2 formuliert werden kann, wobei sich X_l auf die Ausgabe der l^{ten} Schicht bezieht, und sich $[X_0, X_1, ..., X_{l-1}]$ auf die Concatenation-Operation der in den Layers $0, ..., l-1$ erzeugten Merkmalskarten bezieht. Diese Operation bedeutet, dass zwei oder mehr Ausgabetensoren aus verschiedenen Netzwerkschichten entlang der Kanaldimension verkettet werden. Darüber hinaus ist H_l die Kombinationsfunktion namens nicht-linearer Transformationsfunktion der l^{ten} Schicht, die hier aus der Reihe von BN, ReLU, 1×1 Conv, BN, ReLU und 3×3 Conv besteht[HLVDMW17], wobei die Reihe von BN, ReLU und 1×1 Conv als Bottleneck Layer die Anzahl der Input-Feature-Maps reduziert und somit die Recheneffizienz erhöht. Außerdem wird die Anzahl der von jeder H_l-Funktion erzeugten neuen Feature-Maps in jeder Schicht durch die Wachstumsrate k bestimmt[HLVDMW17]. Die Wachs-

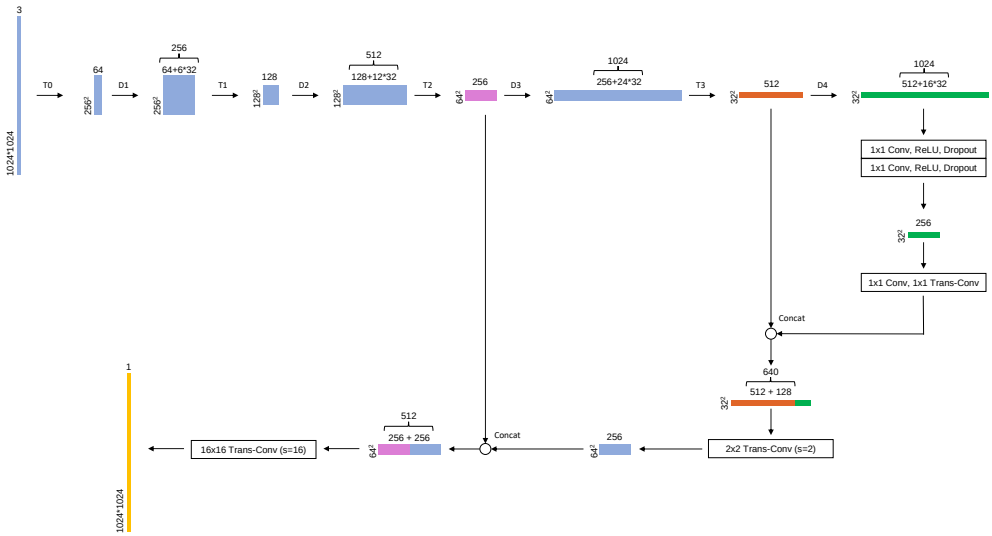


Abbildung 4.11. Modell mit TableNet-Architektur basierend auf dem *DenseNet-121-BC (k=32)* Backbone nach Sagar[Sag21].

tumsrate des diesmal verwendeten Modells wird auf 32 gesetzt. Als Beispiel ist die Struktur des ersten Dense-Blocks (D1) von DenseNet-121-BC in Abbildung 4.10(b) dargestellt.

$$X_l = H_l[X_0, X_1, ..., X_{l-1}] \tag{4.2}$$

Ein wichtiger Teil des Faltungsnetzwerks ist die Downsampling-Schicht, die die Größe der Merkmalskarte ändert. Wenn sich jedoch die Größe der Merkmalskarte ändert, ist die Verkettungsoperation im Formel 4.2 nicht durchführbar. Daher wird die Downsampling-Schicht als Transition-Layer zwischen beiden Dense-Blöcken hinzugefügt, die aus BN, ReLU, 1×1 Conv mit Stride 1 und 2×2 Average-Pooling (AvgPooling) mit Stride 2 besteht[Tor22]. Außerdem wird hier ein Komprimierungsverhältnis von 0,5 verwendet, um die Kanäle der transformierten Merkmalskarten nach Transition Layer zu reduzieren und die Kompaktheit des Modells zu verbessern. Im Vergleich dazu befinden sich in dem anfänglichen Transition-Layer (T0) eine Reihe von BN, ReLU, 7×7 Conv mit Stride 2 und 3×3 Max-Pooling mit Stride 2. Es sollte beachtet werden, dass der Stride-Parameter in unspezifizierten Faltungs- und Pooling-Operationen alle 1 ist. Darüber hinaus nimmt die Faltung innerhalb jedes Dense-Blocks eine geeignete Padding an, um die Größe unverändert zu lassen. Zur Anpassung an die semantische Segmentierungsaufgabe wird der letzte Classification-Layer verworfen und durch einen Decodierungsteil auf Basis der TabletNet-Architektur ersetzt. Bisher ist der Merkmalsextraktionsteil des Modells wie in Abbildung 4.10(a) aufgebaut. Wenn die Eingabe ein Bild mit Größe $1024 \times 1024 \times 3$ ist, werden nach dem letzten Dense-Block 1024 Merkmalskarten mit Größe 32×32 generiert.
Der Dekodierungsteil des Modells wird anhand TableNet-Architektur[PVR+19] wie in Abbil-

dung 4.11 aufgebaut, wobei Features von drei verschiedenen Teilen des DenseNet-121-Encoders in den Decoder eingespeist werden. Die erste Eingabe ist der vom vierten Dense-Block (D4) ausgegebene Feature-Maps-Satz, der als ein Tensor mit Size $[B, 1024, 32, 32]$ gezeigt werden kann, wobei B Batch-Size des Trainings bedeutet und $[..., 1024, 32, 32]$ zeigt, dass der Satz aus 1024 Feature-Maps mit der Größe 32×32 besteht. Die Größe dieses eingegebenen Merkmalssatzes wird als $[B, 256, 32, 32]$ durch eine zusammengesetzte Faltungsschicht verarbeitet, die aus 1×1 Conv, ReLU, Dropout[SHK+14b], 1×1 Conv, ReLU, Dropout besteht. Die Haltewahrscheinlichkeit im Dropout ist 0.8[PVR+19]. Nach einer zusätzlichen 1×1 Conv und eine 1×1 Transposed Convolution mit Stride 1 wird der Feature-Tensor $[B, 256, 32, 32]$ auf $[B, 128, 32, 32]$ verändert und danach mit der zweiten Eingabe verkettet, die mit der Größe $[B, 512, 32, 32]$ aus dem dritten Transition-Layer (T3) kommt. Der auf dieser Wiese generierte Tensor mit Size $[B, 640, 32, 32]$ wird dann nach einer 2×2 Transposed Convolution mit Stride 2 auf $[B, 256, 64, 64]$ verändert und mit der dritten Eingabe aus dem zweiten Transition-Layer (T2) mit Size $[B, 256, 64, 64]$ verkettet. Der generierte Tensor mit Size $[B, 512, 64, 64]$ wird danach in eine 16×16 Transposed Convolution mit Stride 16 eingespeist. Am Ende ist ein $[B, 1, 1024, 1024]$ Tensor ausgegeben, der als Vorhersage des Tabellenbereichs zur Verfügung steht.

4.3. Tabellenstrukturanalyse

Nach dem Positionieren und Ausschneiden des Tabellenbereichs sollte jede Zelle zur Analyse der Tabellenstruktur lokalisiert werden. In diesem Abschnitt werden drei verschiedene Tabellenanalyseverfahren eingeführt, wobei das erste Verfahren auf Linienrekonstruktion und das zweite und dritte auf einer Textblockerkennung basiert. Nach der Lokalisierung der Zellen wird die Zuweisung der Positionslabeln im ersten Verfahren anhand den streng ausgerichteten Koordinationen mit einfacher bedingter Beurteilungen durchgeführt. Da die Koordinationen, die im zweiten und dritten Verfahren mittels Textblöcken bestimmt werden, normalerweise Abweichungen zeigen, wird der K-Means-Clustering[HW79] Algorithmus hierzu eingesetzt. Im dritten Verfahren wird außerdem tiefe, neuronale Netze verwendet, um die Spaltenerkennung zu unterstützen, somit lässt sich die variierenden Kopfzeilen besser bewältigen.

4.3.1. Linienrekonstruktionsbasierte Tabellenstrukturanalyse

Anhand von den vollständigen Zellenrahmen lässt sich die Struktur der Tabelle bzw. die Positionen jeder Zellen und deren Beziehungen zueinander leicht analysieren. Es gibt jedoch viele Tabellen wie Tab. 4.1, die aufgrund nur teilweise expliziter Linien nicht dekonstruiert werden kann. Basierend auf dieser Beobachtung, dass sich ein erheblicher Leerraum zwischen Zeilen oder Spalten befinden, können alle Zellteiler entsprechend wiederaufgebaut werden. Der konkrete Umsetzungsprozess enthält sechs Schritte und wird mit der Beispieltabelle in Abbildung

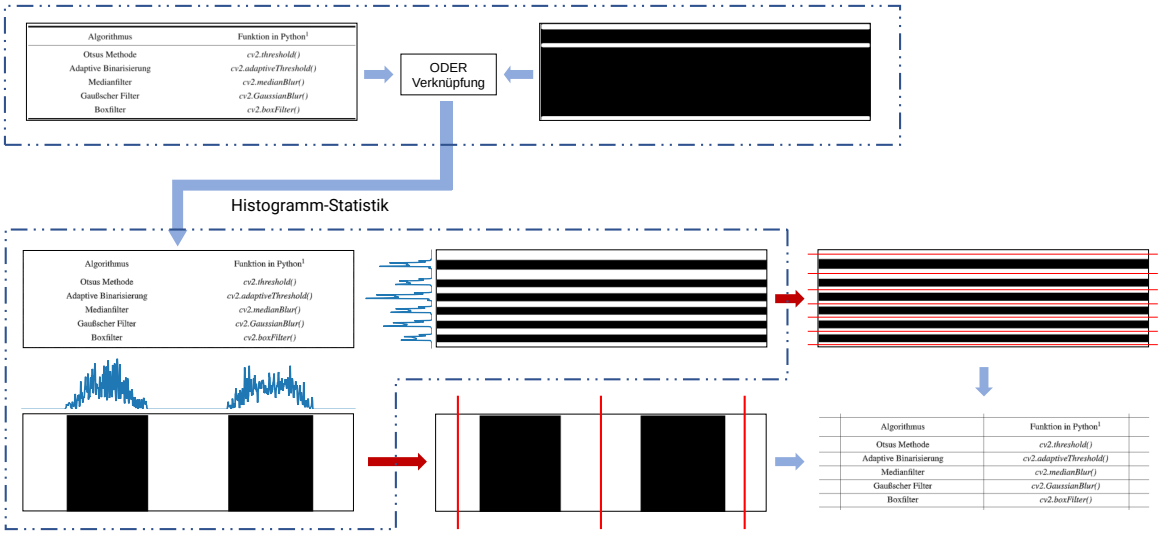


Abbildung 4.12. Umsetzungsprozess linienrekonstruktionsbasierter Tabellenstrukturanalyse auf der Beispieltabelle 4.1.

4.12 dargestellt.

1) **Linienentfernung.** Um den vereinheitlichten Algorithmus zu verwenden, werden alle geraden Linien in der Tabelle entfernt, sodass alle Tabellenvarianten auf Tabellen mit rahmenlosen Zellen normalisiert werden. Alle explizite Linien werden wie im Abschnitt 4.2.2 zuerst lokalisiert, entweder durch Liniendetektionsverfahren oder morphologische Manipulation. Zur Linienentfernung wird dann das schwarze Hintergrundbild, das nur weiße verdickte Linien enthält, mit dem originalen binären Bild ODER-verknüpft.

2) **Histogramm-Statistik.** Da die Pixelwerte im Binärbild nur 0 und 1 enthalten, kann bestimmt werden, ob eine Spalte von Pixeln zu der Lücke zwischen zwei Spalten der Tabelle gehört, indem in vertikaler Richtung die Summe der Pixelwerte jeder Spalte berechnet wird. In ähnlicher Weise wird horizontal Reihe für Reihe jede Pixelzeile als zu einer Lücke oder Textzeile gehörig klassifiziert. Die statistischen Histogramme der Pixelwerte in horizontaler und vertikaler Richtung sind in Abbildung 4.12 dargestellt. Entsprechend den statistischen Ergebnissen sind die zur Zelllücke gehörenden Pixelzeilen und -spalten weiß markiert.

3) **Filtern.** In einigen Fällen wie Abbildung 4.13 (a) ist zu beobachten, dass die Pixel in einer Spalte, die zum Textbereich sortiert werden sollten, fälschlicherweise in Lücken unterteilt sind. Dieses Rauschen auf beiden Extraktionskarten wird durch morphologischen Opening[Goy11] entfernt, somit ist Abbildung 4.13 (b) als Folge. Dieser Vorgang ist im Flussdiagramm 4.12 vereinfacht als roter Pfeil dargestellt.

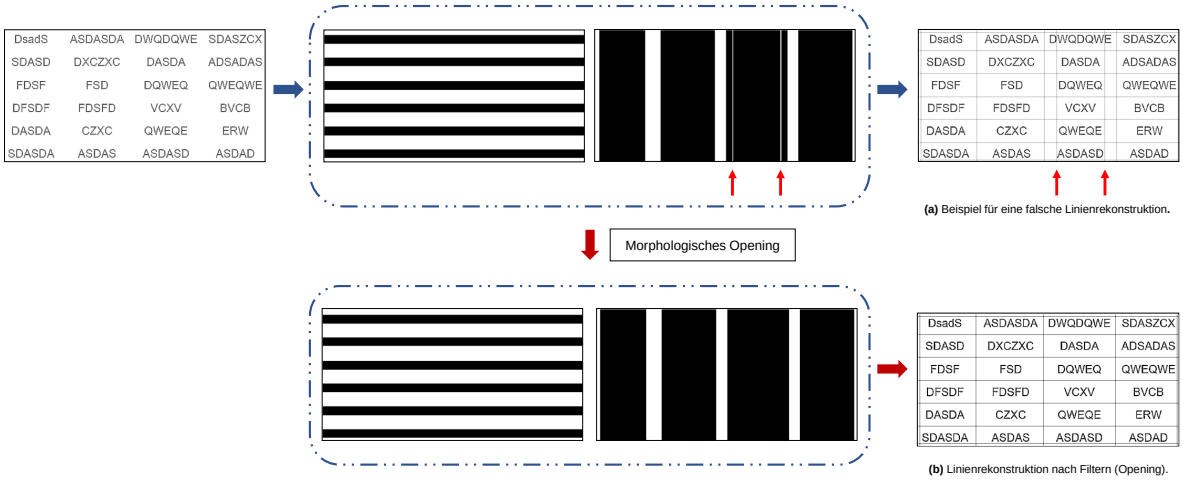


Abbildung 4.13. Beispielbild, das zufällig zu falschen Lückenerkennungsergebnissen führen kann, und das Filtern

4) **Linienrekonstruktion.** Mit der Funktion *cv2.findContours* wird jeder weißer Farbblock lokalisiert, damit die horizontalen oder vertikalen Mittellinien als rekonstruierten Zellrand bestimmt werden.

5) **Lokalisierung jeder Zelle anhand der Schnittpunkte.** Aus den Koordinaten der rekonstruierten horizontalen und vertikalen Linien können alle Schnittpunkte berechnet werden, wodurch jede Zelle genau lokalisiert werden kann. Durch ein Array $[x, y, w, h]$ kann die Position und Größe jeder Zelle eindeutig charakterisiert werden, wobei x und y die Koordinaten der oben links Zellecke sowie w und h die Breite und Höhe sind.

6) **Generierung der Labels.** Die Labels sind hier definiert als die Zeilen- und Spalteninformationen der Zelle, d.h. zu welcher Zeile und zu welcher Spalte die Zelle gehört. Die auf der Grundlage der Linienrekonstruktion generierten Koordinaten des Zellenzentrums sind streng in horizontaler und vertikaler Richtung ausgerichtet, sodass Zellen in derselben Spalte und Zeile unter Verwendung einfacher bedingter Beurteilungen klassifiziert werden können: Zellen, die zu derselben Zeile oder Spalte gehören, haben gleiches y oder x. Dies wird in Abb. 4.20 gesehen. Nun wird der Array, der jede Zelle darstellt, aktualisiert als $[x, y, w, h, row_label, col_label]$, wobei die ersten vier Elemente das Schneiden des Interessengebiets und die letzten beiden Elemente die Rekonstruktion der Tabellenstruktur unterstützen.

4.3.2. Textblöckebasierte Tabellenstrukturanalyse

Ein anderes Verfahren zur Zellenpositionierung in Tabellen, wie Abb. 4.3 (b), (c) und (d), besteht darin, ein morphologisches Closing[Goy11] an dem invertierten Bild durchzuführen, so

dass die Zeichen, die näher beieinander liegen, miteinander vermischt werden. Während die Wörter in einer Zelle auf diese Weise zu einem Textblock zusammengeführt werden, bleiben alle Zellen noch aufgrund der Lücken isoliert, damit jede Zelle positioniert wird. Dieses Verfahren besteht aus den folgenden Schritten, wird in Abbildung 4.14 visualisiert:

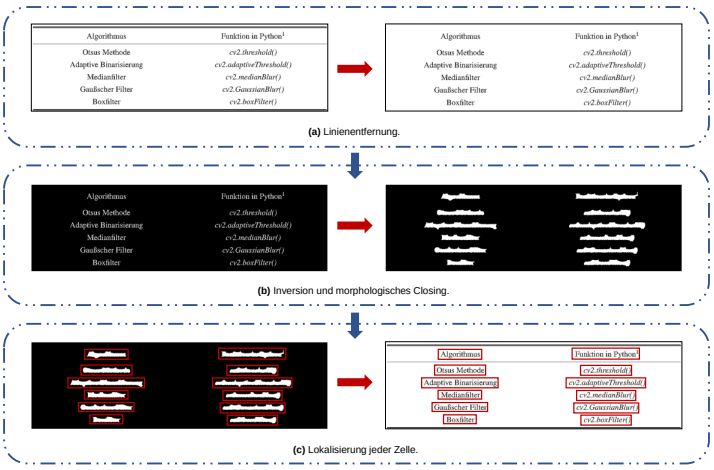


Abbildung 4.14. Umsetzungsprozess textblöckebasierter Tabellenstrukturanalyse auf der Beispieltabelle 4.1.

1) **Linienentfernung.** In diesem Schritt wird alle Linien auf gleiche Weise wie im Abschnitt 4.3.1 entfernt. Die Linienentfernung wird durch eine ODER-Verknüpfung zwischen dem schwarzen Hintergrundbild mit nur weißen Linien und dem Originalbild erreicht.

2) **Inversion und morphologisches Closing.** Die Schwarz-Weiß-Inversion ist normalerweise eine Vorverarbeitung der morphologischen Operation. Danach wird das Bild durch morphologisches Closing verarbeitet, sodass in jeder Zelle die weißen Pixel, die den Text darstellen, erweitert und zu einem einzigen weißen Farbblock verschmelzen. Gleichzeitig muss die Größe des morphologischen Operationskerns gesteuert werden, um sicherzustellen, dass verschiedene weiße Blöcke voneinander isolieren. Jede isolierte weiße Blöcke stellt hierbei die Position jeder Zellen dar, während der schwarze Hintergrund die Lücke zwischen den Zellen bedeutet.

3) **Lokalisierung jeder Zelle.** Zur Lokalisierung und Abmessung aller weißen Blöcke wird die Funktion *cv2.findContours* auf dem schwarzen Bild durchgeführt. Mit dieser Funktion kann die genaue Position und Größe jeder Zelle ermittelt werden, indem jeder Textblock durch das kleinste, nicht-geneigte Rechteck umschlossen wird. Jetzt kann das Array $[x, y, w, h]$ wie in Abschnitt 4.3.1 bestimmt werden, das die Position und Größe der Zelle eindeutig charakterisiert.

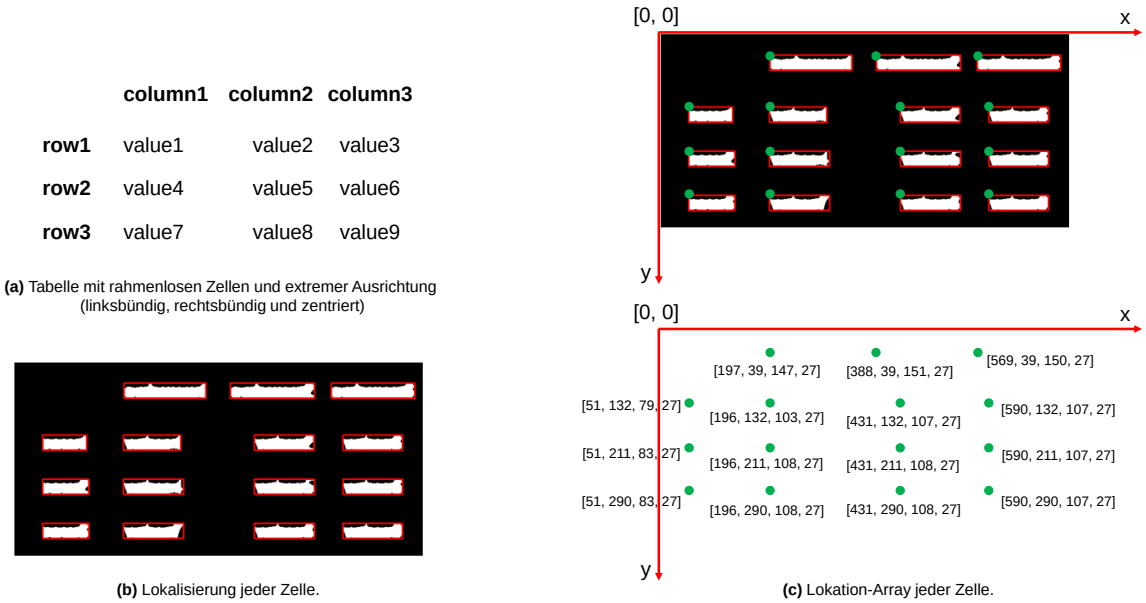


Abbildung 4.15. Beispieltabelle mit linksbündigen, rechtsbündigen und zentrierten Zellen sowie die Lokalisierung der Zellen.

4) **Generierung der Labels.** Im Vergleich zu den Koordinaten der oberen linken Ecke im Abschnitt 4.3.1 sind sie hier in horizontaler und vertikaler Richtung nicht streng ausgerichtet. Diese Abweichungen weisen eine Zufälligkeit aufgrund unterschiedlicher Textblöcke auf, was die Gruppierung beschwert. Wenn der Zelleninhalt linksbündig oder rechtsbündig ausgerichtet ist, werden außerdem die vertikalen Koordinaten x und y, die mittels der Textblöckebasierte Tabellenstrukturanalyse bestimmt werden, ebenfalls drastisch verschoben. Ein Beispiel hierfür ist in der Abb. 4.15 zu finden. Daher sollten die Zellen zunächst gemäß den horizontalen und vertikalen Koordinaten in Sätze von Zeilen und Spalten gruppiert werden. Der zur Gruppierung verwendete Algorithmus ist hier K-Means-Clustering[HW79], der unüberwacht alle Abszissen und Ordinaten nur anhand der im Voraus eingegebener Nummer k jeweils in k Kategorien einteilen kann. Aus den Eigenschaften der Daten selbst sollte die dem besten k-Wert entsprechende Kategorie den Abstand innerhalb der Klasse minimieren und den Abstand zwischen Klassen maximieren. Dieses Merkmal kann anhand des durchschnittlichen Silhouettenkoeffizienten (s_k) mit dem Formel 4.3 bewertet werden[Cha20], wobei $s_k(j)$ der Silhouettenkoeffizient unter dem k-Wert ist, und $a(j)$ der durchschnittliche Abstand zwischen der Datenpunkt j und allen anderen Datenpunkten in dem Cluster, zu dem der Datenpunkt j gehört, sowie $b(j)$ der minimale, durchschnittliche Abstand zwischen der Datenpunkt j und allen anderen Datenpunkten in einem Cluster, zu dem der Datenpunkt j nicht gehört. Der $a(j)$ wird auch Cohesion von Clustering genannt, während $b(j)$ Separation charakterisiert. Außerdem wird in dieser Arbeit der euklidische Abstand verwendet.

$$s_k(j) = \frac{b(j) - a(j)}{max\{a(j), b(j)\}} \tag{4.3}$$

Der Formel kann auch in folgenden Form umgewandelt werden:

$$s_k(j) = \begin{cases} 1 - \frac{a(j)}{b(j)}, & wenn\ a(j) < b(j) \\ 0, & wenn\ a(j) = b(j) \\ \frac{b(j)}{a(j)} - 1, & wenn\ a(j) > b(j) \end{cases} \tag{4.4}$$

Deshalb ist es deutlich zu sehen:

$$-1 \leq s_k(j) \leq 1 \tag{4.5}$$

Ein $s_k(j)$ nahe 1 zeigt an, dass dieser Datenpunkt bei diesem k-Wert richtig geclustert ist. Wenn $s_k(j)$ nahe bei –1 liegt, ist es angemessener, sich in seinen benachbarten Clustern zu gruppieren. Außerdem bedeutet ein $s_k(j)$ nahe 0, dass der Punkt auf der Grenze zweier natürlicher Cluster liegt. Daher ist der maximale Mittelwert von $s_k(j)$ aller Daten im gesamten Datensatz ein Maß für die Angemessenheit der Datenclusterung[KR09].

$$SC = max\ \bar{s}_k \tag{4.6}$$

Der beste k-Wert kann also automatisch bestimmt werden, indem aufzählungsweise der SC-Wert unter jedem k berechnet werden. Der Bereich der Aufzählung wird auf $[2, 50]$ gesetzt. Die endliche k Kategorien beziehen sich auf die k Spalten (oder Zeilen) der Tabellen. So ist jede Zelle durch $[x, y, w, h, row_label, col_label]$ charakterisiert. Der K-Means-Clustering wird in Abb. 4.16 und 4.17 visualisiert.

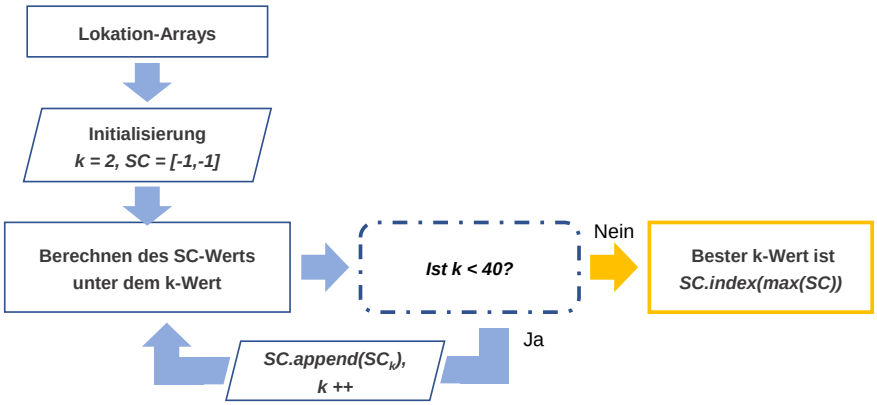


Abbildung 4.16. Workflow zur Bestimmung des besten k-Werts.

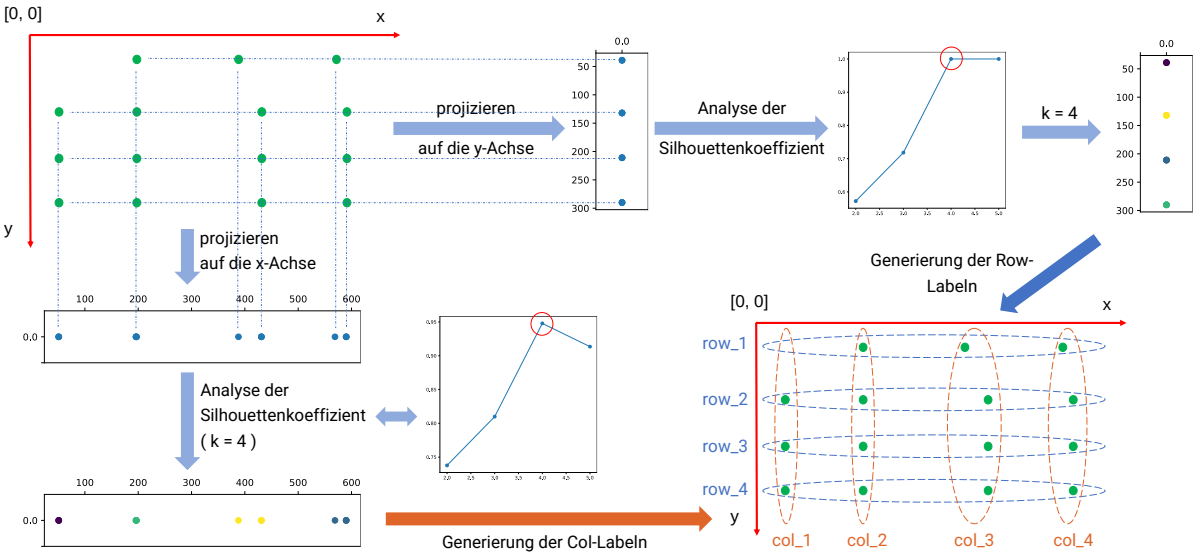


Abbildung 4.17. Visulalisierung des K-Means-Clusterings auf der Tabelle 4.15.

5) **Korrektur der Labels.** Bei Spaltenklassifizierungsaufgaben in komplexen Tabellen ist es nicht immer möglich, den besten k-Wert ausschließlich auf der Grundlage des SC-Werts zu finden. Wenn sich die Länge der Zellen in einer Spalte zu stark unterscheidet, wird diese Spalte in Unterspalten unterteilt, die zusammengeführt bzw. korrigiert werden müssen. In gleicher Weise sollten die Ergebnisse auch bei der Zeilenklassifikation korrigiert werden. Am Beispiel der Spaltenklassifizierung zeigt Abbildung 4.18 (a) die Spaltenklassifizierung einer Tabelle, in der die Zellen mit großen Längenunterschieden enthält, wobei die roten Linien die Cluster-Zentren der Abszisse der oberen linken Zellenecke darstellen. Es ist ersichtlich, dass beispielsweise die vierte Spalte in drei Unterspalten unterteilt wird, die gemäß der durchschnittlichen Länge der Zellen zu einer Spalte zusammengeführt werden müssen. Dazu wird die durchschnittliche Größe aller Textblöcke, die sich in dieser Tabelle befinden, gezählt, wie in der Tabelle 4.2 gezeigt.

Tabelle 4.2. Statistik der Textblöcke in der Tabelle 4.18 (a).

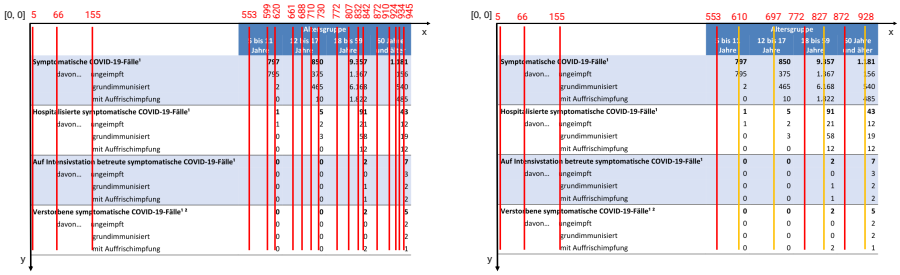
Größe	Mittelwert μ[1]	Standardabweichung s[2]	Referenzgröße R[3]
Weite	37.1	68.6	29.3
Höhe	10.9	5.87	10.9

[1] $\mu = \frac{1}{N}\sum_{i=1}^{N} X_i$

[2] Standardabweichung der Grundgesamtheit $s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(X_i - \mu)^2}$

[3] Mittelwert unter Three-Sigma Limit[KEN22]

Die Schwierigkeit der Spaltenklassifikation wird von der großen Standardabweichung der Zellenweite widergespiegelt. Als allgemeinere Parameter zum Korrigieren der Ergebnisse von Spalten-Clustering wird der Mittelwert unter Three-Sigma Limit[KEN22] verwendet. Dementsprechend werden in dieser Arbeit die Spalten-Cluster-Zentren, deren Abstand kleiner als die Referenzgröße (29.3 Pixels) der Länge von Textblöcken ist (und die Zeilen-Cluster-Zentren, deren Abstand kleiner als die Referenzgröße (10.9 Pixels) der Höhe ist), zusammen integriert. Die korrigierten Ergebnisse von Abbildung 4.18 (a) werden in (b) gezeigt.



(a) Cluster-Zentren der Tabelle. **(b)** Cluster-Zentren der Tabelle nach Korrektur.

Abbildung 4.18. Korrektur der Ergebnisse von K-Means-Clustering auf der Tabelle *Wochenbericht 2022-01-06 24*.

4.3.3. Tabellenstrukturanalyse mit Unterstützung neuronaler Netze

Wie in der Tabelle 4.2 analysiert, unterscheiden sich die Zellenweiten in den Spalten der komplexen Tabelle stark, was die strukturelle Analyse der Tabelle erschwert. Darüber hinaus erhöht die gemischte Verwendung mehrerer vertikaler Ausrichtungsmethoden auch die Schwierigkeit der Analyse. Daher wird es versucht, ein semantisches Segmentierungsmodell zur Unterstützung der Spaltenerkennung einzusetzen. Durch eine manuelle Kennzeichnung der Bereiche in der Tabelle, wie Abb. 4.7 (c), die als Spalten identifiziert werden, kann das Modell zwei Spalten, die unter derselben Überschrift der ersten Ebene liegen, genauer unterscheiden. Im Gegensatz zu Spalten haben die Zeilen innerhalb einer Tabelle weniger Varianten, worauf führt zurück, dass der K-Means-Clustering für die Zeilenanalyse ausreichend ist.

1) **Training des Modells.** Das hier verwendete Modell, die Parameter für das Modelltraining und der für das Training verwendete Datensatz ähneln denen in Abschnitt 4.2.3. Wie in Abb. 4.7 (c) gezeigt, entsprechen die geladenen Masken hier den Spaltenbereichen innerhalb der Tabelle.

2) **Vorhersage jeder Spalte mittels des trainierten Modells.** Eine Maskenkarte für Spalten wird unter Verwendung des Modells vorhergesagt, um jede Spalte in der Tabelle zu

identifizieren. An den vorhergesagten Ergebnissen wird dann eine morphologische Öffnung durchgeführt, wobei der Betriebskern mit einer größeren Größe in der vertikalen Richtung verwendet wird, um eine möglichst genaue Spaltenfläche zu erhalten.

3) **Bestimmung der Basiskoordinaten.** Die Funktion *cv2.findContours* wird auf der vom Modell vorhergesagten Spaltenmaskenkarte ausgeführt, um den Spaltenbereich zu lokalisieren. Durch weitere Verarbeitung werden die Mittellinienkoordinaten der Spalte als die Basiskoordinaten in vertikalen Richtung erhalten. Außerdem werden die linken und rechten Ränder jeder Spalte markiert, um die Zugehörigkeit der Zellen in nachfolgenden Verarbeitungsschritten zu bestätigen. Die Ergebnisse sind in Abbildung 4.19 dargestellt, wobei die rote Linie die Basiskoordinaten und die grüne Linie den Bereich jeder Spalte darstellt.

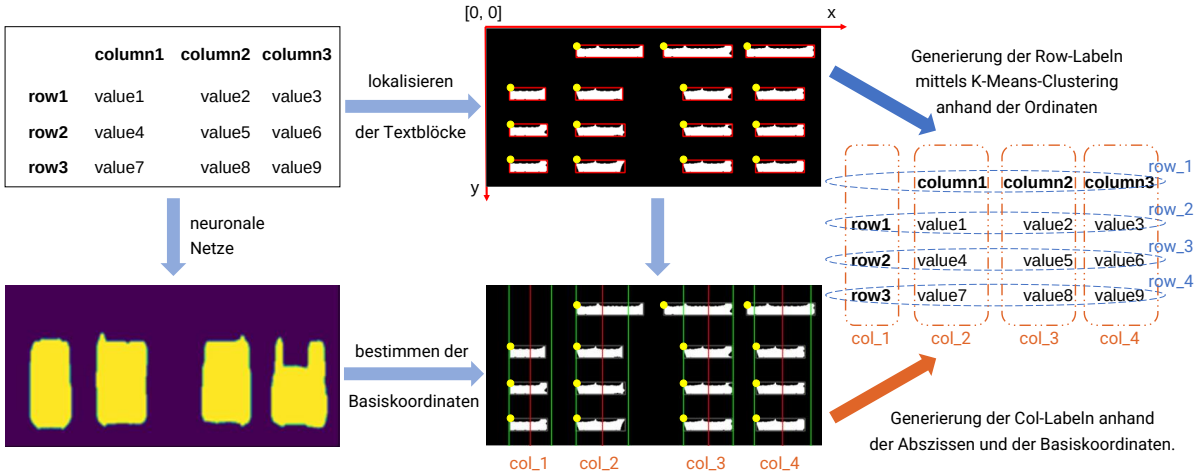


Abbildung 4.19. Tabellenstrukturanalyse mit neuronalen Netzen auf der Tabelle 4.15.

4) **Lokalisierung der Textblöcke.** Hier werden die ersten drei Schritte im Abschnitt 4.3.2 durchgeführt, nämlich Linienentfernung, Inversion und Closing und Lokalisierung jeder Zelle. Die Koordinaten und die Größe jeder Zelle werden dadurch identifiziert und in den Array $[x, y, w, h]$ zusammengefasst.

5) **Generierung der Col-Labels anhand der Abszissen x und der Basiskoordinaten.** Die Generierung von col_label erfolgt basierend auf den im vorherigen Schritt erhaltenen Basiskoordinaten und den Abszissen x der oberen linken Zellenecke. Hier werden die Zellen, deren Spaltenkoordinaten sich innerhalb des Bereichs befinden, der durch die grünen Linien auf beiden Seiten in der Nähe einer roten Linie begrenzt ist, das col_label von der roten Linie erhalten. Wenn die Abszissen x der oberen linken Zellenecke zu keinem definierten Bereich gehört, wird die Zelle der nächsten Spalte zugewiesen.

6) **Generierung der Row-Labels anhand der Ordinaten y.** Das $row_l abel$ wird mittels des K-Means-Clustering Algorithmus bestimmt, wie im Abschnitt 4.3.2. Die Labels jeder Zelle sind endlich im Form $[x, y, w, h, row_label, col_label]$.

4.4. Texterkennung und Tabellenrekonstruktion

Die optische Erkennungsoperation in dieser Arbeit wird eher an segmentierten Bildern einzelner Zellen als an vollständigen Tabellen durchgeführt. Dies ist vorteilhaft, um das Rauschen zu reduzieren. Jede Zelle wird entsprechend den originellen Koordinaten x, y, w und h im Array $[x, y, w, h, row_label, col_label]$ abgeschnitten werden. Nach dem Erhalten der Inhalt wird das durch optische Erkennung erhaltene Zeichen zum Array hinzugefügt und in $[result, row_label, col_label]$ vereinfacht. Daraus lässt sich die gesamte Tabelle mittels des Pakets *pandas* rekonstruieren. Die Details des Implementierungsprozesses sind wie folgt:

1) **Festlegen des Interessengebiets (ROI).** Das Interessengebiet bezieht sich hier auf jeden Zellbereich, der durch die Koordinaten und die Größe jeder Zelle, die im Array $[x, y, w, h, row_label, col_label]$ als x, y, w und h gespeichert werden, gekennzeichnet ist.

2) **Optical Character Recognition (OCR) auf dem ROI.** Der Inhalt jeder Zelle wird mittels *Tesseract* identifiziert. Die Zellen werden hierbei mit *„(empty_cell)“* gekennzeichnet, wenn ihr Inhalt nicht korrekt identifiziert wird oder leer ist. Nach OCR wird der Array jeder Zelle in $[result, row_label, col_label]$ erneuert.

3) **Rekonstruktion der Tabelle.** Anschließend wird die Funktion *DataFrame* im Paket *pandas* aufgerufen, um den Array $[result, row_label, col_label]$ in eine tabellarische Datenstruktur zu konvertieren und die interaktive Tabelle zu rekonstruieren. Außerdem steht der Zeichen *„(empty_cell)“* zur Verfügung, um *„NaN“* in dem rekonstruierten *DataFrame* zu erfüllen.

Wenn leere Zellen vorhanden sind, unterscheiden sich die Lokation-Arrays, die mittels der linienrekonstruktionsbasierten Tabellenstrukturanalyse erhalten werden, von den Lokation-Arrays, die durch die textblöckebasierte Tabellenanalyse erhalten werden. Dies ist in Abbildung 4.20 zu sehen. Aber die rekonstruierte *DataFrame*s sind gleich, da die basierend auf der linienrekonstruktionsbasierten Tabellenstrukturanalyse erhaltene ROIs leere Zellen enthält, die in der OCR als *„(empty_cell)“* erkannt werden, während die auf der Grundlage der textblöckebasierte Tabellenanalyse erhaltene ROIs keine leeren Zellen enthält, die jedoch zu *„NaN“* im *DataFrame* führen und durch *„(empty_cell)“* ersetzt wird.

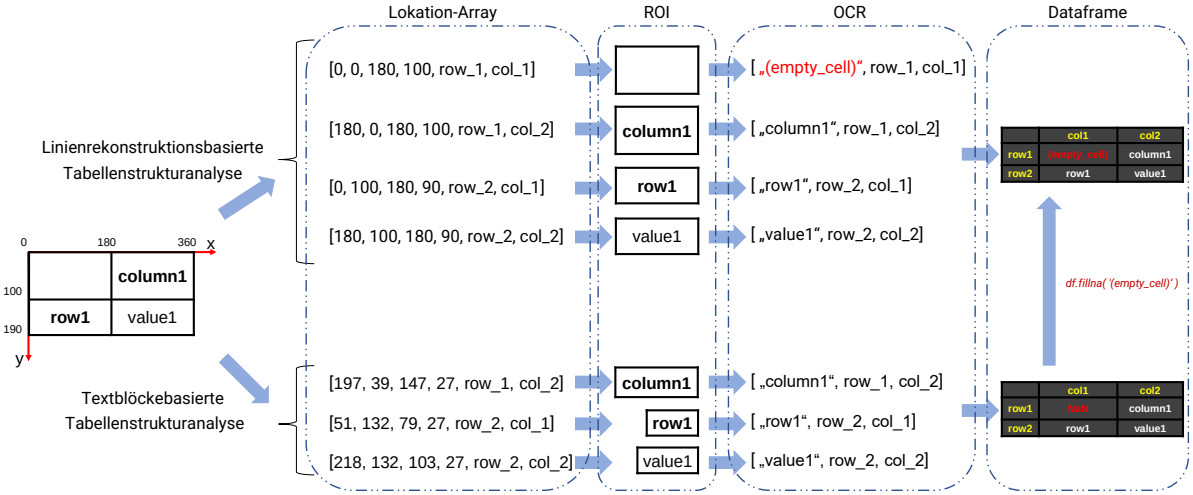
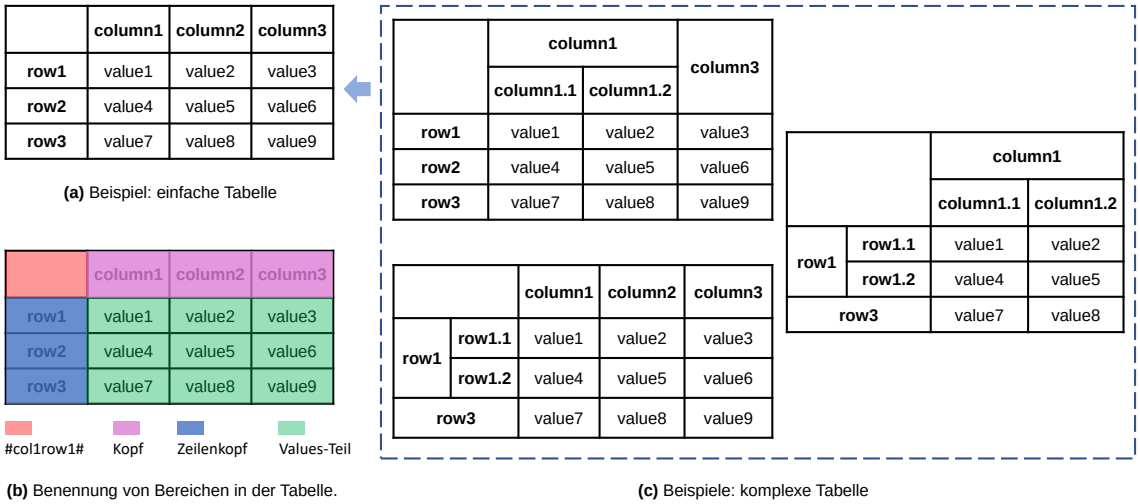


Abbildung 4.20. Texterkennung und Tabellenrekonstruktion auf einer Beispieltabelle.

4.5. Nachverarbeitung und Einschreibung in Datenbank

Um die Nachbearbeitung zu erleichtern, wird zunächst die Struktur der Tabelle festgelegt. Die Strukturen der einfachen Tabelle und der komplexen Tabelle sind in Abbildung 4.21 (a) und (c) zu vergleichen: Einfache Tabelle hat einzeiligen Kopf und einspaltigen Zeilenkopf, während komplexe Tabelle mehrzeiligen Kopf oder / und mehrspaltigen Zeilenkopf hat.
Es ist in der Tabellenstrukturanalyse erwähnenswert, dass die Generierung der Positionslabels auf die obere linke Ecke jeder Zelle basiert. Das führt dazu, dass die spaltenübergreifenden Köpfe in der komplexen Tabelle nach der Rekonstruktion zu der nächsten linken Spalte zugewiesen werden. In der originalen Tabelle wird eine Spalte unter dem spaltenübergreifenden Kopf zu diesem Kopf zugeordnet. Der Computer kann diese Zuordnungen jedoch in der rekonstruktierten Tabelle nicht finden. Diese Zuordnung ist aber beim Schreiben in die Datenbank notwendig, da sonst wichtige Informationen verloren gehen. Daher wird zuerst eine Nachbearbeitung durchgeführt, um diese Beziehung zu rekonstruieren. Hierbei wird der spaltenübergreifende, primäre Kopf kopiert und mit den entsprechenden untergeordneten Köpfen zu neuen Köpfen zusammengeführt. In Abbildung 4.22 werden die zu kopierenden Zellen mit den roten Kästchen markiert, und die leeren Zellen, die durch eine kopierte Zelle einfüllt werden müssen, mit den grünen Kästchen. Außerdem markieren die blaue Kästchen die Regionen, in denen Verschmelzungsoperationen durchgeführt werden müssen.
Ziel ist es, einen robusten Weg zu finden, komplexe Tabellen in einfache Tabellen umzuwandeln (von Abb. 4.22 (a) zu Abb. 4.22 (b)), wofür zunächst die folgenden sorgfältig ausgearbeiteten Regeln spezifiziert werden. Die Visualisierung der Regeln ist in Abb. 4.23 zu finden.



(a) Beispiel: einfache Tabelle

(b) Benennung von Bereichen in der Tabelle.

(c) Beispiele: komplexe Tabelle

Abbildung 4.21. Vergleichung von einfachen und komplexen Tabellen und die Benennung der Teile.

Regel 1) **Benennung von Bereichen in der Tabelle.** Die Tabelle sollte als aus vier Teilen bestehend betrachtet werden, wie in Abbildung 4.21 (b) gezeigt.

Regel 2) **Zwei Zustände der Zelle.** Die Zustände der Zelle wird in zwei Typen abstrahiert: Einer ist leer, was bedeutet, dass der Zellinhalt *„(empty_cell)"* ist. Im Gegensatz dazu ist der andere Zustand voll.

Regel 3) **Zwei Typen der Tabelle.** Wie bereits in Abbildung 4.21 erwähnt, werden Tabellen in einfache Tabellen und komplexe Tabellen eingeteilt. Eine einfache Tabelle bedeutet, dass der Kopf nur eine Zeile und der Zeilenkopf nur eine Spalte hat, während die Kopf oder Zeilenkopf der komplexen Tabelle aus mehreren Zeilen oder Spalten besteht.

Regel 4) **Zwei Zustände der Zeile und Spalte.** Wenn eine Zeile oder Spalte mindestens eine leere Zelle enthält, wird der Zustand davon als leer definiert, andernfalls als voll.

Regel 5) **Bestimmte Transformation.** Eine komplexe Tabelle kann unbedingt nach einigen Verarbeitungen in einer einfachen Tabelle transformiert werden. Das bedeutet auch, dass nach der Transformation jede Spalte durch einen eindeutigen Kopf und jede Zeile durch einen eindeutigen Zeilenkopf definiert wird.

Regel 6) **Tendenz des Zustands von Zeilen und Spalten.** Bei den obigen Abschnitten 4.3.1, 4.3.2, 4.3.3 zur der Tabellenstrukturanalyse wird die Generierung der Positionlabeln auf die obere linke Ecke jeder Zelle basiert und gleichzeitig oben links der Ursprung des Koordinatensystems im Bild definiert. Dadurch werden die versetzten Zellen nach oben links korrigiert und hat der Zustand der oberen Zeile und der linken Spalte eine größere Tendenz, voll zu werden.

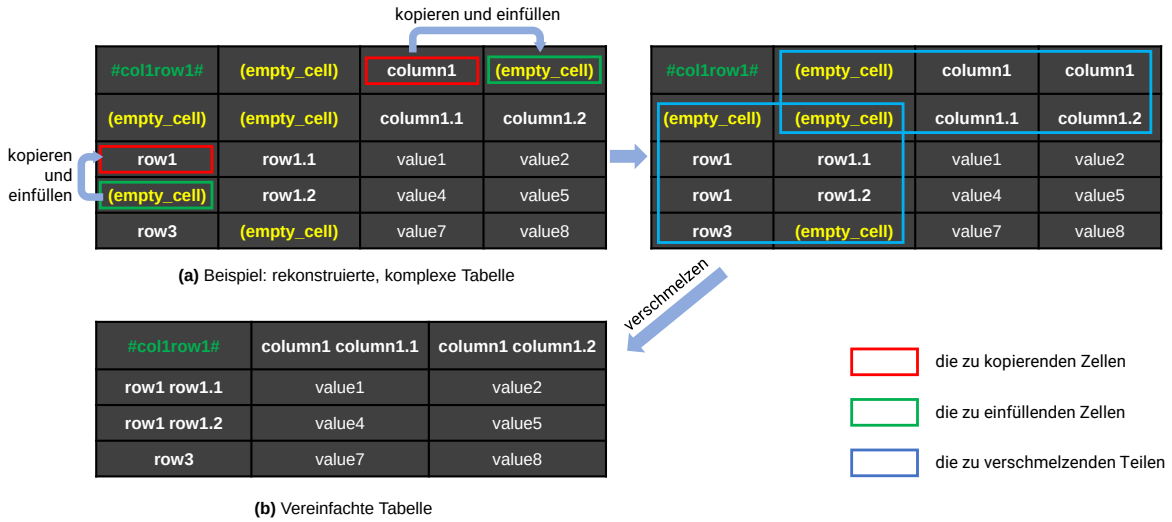


Abbildung 4.22. Visualisierung der achten Regeln und die Schmelzung.

Regel 7) **Die obere linke Zelle ist immer voll.** Um den nachfolgenden Analyseprozess in der Arbeit zu vereinfachen, sollte der Zustand der obersten linken Zelle immer als voll angenommen werden. Sie wird mit künstlich definiertem Inhalt („*#col1row1#*“) gefüllt, um ihren Zustand in voll umzuwandeln, wenn sie leer ist.

Regel 8) **Vollständigkeit von Köpfen und Zeilenköpfen.** Jede Spalte sollte ein eindeutiger, entsprechender Kopf haben, und jede Zeile sollte ein eindeutiger, entsprechender Zeilenkopf haben, d. h. es sollten keine leeren Zellen in den Köpfen sowie den Zeilenköpfen von der vereinfachte Tabelle, die aus komplexe Tabelle transformiert wird, vorhanden sein.

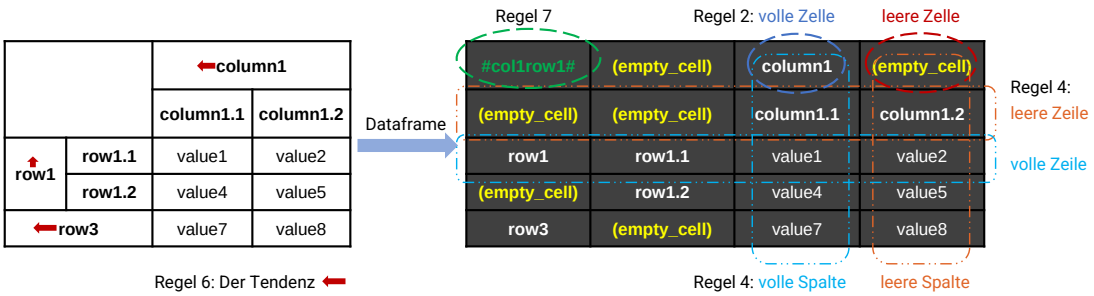


Abbildung 4.23. Visualisierung der Regeln.

Darüber hinaus werden zur Vereinfachung des Nachverarbeitungsprozesses die folgenden Annahmen verwendet. Es sollte betont werden, dass diese drei Annahmen nicht genug robust

sind. Nicht alle Tabellenvarianten können gemäß diesen beiden Annahmen verarbeitet werden. Sie sind nur als ein Versuch nach der Lösung zur Analyse und Rekonstruktion von meisten komplexen Tabellen eingesetzt.

Annahme 1) **Im Values-Teil gibt es keine leere Zelle.**

Annahme 2) **In der rekonstruierten Tabelle überschreitet die Anzahl der Zeilen im Kopf nicht die Hälfte der Gesamtzahl der Zeilen in der Tabelle, ist aber mindestens zwei.**

Annahme 3) **Der Zeilenkopfbereich in einer rekonstruierten, komplexen Tabelle besteht aus bis zu zwei Spalten.**

Schließlich ist es auch notwendig, die Ursachen komplexer Tabelle zu analysieren, um den konkreten Prozess der Nachbearbeitung zu klären. Die folgenden zwei Gründe (in Abb. 4.24 und 4.25) führen zu komplexen Tabellenköpfen oder Zeilenköpfen:

Grund 1) **Zu lange Text in einer Zelle.** Um die Daten schön und kompakt darzustellen, sind die Spalten der Tabelle normalerweise nicht zu breit. Das führt dazu, dass langer Text umgebrochen werden muss, damit er in die Zelle passt. Wenn die Tabellenstruktur basierend auf Textblöcken analysiert wird, wird jede Zelle, die darin Zeilenumbruch erzeugt wird, meistens in unabhängige Zellen unterteilt, wie in Abbildung 4.24 (a) gezeigt. Dadurch entstehen komplexer Tabellenkopfbereich oder Zeilenkopfbereich. Diese Art von komplexer Tabelle wird bei der Tabellenstrukturanalyse basieren auf Linienrekonstruktion selten erzeugt (Abb. 4.24 (b)) und tritt nur auf, wenn alle Zellen in derselben Zeile Zeilenumbrüche aufweisen (Abb. 4.24 (c)).

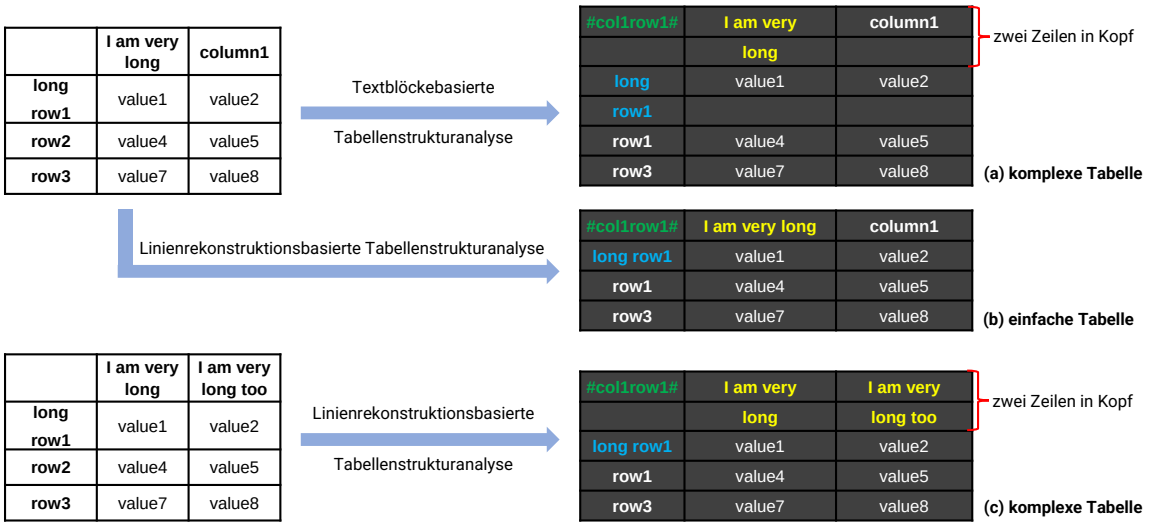


Abbildung 4.24. Komplexe Tabelle wegen zu langen Text.

Grund 2) **Mehrstufigen Köpfe.** Strukturierte Köpfe mit mehreren Ebenen wie in der Abbildung 4.25 (a) sind oft besser geeignet, um Datenbeziehungen auszudrücken. Diese mehrstufigen Köpfe lassen sich gut von Menschen lesen, können aber von einem Computer ohne Weiteres nicht verstanden werden. Da primäre Tabellenköpfe häufig Spalten überspannen, verschieben sie anhand Regel 6 in der rekonstruierten Tabelle nach links. Dies führt dazu, dass die erste Zelle in der rechten Spalte leer ist. Dieses Phänomen ist in Abbildung 4.25 (b) dargestellt, wobei das rote Kästchen die Kopfzeile der ersten Ebene markiert, das grüne Kästchen die Kopfzeile der zweiten Ebene und das blaue Kästchen die leeren Zellen, die auf der Verschiebung zurückführen. Eine ähnliche Situation ist im Zeilenkopfbereich zu finden. Diese komplexe Tabelle tritt nur auf, wenn textblöckebasierte Tabellenanalyse verwendet werden. Bei der Verwendung von linienrekonstruktionsbasierter Analyse wird eine einfache Tabelle mit falschen Verschmelzungen (Abb. 4.25 (c)) erhaltet.

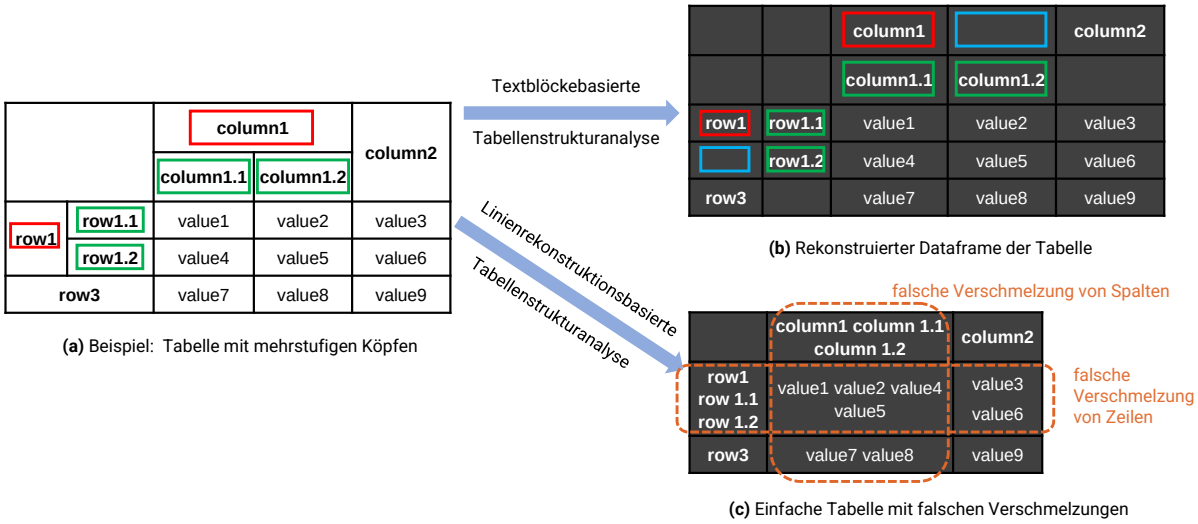


Abbildung 4.25. Komplexe Tabelle wegen mehrstufigen Köpfe.

Unter der Prämisse der obigen acht Regeln und der drei Annahmen wird gemäß der Analyse der Gründe für die Bildung der komplexen Tabellen der folgende Umsetzungsprozess formuliert. Der ganze Programmablaufplan der Nachverarbeitung wird in der Abbildung 4.34 dargestellt.

1) **Bestimmung des Tabellentyps.** Der Zweck der Nachbearbeitung besteht darin, eine komplexe Tabelle in eine einfache Tabelle umzuwandeln, daher sollte zuerst beurteilt werden, ob die zu verarbeitende Tabelle eine komplexe Tabelle oder eine einfache Tabelle ist. Gemäß Annahmen 1 und 2 sowie Regel 7 enthält eine einfache Tabelle keine leeren Zellen in der oberen Hälfte des Values-Teils. Leere Zellen werden nur durch komplexe Köpfe oder Zeilenköpfe erzeugt. Daher kann der Tabellentyp bestimmt werden,

indem abgefragt wird, ob in der oberen Hälfte der rekonstruierten Tabelle leere Zellen vorhanden sind. Die nachfolgenden Verarbeitungen werden nur durchgeführt, wenn die Tabelle eine komplexe Tabelle ist. Bei einfachen Tabellen wird Schritt 11 direkt ausgeführt.

2) **Vertikales Schmelzen.** Der Zustand jeder Zellen in jeder Zeile wird Zeile für Zeile in der gesamten rekonstruierten Tabelle beurteilt. Wenn in der Tabellen zwei benachbarte Leerzeilen vorhanden sind, die wie ein Reißverschluss zusammenpassen können, werden sie zusammengeführt. Dies kann bei sehr komplexen Köpfen passieren, Abbildung 4.26 (a) ist ein Beispiel. Wenn außerdem in einer Zeile nur die erste Zelle voll ist, während alle anderen Zellen leer sind, kann man sagen, dass diese Zeile aufgrund des zu langen Inhalts von Zeilenkopf generiert ist. Nach Regel 7 gehört diese Leerzeile eher zur darüber liegenden Zeile, also wird diese Zeile nach oben zusammengeführt. Dieser Schritt ist in Abbildung 4.26 (b) gezeigt. In der Praxis wird angesichts der Tatsache, dass die Zeilenköpfe aufgrund ihres großen Längenunterschieds in zwei Spalten aufgeteilt werden können oder ursprünglich aus zwei Spalten bestehen, eine umsichtigere Implementierung gewählt, d. h. seit die dritte Zelle in jeder leeren Zeile zu bestimmen, ob danach alle Zellen leer sind, um zu bestimmen, ob die leere Zeile zusammengeführt werden soll. Darüber hinaus werden mehrere Fusionsrunden iterativ durchgeführt, bis solche leeren Zeilen nicht mehr in der Tabelle vorhanden sind.

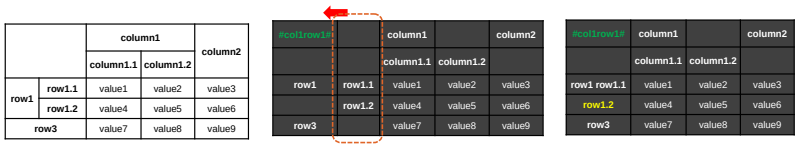
(a) Vertikales Schmelzen der Zeilen in Form von Reißverschluss

(b) Vertikales Schmelzen der mehrzeiligen Zeilenköpfe

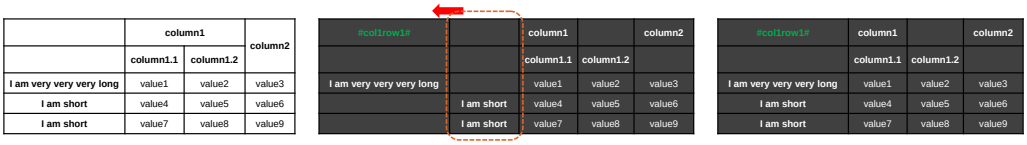
Abbildung 4.26. Visualisierung vom vertikalen Schmelzen.

3) **Horizontales Schmelzen der Zeilenköpfe.** Wie in Abbildung 4.27 gezeigt, kann der Zeilenkopfbereich in der rekonstruierten Tabelle aus zwei oder mehr Spalten bestehen. Ein möglicher Grund dafür ist, dass die Länge der Zeilenköpfe stark unterschiedlich

sind. Daneben kann auch ein anderer Grund sein, dass der Zeilenkopfbereich in dieser Tabelle zwei oder mehr Stufen enthält. In beidem Fall sollten solcher Zeilenkopfbereich, der aus mehreren Spalten besteht, zu einer einzigen Spalte zusammengeführt werden. Basierend auf Annahme 3 wird nur die horizontale Verschmelzung von Zeilenkopfbereich, die aus zwei Spalten bestehen, diskutiert. Ob der Zeilenkopfbereich aus zwei Spalten besteht, wird ermittelt, indem in den ersten beiden Spalten der unteren Hälfte der Tabelle nach leeren Zellen gesucht wird. Der Grund, nur in die untere Hälfte der Tabelle zu schauen, besteht darin, die Auswirkung des unverarbeiteten Kopfbereichs auszuschließen. Aufgrund des zweiten Schritts ist es sicher zu sagen, dass diese leeren Zellen unbedingt durch den mehrspaltigen Header erzeugt werden, wenn in der unteren Hälfte noch leere Zellen vorhanden sind. In diesem Fall wird die zweite Spalte der Tabelle nach links in die erste Spalte zusammengeführt.



(a) Rekonstruierte Tabelle mit mehrstufigen Zeilenköpfen und die Verschmelzung.



(b) Rekonstruierte Tabelle mit Zeilenköpfen mit starken unterschiedlichen Längen und die Verschmelzung.

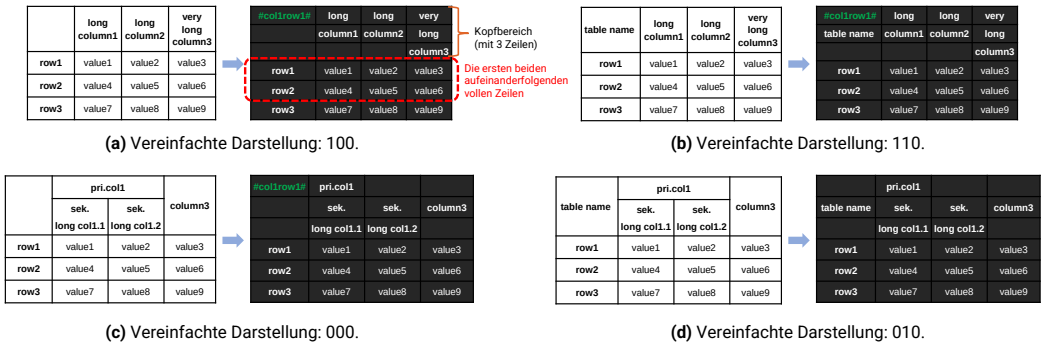
Abbildung 4.27. Horizontales Schmelzen der Zeilenköpfe.

4) **Bestimmung des Tabellentyps nochmal.** Nach der Verarbeitung des Zeilenkopfbereichs sollte beurteilt werden, ob die Tabelle zu diesem Zeitpunkt schon eine einfache Tabelle ist. Dieser Beurteilungsprozess ist derselbe wie der erste Schritt. Wenn es sich noch um eine komplexe Tabelle handelt, werden die nachfolgenden Verarbeitungen durchgeführt, andernfalls wird die Nachverarbeitung gestoppt und direkt der Schritt 11 ausgeführt.

5) **Bestimmung der Anzahl der Zeilen von Tabellenkopfbereich.** Die Behandlungen von Zeilenkopfbereich in den Schritten 2 und 3 sowie Regel 7 stellen sicher, dass die Verarbeitung von Kopfbereich nicht von anderen Faktoren als ihnen selbst beeinflusst wird. Es ist nun möglich, den komplexeren Kopfbereich zu normalisieren. Zunächst sollte die Anzahl der Zeilen im Tabellenkopfbereich bestimmt werden. Aufgrund von Annahme 2 wird nur der Fall beurteilt, in dem die Anzahl der Zeilen größer oder gleich zwei beträgt. Beginnend mit der dritten Zeile der rekonstruierten Tabelle wird Zeile für

Zeile der Zustand jeder Zeile beurteilt. Der Teil ober den ersten beiden aufeinanderfolgenden vollen Zeilen ist der Kopfbereich, was in Abb. 4.28 (a) gezeigt wird. Beispielweise haben die rekonstruierten Tabellen in Abb. 4.26 und 4.27 beide Köpfe mit zwei Zeilen. Im Vergleich dazu sind die Anzahl der Köpfe alle Tabellen in Abb. 4.28 drei.

6) **Bestimmung der Ursache für komplexen Tabellenkopfbereich.** Bei komplexen Kopfzeilen aus Grund 1 sollten einfach die Kopfzeilen mit mehreren Zeilen zu einer Zeile zusammengefasst werden, während bei komplexen Headern, die durch Grund 2 verursacht werden, zuerst nach den Köpfen der ersten Ebene in der ersten Zeile gesucht werden muss, danach werden die Köpfe der ersten Ebene kopiert und in die richtigen leeren Zellen einspeist. Anschließend lässt sich die Verschmelzung der Kopfzeilen durchführen. Als Beispiel werden übliche komplexe Kopfzeilen mit drei Zeilen, die durch die obigen zwei Gründe verursacht werden, in Abbildung 4.28 erschöpft. Darunter werden drei Zahlen verwendet, um die Darstellung von dem Zustand der Kopfzeile zu vereinfachen: Beispielsweise stellen in 010 (Abb. 4.28 (d)) die Zahlen von links nach rechts den Zustand der ersten, zweiten und dritten Zeile dar, wobei 1 für voll und 0 für leer. Es ist ersichtlich, dass die erste Zeile im komplexen Kopfbereich, der durch die mehrstufigen Köpfe erzeugt wird, leer sein muss. Wenn also der Zustand der ersten Zeile in der rekonstruierten Tabelle leer ist, soll die Duplizierung der primären Köpfe zu diesem Zeitpunkt durchgeführt werden. Wenn anderenfalls der Zustand der ersten Zeile voll ist, werden Schritte 7, 8 und 9 übergesprungen und wird direkt Schritt 10 drchgeführt.



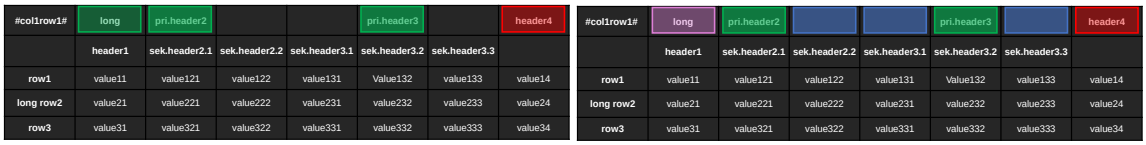
(a) Vereinfachte Darstellung: 100.

(b) Vereinfachte Darstellung: 110.

(c) Vereinfachte Darstellung: 000.

(d) Vereinfachte Darstellung: 010.

Abbildung 4.28. Übliche komplexe Köpfe mit drei Zeilen.



(a) Vorläufige Eignungsprüfung auf Beispieltabelle.

(b) Endgültige Eignungsprüfung auf Beispieltabelle.

Abbildung 4.29. Visualisierung der vorläufigen und endgültigen Eignungsprüfung.

7) **Vorläufige Eignungsprüfung.** Nicht alle Köpfe in der ersten Zeile können kopiert werden. Die Eignung jedes Kopfes in der ersten Zeile wird vorläufig durch die Zustände aller Zellen bestimmt, die sich darunter in der gleichen Spalte im Kopfbereich befinden. Wenn darunter nur leere Zellen sind, hat dieser Kopf keinen untergeordneten Kopf und sollte nicht kopiert werden. Solange es andererseits darunter Zellen gibt, deren Zustand voll ist, wird der Kopf temporär qualifiziert. Nun wird die Eignungen der Tabelle in der Abbildung 4.26 wie Abbildung 4.29 (a) gezeigt, wobei grüne und rote Kästchen qualifizierte und unqualifizierte Zellen markieren. Die erste Spalte nimmt nicht an der Eignungsprüfung teil.

8) **Endgültige Eignungsprüfung.** Mehrzeilige Kopfzeilen wie die Zelle *„long“* in Abb. 4.29 (a), die durch zu langen Text verursacht werden, sollten anhand der folgenden Analyse weiter herausgefiltert werden: Die Köpfe in die erste Zeile, die kopiert werden können, sind die primären Köpfe und steuern mindestens zwei Köpfe der unteren Ebene. Wenn die Anzahl der untergeordneten Köpfe ungerade ist, sind nach der Positionskorrektur in der linken und rechten Seite von dem primären Kopf leere Zellen vorhanden. Wenn die Anzahl eine gerade Zahl ist, ist dieser primäre Kopf spaltenübergreifend und wird nach der Positionskorrektur in die linke Spalte verschoben, sodass rechts leere Zellen vorhanden sind. Dies basiert auf der zentrierten Ausrichtung der Zellen im Kopf. Im Allgemeinen muss sich daher rechts neben einem kopierbaren Kopf eine leere Zelle befinden. Die Köpfe wie die Zelle *„long“* in Abb. 4.29 (a), die aus zu langem Text generiert wurden, erstrecken sich im Gegensatz dazu nicht über Zeilen, sodass rechts davon keine leeren Zellen vorhanden sind. Die beiden können also auf dieser Grundlage ungenau unterschieden werden. In Abbildung 4.29 (b) bleiben nun die endlich qualifizierten Köpfe mit Eignungen grün, während die Köpfe, die nicht mehr verarbeitet werden können, mit violetten Kästchen markiert werden. Zusätzlich werden alle leeren Zellen mit blauen Kästchen markiert.

9) **Diffusion der qualifizierten primären Köpfe.** Nun wird der zweiseitige Diffusionsprozess der geeigneten Tabellenköpfe durch einen iterativen Schleifenprozess erreicht: Die erste Schleifenrunde wird zuerst durchgeführt: Jeder diffusible Kopf wird kopiert und in die leere Zelle rechts davon gefüllt. Danach wird bestimmt, ob die Zelle links davon auch leer ist. Wenn die linke Zelle nicht leer ist, ist der Header deaktiviert, d. h. für die Diffusion disqualifiziert. Dieses Inaktivierungskriterium basiert auf der zentrierten Ausrichtung, die in Schritt 8 analysiert wird. Wenn die linke Zelle leer ist, wird dieser Kopf kopiert und in die linke Zelle gefüllt und bleibt qualifiziert. An diesem Punkt ist die erste Zirkulationsrunde beendet. Wenn es nach der ersten Zirkulationsrunde noch qualifizierte Köpfe gibt, werden weitere Zirkulationsrunden gestartet. In den folgenden Schleifen ist die Verarbeitung genauso wie in der ersten Schleife, außer dass statt der Zellen links und rechts die N-te Zelle links und rechts berücksichtigt werden muss: N ist die aktuelle Anzahl der Schleifen. Die Köpfe werden iterativ durch den obigen Schlei-

fenprozess (wie in Abb. 4.30) verarbeitet, bis es keinen qualifizierten Kopf mehr gibt. An diesem Punkt endet die Verarbeitung des Kopfbereichs. Es ist erwähnenswert, dass die leeren Zellen nicht in zwei Kategorien unterteilt sind: diejenigen, die gefüllt werden können, und diejenigen, die nicht gefüllt werden können. Die nicht füllbaren leeren Zellen wie die Zelle über Kopf 2 in Abbildung 4.31 werden am Ende nicht gefüllt, daher erübrigt sich die Diskussion leerer Zellen.

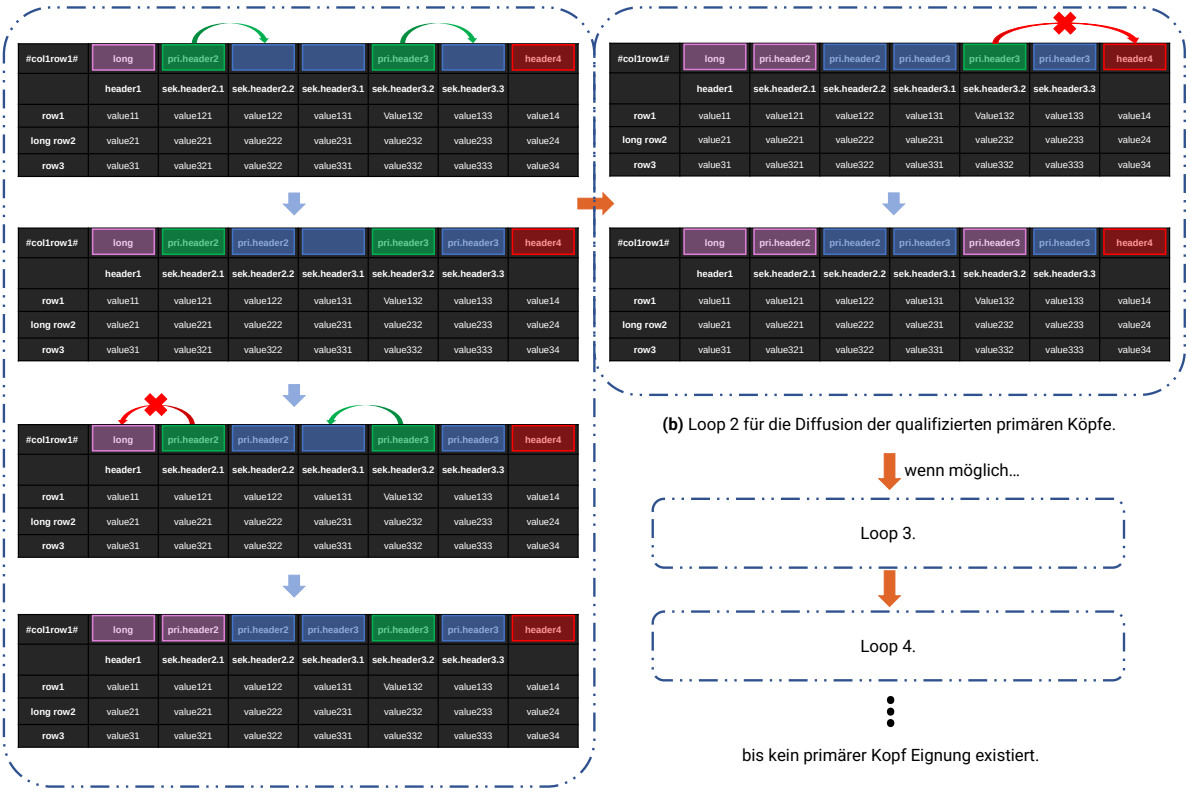


Abbildung 4.30. Loops für die Diffusion der qualifizierten primären Köpfe.

10) **Vertikales Schmelzen der Tabellenköpfe.** In diesem Schritt wird der mehrzeilige Kopfbereich zu einer Zeile zusammengeführt. Jetzt sind komplexe Tabellen in einfache Tabellen wie in Abb. 4.32 umgewandelt.

11) **Einschreibung in Datenbank.** Tabelle wird zeilenweise in JSON-Objekte konvertiert und in die Datenbank geschrieben. Die in dieser Arbeit verwendete Datenbank ist Elasticsearch Datenbanken. Auch andere Datenbanken können aber hier einsetzt werden. Als Beispiel wird das JSON-Objekt der Abb. 4.32 (a) in Abb. 4.33 dargestellt.

#col1row1#		pri.header1			
	sek.	sek.	sek.	sek.	header2
	header1.1	header1.2	header1.3	header1.4	
row1	value111	Value112	value113	value114	value12
long row2	value211	value212	value213	value214	value22
row3	value311	value312	value313	value314	value32

#col1row1#	pri.header1	pri.header1	pri.header1		
	sek.	sek.	sek.	sek.	header2
	header1.1	header1.2	header1.3	header1.4	
row1	value111	Value112	value113	value114	value12
long row2	value211	value212	value213	value214	value22
row3	value311	value312	value313	value314	value32

#col1row1#		pri.header1	pri.header1		
	sek.	sek.	sek.	sek.	header2
	header1.1	header1.2	header1.3	header1.4	
row1	value111	Value112	value113	value114	value12
long row2	value211	value212	value213	value214	value22
row3	value311	value312	value313	value314	value32

#col1row1#	pri.header1	pri.header1	pri.header1	pri.header1	
	sek.	sek.	sek.	sek.	header2
	header1.1	header1.2	header1.3	header1.4	
row1	value111	Value112	value113	value114	value12
long row2	value211	value212	value213	value214	value22
row3	value311	value312	value313	value314	value32

(a) Loop 1 für die Diffusion der qualifizierten primären Köpfe. **(b)** Loop 2 für die Diffusion der qualifizierten primären Köpfe.

Abbildung 4.31. Beispieltabelle, in der die leeren Zellen nicht alle ausgefüllt werden.

#col1row1#	pri.header1	pri.header1	pri.header1	pri.header1	
	sek.	sek.	sek.	sek.	header2
	header1.1	header1.2	header1.3	header1.4	
row1	value111	Value112	value113	value114	value12
long row2	value211	value212	value213	value214	value22
row3	value311	value312	value313	value314	value32

Verschmelzung

#col1row1#	pri.header1 sek. header1.1	pri.header1 sek. header1.2	pri.header1 sek. header1.3	pri.header1 sek. header1.4	header2
row1	value111	Value112	value113	value114	value12
long row2	value211	value212	value213	value214	value22
row3	value311	value312	value313	value314	value32

(a) Eine Tabelle mit 3-zeiligen Köpfen und ihre Verschmelzung.

#col1row1#	long	pri.header2	pri.header2	pri.header3	pri.header3	pri.header3	header4
	header1	sek.header2.1	sek.header2.2	sek.header3.1	sek.header3.2	sek.header3.3	
row1	value11	value121	value122	value131	Value132	value133	value14
long row2	value21	value221	value222	value231	value232	value233	value24
row3	value31	value321	value322	value331	value332	value333	value34

Verschmelzung

#col1row1#	Long header1	pri.header2 sek.header2.1	pri.header2 sek.header2.2	pri.header3 sek.header3.1	pri.header3 sek.header3.2	pri.header3 sek.header3.3	header4
row1	value11	value121	value122	value131	Value132	value133	value14
long row2	value21	value221	value222	value231	value232	value233	value24
row3	value31	value321	value322	value331	value332	value333	value34

(b) Eine Tabelle mit 2-zeiligen Köpfen und ihre Verschmelzung.

Abbildung 4.32. Vertikales Schmelzen der Tabellenköpfe.

#col1row1#	pri.header1 sek. header1.1	pri.header1 sek. header1.2	pri.header1 sek. header1.3	pri.header1 sek. header1.4	header2
row1	value111	Value112	value113	value114	value12
long row2	value211	value212	value213	value214	value22
row3	value311	value312	value313	value314	value32


```
1  [
2      {
3          "#colirow1#": "row1",
4          "pri.header1_sek._header1.1": "value111",
5          "pri.header1_sek._header1.2": "value112",
6          "pri.header1_sek._header1.3": "value113",
7          "pri.header1_sek._header1.4": "value114",
8          "header2": "value12",
9          "uniqueId": "table_2_of_json.jpg",
10         "fileName": "json.jpg"
11     },
12     {
13         "#colirow1#": "long row2",
14         "pri.header1_sek._header1.1": "value211",
15         "pri.header1_sek._header1.2": "value212",
16         "pri.header1_sek._header1.3": "value213",
17         "pri.header1_sek._header1.4": "value214",
18         "header2": "value22",
19         "uniqueId": "table_2_of_json.jpg",
20         "fileName": "json.jpg"
21     },
22     {
23         "#colirow1#": "row3",
24         "pri.header1_sek._header1.1": "value311",
25         "pri.header1_sek._header1.2": "value312",
26         "pri.header1_sek._header1.3": "value313",
27         "pri.header1_sek._header1.4": "value314",
28         "header2": "value32",
29         "uniqueId": "table_2_of_json.jpg",
30         "fileName": "json.jpg"
31     }
32 ]
```


Abbildung 4.33. JSON-Objekt der Abb. 4.32 (a).

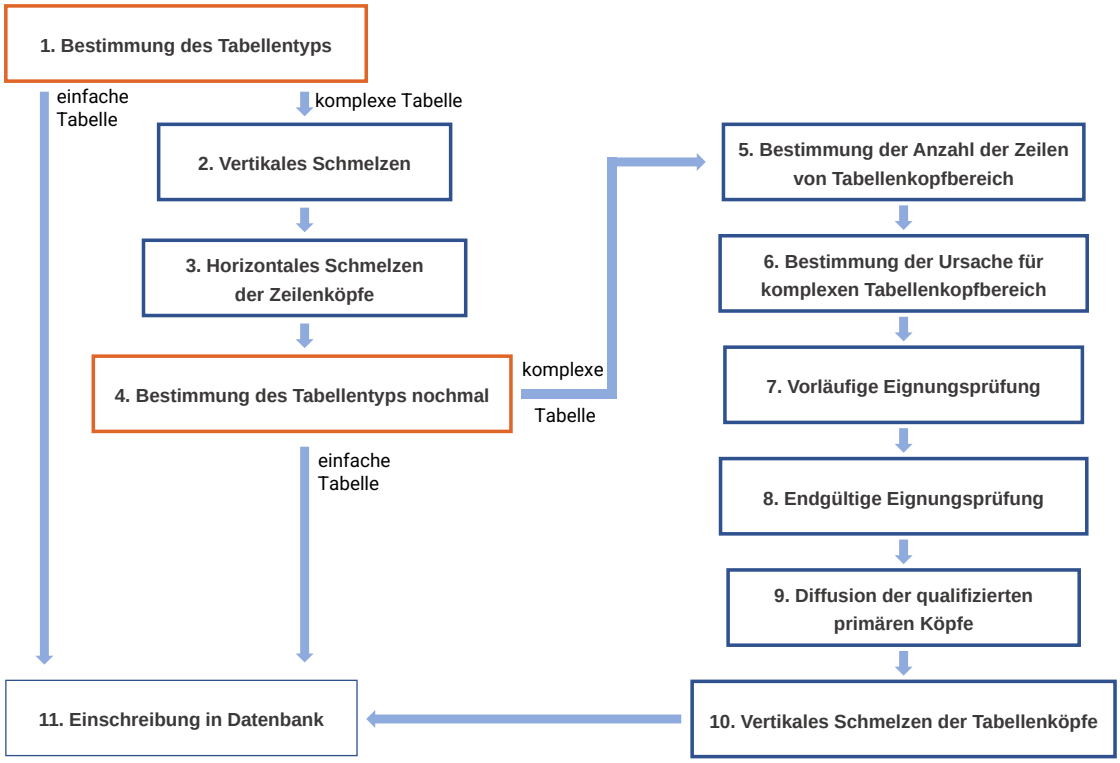


Abbildung 4.34. Programmablaufplan der Nachverarbeitung.

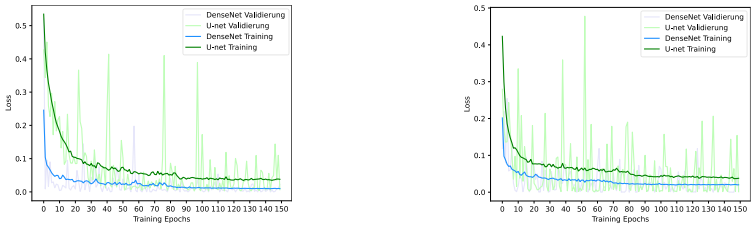
5. Ergebnisse und Auswertung

In diesem Kapitel wird zunächst Mean Intersection Over Union (mIoU)[Tiu19] an den 85 Bildern im Testsatz berechnet, um die Leistung der Modelle mit Backbone DenseNet-121-BC und U-net (im Weiteren bezeichnet jeweils als das *DenseNet* Modell und *U-net* Modell) in Bezug auf die Erkennung von Tabellen- und Spaltenbereichen zu bewerten. Die drei Verfahren der Tabellenbereichserkennung nach gleichen Vorverarbeitungen werden dann im Abschnitt 5.2 verglichen. Abschließend befinden sich die Vergleichungen zwischen den drei Verfahren der Tabellenstrukturanalyse sowie die Auswertungen, wofür zehn zufällige Tabellen aus dem Testset und gleiches Verfahren zur Tabellenrekonstruktion und Nachverarbeitung genutzt werden.

5.1. Auswertung von DenseNet Modell und U-net Modell

Vier neuronale Netze wurden trainiert, wobei ein *DenseNet* Modell und ein *U-net* Modell zur Tabellenbereichserkennung sowie ein *DenseNet* Modell und ein *U-net* Modell zur Spaltenbereichserkennung. Wie im Abschnitt 4.2.3 genannt, wird nach dem Training das Modell mit wenigstem Verlust basierend auf der Visualisierung des Verlusts im Trainings- und Validierungssatz ausgewählt. Die Abbildung 5.1 zeigt Verlustkurven der beiden Modelle in den Tasks Tabellenbereichserkennung und Spaltenbereichserkennung während des Trainings. Es ist ersichtlich, dass die Verlustkurven der Modelle auf dem Trainingssatz mit zunehmenden Epochen insgesamt einen Abwärtstrend zeigen. Darüber hinaus bleiben die Training-Verluste in Abb. 5.1 (a) nach dem 80-ten Epoch fast konstant. Die ähnlichen Situationen treten auf den Kurven der Training-Verluste nach 70-ten Epoch von *DenseNet* Modell und nach dem 90-ten Epoch von *U-net* Modell in Abb. 5.1 (b). Der Validierungssatz ist nicht am Training beteiligt, sondern wird nur zur Parameterabstimmung verwendet, sodass der Verlust darauf die Verallgemeinerungsfähigkeit des Modells bis zu einem gewissen Grad widerspiegelt[bae22]. Auf dem Validierungssatz sinken die Verluste der beiden Modelle mit starken Schwankungen. Dies führt vielleicht darauf zurück, dass die Datenmenge im Validierungssatz gering ist. Andererseits würde dies auch durch Überanpassung verursacht[bae22].
Da die Verlustkurve auf dem in diesem Training erhaltenen Verifikationssatz mit starken Schwankungen nicht repräsentativ ist, werden die Modelle mit möglichst kleinen Epochs und Verlust ausgewählt, um Überanpassung zu vermeiden. Dementsprechend werden das nach 90-ten Training-Epoch erhaltene *DenseNet* Modell und das nach 85-ten Training-Epoch erhaltene

U-net Modell zur Tabellenbereichserkennung als die besten ausgewählt, sowie das nach 70-ten Training-Epoch erhaltene *DenseNet* Modell und das nach 90-ten Training-Epoch erhaltene *U-net* Modell zur Spaltenbereichserkennung.



(a) Verlustkurven bei Tabellenbereichserkennung. **(b)** Verlustkurven bei Spaltenbereichserkennung.

Abbildung 5.1. Verlustkurven der Modelle im Training.

Um den Segmentierungseffekt der besten Segmentierungsmodelle auf diesen experimentellen Testsatz zu beurteilen, werden Mean Intersection Over Union (mIoU) als Bewertungsindikator verwendet. Intersection Over Union (IoU) ist die Überlappung zwischen der vorhergesagten Segmentierung und der Ground-Truth dividiert durch die Union zwischen der vorhergesagten Segmentierung und der Ground-Truth, wie Abbildung 5.2 gezeigt. Für die binäre Segmentierung in dieser Arbeit wird die mittlere IoU des Bildes berechnet, indem die IoU-Score jeder Klasse genommen und gemittelt wird. Das heißt, dass in den Modellen zur Tabellenbereichserkennung die mittlere IoU-Score von Tabellenbereich und Hintergrund berechnet wird, während in den Modellen zur Spaltenbereichserkennung die mittlere IoU von Spaltenbereich und Hintergrund.

$$IoU = \frac{Ueberlappung}{Union} =$$

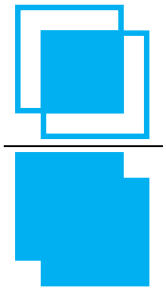


Abbildung 5.2. Visualisierung der Berechnung von Intersection Over Union.

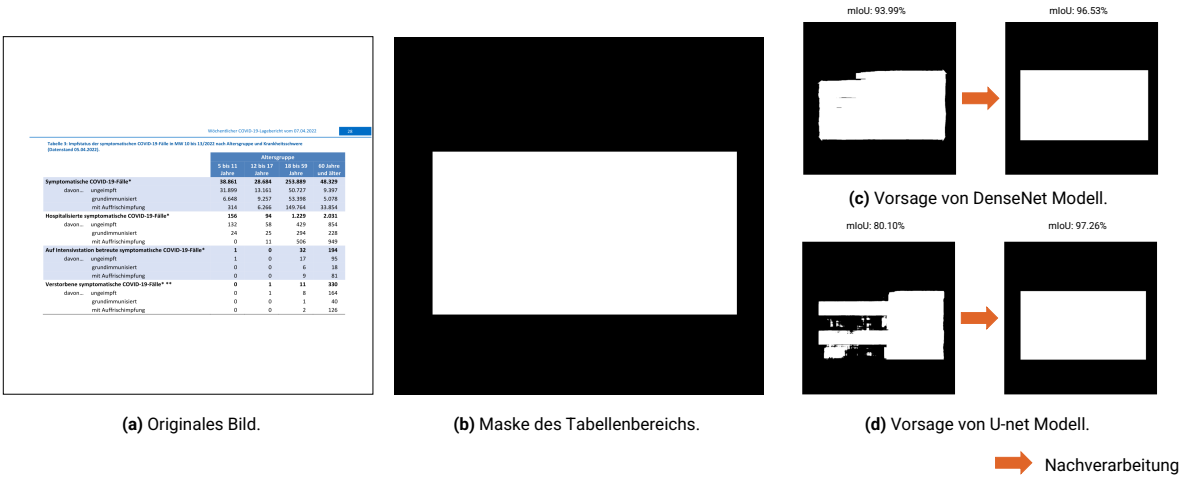
Der zur Berechnung von mIoU verwendete Datensatz besteht aus 85 Bildern, die aus den Covid-19-Wochenberichten des RKI[Ins22] im Januar 2022 kamen. In der Tabelle 5.1 werden die mIoU-Scores des *DenseNet-121-BC* Modells und *U-net* Modells für die Tabellenbereichs- und Spaltenbereichserkennung aufgelistet. Die Vorhersagen der Modelle werden in der Implementierung nicht direkt zur Bestimmung des Tabellenbereichs und Spaltenbereichs verwendet. Stattdessen werden weiter darauf Nachbearbeitungen durchgeführt, die bei Tabellenbereichserkennung (im Abschnitt 4.2.3) aus dem morphologischen Opening und Regularisierung mit

Tabelle 5.1. Mean Intersection Over Union (mIoU) des *DenseNet-121-BC* und *U-net* Modells bei Tabellenbereichserkennung und Spaltenbereichserkennung auf 85 Bildern.

Backbone des Modells	Tabellenbereichserkennung	Spaltenbereichserkennung
DenseNet-121-BC	95.51%, 98.91%*	92.36%, 92.31%*
U-net	86.81%, 95.85%*	81.83%, 90.59%*

[*] mIoUs der Vorhersagen, die weiter nach entsprechenden Nachbearbeitung in bestimmten Aufgaben unterzogen werden.

der *cv2.findcontour()* Funktion sowie bei Spaltenbereichserkennung (im Abschnitt 4.2.3) aus dem morphologischen Opening bestehen. Daher befinden sich in der Tabelle 5.1 zum Vergleich noch die mIoU-Scores der Vorhersagen, die weiter nach entsprechenden Nachverarbeitung in bestimmten Aufgaben unterzogen werden. Die meisten Scores werden erheblich verbessert.
Es ist ersichtlich, dass das *DenseNet* Modell mit den besten Leistungen zur Tabellenbereichssegmentierung und Sapltenbereichssegmentierung dominiert, weshalb es in den Schritten 4.2.3 und 4.3.3 der Implementierung eingesetzt werden.



(a) Originales Bild. (b) Maske des Tabellenbereichs. (c) Vorsage von DenseNet Modell. (d) Vorsage von U-net Modell.

Abbildung 5.3. Vergleich der Vorhersagen der Modelle und Darstellung der Nachverarbeitung.

Der Grund für den Leistungsunterschied könnte zum einen in der Modelltiefe liegen. Das *Densenet* Modell hat eine tiefere Modellstruktur als das *U-net* Modell, sodass es mehr Features erfassen kann. Die tiefere Modellstruktur bringt außerdem ein breiteres, rezeptives Feld (RF)[Ada20]. Dadurch konzentriert sich das Modell mehr auf der ganzen Tabelle. Im Gegensatz dazu sind die vom *U-net* Modell vorhergesagten Tabellenbereiche manchmal unvollstän-

dig. Dieses Aufspaltungsphänomen wird in Abbildung 5.3 gezeigt. Nach der Nachbearbeitung wird das Phänomen gemildert, was sich durch die Bewertungsverbesserung in Tabelle 5.1 widerspiegelt. Damit kann auch erklären, warum die Scores des *Densenet* Modells nach der Nachbearbeitung in der Spaltenbereichserkennung sinkt: Einige benachbarte Spalten, die korrekt unterschieden wurden, werden in der Nachbearbeitung miteinander verschmolzen.
Andererseits lädt das *Densenet* Modell zu Beginn die Modellparameter, die auf dem COCO-Datensatz vortrainiert wurden, was dazu führte, dass es vor Training bereits einige Erfahrungen gesammelt hat.

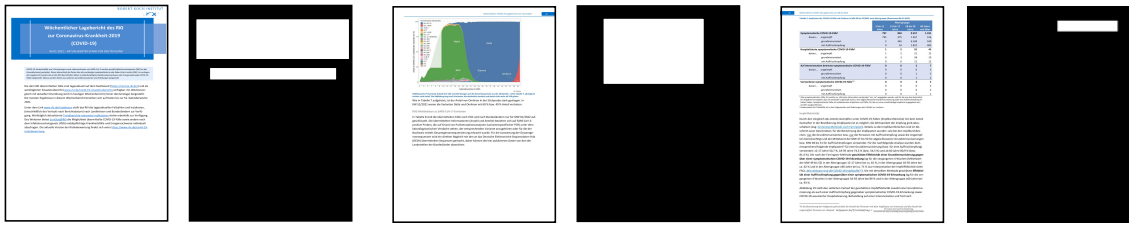
5.2. Tabellenbereichserkennung

Die Leistungen der drei Verfahren bei Tabellenbereichserkennung werden ähnlich wie im Abschnitt 5.1 mittels mIoU verglichen. In der Tabelle 5.2 werden die mIoUs, die zwischen Ground-Truth und der Vorhersage basierend auf Linienerkennung, Textblockerkennung und unter Verwendung neuronaler Netze berechnet, dargestellt. Der Zeitaufwand für jedes der drei Verfahren wird außerdem darin gezeigt.
Es ist zu sehen, dass der beste mIoU-Score vom auf KNN bzw. das *DenseNet* Modell basierten Verfahren erreicht wird. Mit 98.91% mIoU-Score übertrifft dieses Verfahren die beiden anderen deutlich, obwohl es zusätzliche ca. sechs Sekunden zur Verarbeitung jedes Bildes auf CPU kostet. Im Vergleich dazu befinden sich die Scores der beiden traditionellen Verfahren nur gegen 80%. Dies führt darauf zurück, dass die neuronalen Netze eine stärkere Robustheit bei der Erkennung verschiedener Arten von Tabellen haben als das Linienerkennungsbasierte und Textblöckebasierte Verfahren. Im Gegensatz dazu verlangt die Erkennung basierend auf Linienerkennung den vollständigen, äußeren Rahmen der Tabelle, sonst werden die Tabellenvarianten wie Abb. 4.3 (c), (d) und 5.4 (c) beim Text nicht erkennt. Darüber hinaus werden einige andere Elemente im Dokument, z. B. Überschriften (wie Abb. 5.4 (a)) und Illustrationen (wie Abb. 5.4 (b)), von dieser Methode als Tabellenbereiche erkannt.

Tabelle 5.2. Vergleich der drei Verfahren bei Tabellenbereichserkennung.

Verfahren	mIoU	Zeitaufwand (s pro Bild)
Linien-basiert	78.52%	2.11
Textblöcke-basiert	83.77%	1.73
KNN[1]-basiert	98.91%	7.62[2], 2.12[3]

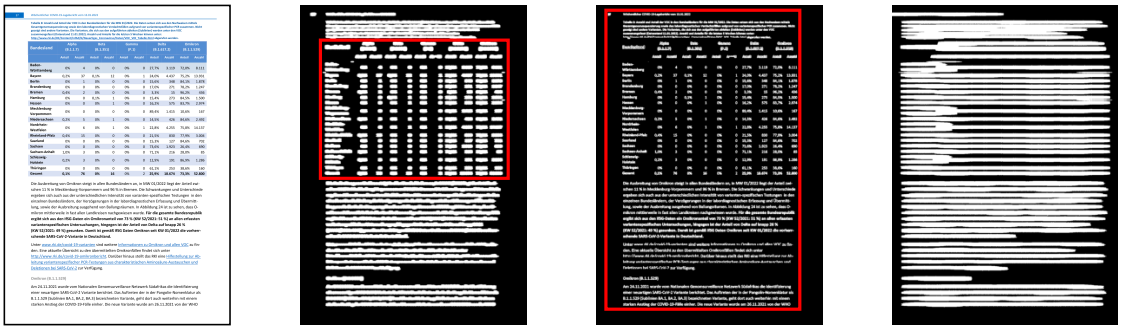
[1] Die künstlichen neuronalen Netze (KNN) sind hier das *DenseNet* Modell, das im Abschnitt 5.1 dominiert.
[2] Zeitaufwand des KNN-basierten Verfahren zur Tabellenbereichserkennung auf CPU.
[3] Zeitaufwand des KNN-basierten Verfahren zur Tabellenbereichserkennung auf GPU.



(a) Verwirrung beim Überschrift. (b) Verwirrung beim Illustration. (c) Ignorierung bei Tabellenvarianten.

Abbildung 5.4. Verwirrung in der linienbasierten Tabellenbereichserkennung.

Außerdem verlässt sich das textblöckebasierte Verfahren auf im Voraus gegebenen Kernel der Dilatation. Ein kleiner Kernel führt dazu, dass der Block der Textzeile in mehrere Textblöcke aufgeteilt wird, die aus einem einzelnen Zeichen bestehen. Wenn der Kernel jedoch zu groß ist, werden die Textblöcke, die zu zwei benachbarten Spalten innerhalb einer Tabellenzeile gehören, zusammengeführt. Beide Fälle machen Tabellenbereiche und Textbereiche voneinander ununterscheidbar (wie in Abb. 5.5). Bei verschiedenen Dokumenten kann der Kernel aufgrund unterschiedlicher Zeichenabstände, Schriftgrößen und Bildgrößen große Unterschiede aufweisen, was die Findung eines robusten Kernels beschwert.



(a) Originales Bild. (b) Dilatation mit geeigneter Größe. (c) Dilatation mit kleiner Größe. (d) Dilatation mit großer Größe.

Abbildung 5.5. Einfluss der Kerngröße auf die Dilatation.

5.3. Tabellenstrukturanalyse

Die Auswertung der drei Tabellenstrukturanalyseverfahren wird basierend auf zehn zufällige Tabellen aus dem Testset durchgeführt. Nach den drei Verfahren wird gleiches Verfahren zur Tabellenrekonstruktion und Nachverarbeitung eingesetzt. Darüber hinaus ist die bedingte Beurteilung zur Generierung der Labels in der linienrekonstruktionsbaiserten Tabellenstrukturanalyse wegen der starken Ausrichtung der Zellenkoordinaten in horizontaler und vertikaler Richtung äquivalent zu K-Means-Clustering. Daher wird K-Means-Clustering in den drei Tabellenstrukturanalyseverfahren zur Generierung der Labels eingesetzt.

Pipeline 1) *Linienrekonstruktionsbaiserte Tabellenstrukturanalyse (mit K-Means-Cluster-ing) + OCR und Tabellenrekonstruktion + Nachverarbeitung*

Pipeline 2) *Textblöckebaiserte Tabellenstrukturanalyse (mit K-Means-Clustering) + OCR und Tabellenrekonstruktion + Nachverarbeitung*

Pipeline 3) *Tabellenstrukturanalyse mit Unterstützung neuronaler Netze (und K-Means-Clustering) + OCR und Tabellenrekonstruktion + Nachverarbeitung*

Das Bewertungskriterium ist im Values-Teil, Köpfe-Teil und Zeilenköpfe-Teil die durchschnittliche Zellgenauigkeitsrate der Ergebnisse, die von den zehn Bildern nach den folgenden drei Pipelines erhalten werden. Dies wird mit Formel 5.1 berechnet, wobei *RZ* die Anzahl der Zellen mit richtigem Inhalt in jedem Teil und *N* die Anzahl der Zellen in dem Teil der originalen Tabelle ist. Die N wird daneben mit Formel 5.2 berechnet, wobei *ZN* die Anzahl der Zeilen im entsprechenden Teil und *SN* die Anzahl der Spalten ist. Die Vergleichung der drei Pipelines wird in der Tabelle 5.3 dargestellt.

$$Zellgenauigkeit = \frac{RZ}{N} \tag{5.1}$$

$$N = (ZN + 1) \times (SN + 1) \tag{5.2}$$

Tabelle 5.3. Durchschnittliche Zellgenauigkeitsrate nach drei Pipelines und der mittele Zeitaufwand.

Pipeline	Zellgenauigkeitsrate			Zeitaufwand (s pro Bild)
	Values-Teil	Köpfe-Teil	Zeilenköpfe-Teil	
Pipeline 1	41.32%	12.90%	98.73%	14.51
Pipeline 2	90.24%	67.74%	90.97%	16.74
Pipeline 3	99.18%	75.81%	90.97%	19.82[1], 17.06[2]

[1] Zeitaufwand des KNN-basierten Verfahren zur Tabellenbereichserkennung auf CPU.
[2] Zeitaufwand des KNN-basierten Verfahren zur Tabellenbereichserkennung auf GPU.

Die Pipeline 3 bzw. die Tabellenstrukturanalyse mit Unterstützung neuronaler Netze zeigt die höchste Zellgenauigkeit im Values- und Köpfe-Teil. Der Schlüssel zum Gewinnen ist die Aufteilung mehrerer Spalten unter den spaltenübergreifenden Köpfen, wie die Abbildung 5.6 (a). Mithilfe der richtigen Masken (wie 4.7 (c)) erlernen die neuronalen Netze, wie sie die Spalten unter den spaltenübergreifenden Köpfen voneinander unterscheiden. Im Gegensatz dazu werden diese Spalten nach der linienrekonstruktionsbaiserten Tabellenstrukturanalyse (Pipeline 1) als eine Spalte erkennen. Dies wird in Abb. 5.6 (b) ausgesehen, wobei die zweite

und dritte, vierte und fünfte, sowie sechste und siebte Spalte miteinander verschmelzt sind. Außerdem sinkt zwar die Genauigkeitsrate im Values- und Köpfe-Teil bei Verwendung von Pipeline 2 zur Verarbeitung der Tabellenstruktur leicht. Dies führt darauf zurück, dass in der rekonstruierten Tabelle die linke Spalte (die rote Linie in Abb. 5.7 (a)) der beiden benachbarten Spalten durch die rechte Spalte (die grüne Linie in Abb. 5.7 (a)) überschrieben wird, wodurch ein Teil der Köpfe und ein Teil der Values verschwinden.
Ähnliche Überschreibung tritt auch in der Zeilen auf, wenn K-Means-Clustering in Pipeline 2 und 3 eingesetzt wird. Im Gegensatz dazu gibt es keine Überschreibung in der Pipeline 1, sondern die Verschmelzung der zwei Spalten. Es wird in Abb. 5.6 (b) und 5.7 (c) miteinander verglichen. In Abbildung 5.7 (c) befindet sich die blaue Zelle, die mittels Pipeline 1 rekonstruiert wird. Es ist zu erkennen, dass der in zwei Spalten aufgeteilte Zeilenanfang korrekt rekonstruiert wird, so dass die erste Pipeline bei der Rekonstruktion des Zeilenanfangs eine Zellgenauigkeit von 98,73% erreicht.

Bundesland	Meldewoche 51		Meldewoche 52		Änderung im Vergleich	
	Anzahl	7-Tage-Inzidenz	Anzahl	7-Tage-Inzidenz	Anzahl	Anteil
Baden-Württemberg	25.388	229	27.579	248	2.191	+9%
Bayern	25.940	197	26.114	199	174	+1%
Berlin	11.559	315	11.387	311	-172	-1%
Brandenburg	10.145	401	9.332	369	-813	-8%
Bremen	1.856	273	3.235	476	1.379	+74%
Hamburg	6.446	348	8.602	464	2.156	+33%
Hessen	11.090	176	14.155	225	3.065	+28%
Mecklenburg-Vorpommern	4.838	300	4.464	277	-374	-8%
Niedersachsen	10.715	134	14.684	183	3.969	+37%
Nordrhein-Westfalen	34.460	192	39.635	221	5.175	+15%
Rheinland-Pfalz	6.311	154	7.271	177	960	+15%
Saarland	1.796	183	2.087	212	291	+16%
Sachsen	18.236	450	13.861	342	-4.375	-24%
Sachsen-Anhalt	9.076	416	6.477	297	-2.599	-29%
Schleswig-Holstein	4.295	148	8.421	289	4.126	+96%
Thüringen	11.780	556	8.653	408	-3.127	-27%
Gesamt	**193.931**	**233**	**205.957**	**248**	**12.026**	**+6%**

(a) Originale Tabelle.

Bundesland	Meldewoche 51 Anzahl 7 Tage Inzidenz	Meldewoche 52 Anzahl 7 Tage Inzidenz	Änderung im Vergleich Anzahl Anteil
Baden-Württemberg	25.388 229	27.579 248	2.191 +9%
Bayern	25.940 197	26.114 199	174 +1%
Berlin	11.559 315	11.387 311	172 -1%
Brandenburg	10.145 401	9.332 369	-813 '8%
Bremen	1.856 273	3.235 476	1.379 +74%
Hamburg	6.446 348	8.602 464	2.156 +33%
Hessen	11.090 176	14.155 225	3.065 +28%
Mecklenburg-Vorpommern	4.838 300	4.464 277	-374 -8%
Niedersachsen	10.715 134	14,684 183	3.969 +37%
Nordrhein-Westfalen	34.460 192	39.635 211	5.175 +15%
Rheinland-Pfalz	6.311 154	7,271 177	960 +15%
Saarland	1.796 183	2.087 212	291 +16%
Sachsen	18.236 450	13.861 342	-4.375 -24%
Sachsen-Anhalt	9.076 416	6.477 297	2.599 -29%
Schleswig-Holstein	4.295 148	8.421 289	4.126 +96%
Thüringen	11.780 556	8.653 408	,3.127 -28%
Gesamt	193 931 233	205.957 248	12.026 +6%

(b) Falsche Verschmelzung der Pipeline 1.

Bundesland	Meldewoche_51_Anzahl	Meldewoche_51_7_Tage_Inzidenz	Meldewoche_52_Anzahl	Meldewoche_52_7_Tage_Inzidenz	Änderung_im_Vergleich_Anzahl	Änderung_im_Vergleich_Anteil
Baden-Württemberg	25.388	229	27.579	248	2.191	+9%
Bayern	25.940	197	26.114	199	174	+1%
Berlin	11.559	315	11.387	311	172	-1%
Brandenburg	10.145	401	9.332	369	-813	'8%
Bremen	1.856	273	3.235	476	1.379	+74%
Hamburg	6.446	348	8.602	464	2.156	+33%
Hessen	11.090	176	14.155	225	3.065	+28%
Mecklenburg-Vorpommern	4.838	300	4.464	277	-374	-8%
Niedersachsen	10.715	134	14,684	183	3.969	+37%
Nordrhein-Westfalen	34.460	192	39.635	221	5.175	+15%
Rheinland-Pfalz	6.311	154	7.271	177	960	+15%
Saarland	1.796	183	2.087	212	291	+16%
Sachsen	18.236	450	13.861	342	-4.375	-24%
Sachsen-Anhalt	9.076	416	6.477	297	2.599	-29%
Schleswig-Holstein	4.295	148	8.421	289	4.126	+96%
Thüringen	11.780	556	8,653	408	,3.127	-27%
Gesamt	193.931	233	205.957	248	12.026	+6%

(c) Ergebnis der Pipeline 2 und 3.

Abbildung 5.6. Verarbeitung der Spalten unter dem spaltenübergreifenden Kopf auf Bild *Wochenbericht_2022-01-06_6*.

Bundesland	Fälle kumulativ		
	Fälle	Differenz Vorta	Fälle/ 100.000 EW
Baden-Württemberg	4.969.952	0	44.675
Bayern	6.611.783	0	50.177
Berlin	1.403.292	0	38.159
Brandenburg	1.093.604	0	43.091
Bremen	297.629	0	43.998
Hamburg	796.043	0	42.938
Hessen	2.850.605	0	45.284
Mecklenburg-Vorpommern	698.700	0	43.366
Niedersachsen	3.751.392	0	46.734

(a) Originales Bild.

Bundesland	Fälle_kumulativ_Fälle	Fälle_kumulativ_Fälle_100.000_EW
Baden-Württemberg	4.969.952	44'675
Bayern	6.611.783	50.117
Berlin	1.403.292	38.159
Brandenburg	1.093.604	43.091
Bremen	1.093.604	43.998
Hamburg	291.629	42.938
Hessen	796.043	45.284
Vorpommern	698.700	43.366
Niedersachsen	3.751.392	46.134

(b) Überschreibung bei Pipeline 2 und 3.

Bundesland	Fälle_kumulativ_Fälle	Fälle_kumulativ_Differenz_Vortrag_Fälle_100.000_EW
Baden-Württemberg	4.969.952	0_44'675
Bayern	6.611.783	0_50.117
Berlin	1.403.292	0_38.159
Brandenburg	1.093.604	0_43.091
Bremen	1.093.604	0_43.998
Hamburg	291.629	0_42.938
Hessen	796.043	0_45.284
Mecklenburg-Vorpommern	698.700	0_43.366
Niedersachsen	3.751.392	0_46.134

(c) Richtige Erkennung und falsche Verschmelzung bei Pipeline 1.

Abbildung 5.7. Ergebnisse bei Pipeline 1, 2 und 3 auf Bild *2023-01-02-de_2*.

6. Diskussion

Im Rahmen dieser Studienarbeit wurde ein *Python*-Skript entwickelt, um Daten, die in Tabellenform auf Bild dargestellt werden, zu erfassen. Die hierzu erfolgte Literaturrecherche ergab, dass es bereits zahlreiche Forschungen und Methoden veröffentlicht, die auf traditionellen Regeln oder neuronalen Netzen für eine oder mehre Teilaufgaben dieser Arbeit basieren. Es wurde jedoch kein Verfahren gefunden, um alle Teilaufgaben zusammen nacheinander zu lösen. Die meisten von ihnen hörten bei der Tabellenstrukturanalyse auf[Yon19, LHY+21] und haben die Tabellenstruktur nicht weiter normalisiert, um in einer Datenbank einzuschreiben. Daher schlag diese Arbeit eine vollständige Methode vor, in der die Tabellenextraktion mittels Tabellenbereichserkennung, Tabellenstrukturanalyse und normalisierte Nachverarbeitung Schritt für Schritt abschließt wurde.
Laut des Vergleichs im Abschnitt 5.2 stützte sich die Tabellenbereichserkennung auf das semantische Segmentierungsmodell, das mit *DenseNet-121-BC* als Backbone eingerichtet wurde, um eine bessere Robustheit im Vergleich zum regelbasierten Basis zu erhalten. Inspiriert von TableNet[PVR+19] wurde die Tabellenstrukturanalyse mittels der dritten Pipeline aufgrund der Kombination von a priori bekannten Reglen (K-Means-Clustering) und neuronale Netze implementiert. In der Nachverarbeitung werden die Daten in der Tabelle gemäß durchdachten Regeln und Annahmen normalisiert und in einer Datenbank gespeichert, wobei insbesondere die primären Kopfzeilen kopiert und mit entsprechenden Subheaders verschmolzen wurden.
Neuronale Netze zu trainieren, um direkt aus Eingabebildern eine rekonstruierte Tabelle zu erhalten, die in eine Datenbank geschrieben werden kann, scheint ein schnellerer Ansatz zu sein als die in dieser Studienarbeit vorgeschlagene mehrstufige Methode, aber dieser End-to-End-Blackbox mangelt es an Interpretierbarkeit.
Es gibt jedoch noch Teile dieser Arbeit, die verbessert werden sollen. Der zum Training der neuronalen Netze verwendete Datensatz bestand nur aus 1155 Bildern und den entsprechenden Masken, sodass die Netze möglicherweise nicht vollständig trainiert wurden, was zu einer Verringerung der Genauigkeit führte. Aber trotzdem schienen neuronale Netze gemäß den Ergebnissen in Tabelle 5.2 und 5.3 besser geeignet für die Tabellenbereichserkennung und Tabellenstrukturanalyse. Daher wäre es ähnlich wie bei der Spaltenerkennung auch möglich, neuronale Netze für die Zeilenerkennung und Verarbeitung der Tabellenköpfe zu trainieren. Darüber hinaus könnten OCR und normalisierte Nachverarbeitung auch unter Verwendung von selbst trainierten neuronalen Netzen durchgeführt werden. Allerdings wurde in dieser Arbeit aus Zeitgründen der für das Training benötigte Datensatz nicht erstellt.

7. Zusammenfassung

Um Informationen strukturiert formulieren zu können, wird ein großer Teil der Daten in Form von Tabellen veranschaulicht und in Form von Bildern publiziert. Um diese Daten wieder zu extrahieren, müssen sie lokalisiert, rekonstruiert und in einer Datenbank gespeichert werden. Insgesamt liegt die wesentliche Problematik bei der Lokalisierung und Extraktion von Daten in der Tabellenbereichserkennung und der Tabellenstrukturanalyse[HKLW02]. Hierzu wurden zahlreiche Forschungen und Methoden veröffentlicht. Die traditionellen Methoden[CK93, Kie98] stützen sich auf a priori bekannten Regeln oder manuell definierte Merkmale, um präzise Ergebnisse auf Tabellen mit einer bestimmten Struktur zu erzielen, die Robustheit sind jedoch eingeschränkt. Die auf neuronalen Netzen basierenden Ansätze[PVR+19, LHY+21] können mit vielfältigen Tabellentypen umgehen, sind jedoch auf CPU zeitaufwendig[Sto22] und mit beschränkter Interpretierbarkeit[GAZ19].
In dieser Arbeit wurde ein *Python*-Skript zum Sammeln und Verarbeiten von Daten, die in Tabellenform auf Bild dargestellt werden, entwickelt. Die Aufgabe war in drei Teilaufgaben unterteilt: die Erkennung von Tabellenbereichen, die Analyse der Tabellenstruktur und die Nachbearbeitung zur Normalisierung vor dem Schreiben in die Datenbank.
Vier neuronale Netze wurden zuerst trainiert und ihre Leistung bei entsprechenden Aufgaben verglichen. Anhand der Beobachtung im Abschnitt 5.1, dass das *DenseNet* Modell in den Tasks Tabellenbereichserkennung und Spaltenbereichserkennung nach Nachbearbeitung 98.91% und 92.31% mIoU-Score erreichte, wurde es zur Tabellenbereichserkennung und Tabellenstrukturanalyse eingesetzt.
Für die Tabellenbereichserkennung wurden die Verfahren unter Unterstützung von künstlichen neuronalen Netzen (KNN), linienbasierter Erkennung und textblöckebasierter Erkennung vorgeschlagen. Auf dem Testsatz wurden anschließend die jeweiligen Erkennungsergebnisse und der Zeitaufwand auf CPU und GPU in der Tabelle 5.2 verglichen. Hierbei dominierte das Verfahren unter Verwendung neuronaler Netze mit dem besten mIoU-Score 98.91%, obwohl es mehr Zeit kostet. Im Vergleich dazu befanden sich die Scores der beiden traditionellen Verfahren nur gegen 80%.
Um die Tabellenstrukturanalyse zu erleichtern, wurde eine Tabelle in vier Teilen (Abb. 4.21 (b)) unterteilt. Für die Tabellenstrukturanalyse und die normalisierte Nachverarbeitung wurden drei Pipelines vorgeschlagen:

Pipeline 1) *Linienrekonstruktionsbaiserte Tabellenstrukturanalyse (mit K-Means-Cluster-ing) + OCR und Tabellenrekonstruktion + Nachverarbeitung*

Pipeline 2) *Textblöckebaiserte Tabellenstrukturanalyse (mit K-Means-Clustering) + OCR und Tabellenrekonstruktion + Nachverarbeitung*

Pipeline 3) *Tabellenstrukturanalyse mit Unterstützung neuronaler Netze (und K-Means-Clustering) + OCR und Tabellenrekonstruktion + Nachverarbeitung*

Bei K-Means-Clustering wurde der Mittelwert der Zellengröße unter Three-Sigma Limit[KEN22] zum Korrigieren der Ergebnisse verwendet. Außerdem wurden in der normalisierte Nachverarbeitung durchdachte Regeln und Annahmen erstellt, um die extrahierten Tabellen zu normalisieren und vollständig in die Datenbank einzuschreiben. Die Zellengenauigkeitsrate jeder Pipeline für die Verarbeitungsergebnisse verschiedener Teile der Tabelle wurden mithilfe zehn Tabellen gezählt. Ähnlich wie bei der Tabellenbereichserkennung zeigte in der Tabelle 5.3 die dritte Pipeline mit KNN die beste Leistung im Values-Teil und Köpfe-Teil. Im Values-Teil erreichte die Genauigkeit dieser Pipeline 99.18%. Selbst in der schwierigsten Analyse des Köpfe-Teils erreichte sie eine 75.81% Zellengenauigkeit. Allerdings erzielte sie jedoch eine geringere Genauigkeit (90.97%) im Zeilenköpfe-Teil als auf der Linienrekonstruktion basierende Pipeline 1 (98.73%).
Insgesamt wurde in dem *Python*-Skript die Tabellenregionen zuerst unter Verwendung des *DenseNet* Modells auf vorverarbeiteten Bilddokumenten bestimmt. Die dritte Pipeline wurde dann verwendet, um die Tabellenstruktur zu analysieren. Dabei wurde die Tabelle digital rekonstruiert, indem zuerst alle geraden Linien entfernt und Textblöcke erkannt wurden, um alle Zellen zu lokalisieren. Zur Bestimmung der Beziehungen zwischen alle Zellen wurden anschließend die Zeilen und Spalten der Tabelle segmentiert, wobei jede Spalte unter Verwendung des *DenseNet* Modells erkannt wurde, während der K-Means-Clustering für die Zeilenerkennung eingesetzt wurde. Schließlich wurden die extrahierten Tabellen unter Verwendung konzipierter Nachbearbeitung im Abschnitt 4.5 in einfache Tabellen normalisiert und in einer Datenbank eingeschrieben.

8. Ausblick

Aufgrund der Tabelle 5.3 dominiert die linienrekonstruktionsbasierte Tabellenstrukturanalyse bei der Zeilenklassifizierung mit einer höchsten Zellgenauigkeitsrate im Zeilenköpfe-Teil. Dies bedeutet, dass das auf Linienrekonstruktion basierende Verfahren Tabellenzeilen genauer unterscheiden kann als mittels K-Means-Clustering. Daher könnten in weiterer Implementierung die Zeilen statt mithilfe K-Means-Clustering basierend auf die Linienrekonstruktion die Zeilen klassifiziert werden.
Wie in der Diskussion genannt, ist es auch möglich, neuronale Netze für die Zeilenerkennung und Verarbeitung der Tabellenköpfe zu trainieren. Darüber hinaus könnte OCR auch unter Verwendung von selbst trainierten neuronalen Netzen durchgeführt werden.
Bei der Nachbearbeitung zur Normalisierung werden komplexe Regeln und Annahmen verwendet, diese Annahmen gelten jedoch nur für eine begrenzte Anzahl von Tabellentypen. In Zukunft wäre es möglich, Annahmen mit einer breiteren Abdeckung zu erstellen, damit das Programm komplexere Tabellen handhaben kann. Es wäre auch möglich, dafür geeignete neuronale Netze zu trainieren.
Wie die Tabellenköpfe im Köpfe-Teil befinden sich oft auch Zeilenköpfe mit mehreren Ebenen im Zeilenköpfe-Teil, dazu wird als ein Beispiel die Tabelle in der Abbildung 4.18 genannt. Die ähnlichen Kopier- und Verschmelzungsoperationen auf dem Köpfe-Teil (der Schritt 5 bis 10 im Abschnitt 4.5) sollten auch auf dem Zeilenköpfe-Teil durchgeführt werden. Aufgrund der Ausrichtung des Zeilenkopfs wurde jedoch kein geeignetes Kopier- und Fusionsverfahren gefunden.
Darüber hinaus besteht der in dieser Arbeit verwendete Datensatz aus nur 1155 Bildern, verglichen mit den Millionen von Datensätzen, die für modernes Deep-Learning-Training verwendet werden. In Zukunft könnte der Datensatz erweitert werden, um die neuronalen Netze besser zu trainieren.
Für dieses *Python*-Skript wurde schon eine auf *HTML* und *Node.js* basierende Webanwendung entwickelt, um ein komfortables User Interface bereitzustellen. Somit lässt sich der Zwischenprozess der Tabellenverarbeitung visualisieren und die in die Datenbank geschriebenen Ergebnisse der Tabellenextraktion abfragen. Diese Anwendung befindet sich jedoch noch im Beta-Stadium und sollte durch die Verwendung von *Angular* verbessert werden.

Literaturverzeichnis

[ABC+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[Ada20] Nikolas Adaloglou. Understanding the receptive field of deep convolutional networks. *AI Summer*, von `https://theaisummer.com/receptive-field/`. 2020. Abgerufen: 14. Januar 2023.

[AW10] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[bae22] baeldung. What is a learning curve in machine learning? *baeldung*, von `https://www.baeldung.com/cs/learning-curve-ml`. 2022. Abgerufen: 12. Januar 2023.

[BGBV16] Kyle Banker, Douglas Garrett, Peter Bakkum, and Shaun Verch. *MongoDB in action: covers MongoDB version 3.0*. Simon and Schuster, 2016.

[BIK+20] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020.

[Bis15] Camily Bishop. Ocr? icr? iwr? omg! get the most from your scanned text. *Crowley*, von `https://thecrowleycompany.com/ocr-icr-iwr-omg-get-scanned-text/`. 2015. Abgerufen: 18. Oktober 2022.

[Bod06] Freimut Bodendorf. *Daten und Wissen*. Springer, 2006.

[Buz11] Thorsten M Buzug. *Einführung in die Computertomographie: mathematisch-physikalische Grundlagen der Bildrekonstruktion*. Springer-Verlag, 2011.

[BYM17] Christian Bartz, Haojin Yang, and Christoph Meinel. Stn-ocr: A single neural network for text detection and text recognition. *arXiv preprint arXiv:1707.08831*, 2017.

[Car22] Pierre Carbonnelle. Pypl popularity of programming language index. von https://pypl.github.io/PYPL.html, 2022. Abgerufen: 03. Oktober 2022.

[CDG+06] F Chang, J Dean, S Ghemawat, WC Hsieh, DA Wallach, M Burrows, T Chandra, A Fikes, and R Gruber. Bigtable: A distributed structured data storage system. In *7th OSDI*, volume 26, pages 305–314, 2006.

[Cha20] Mukesh Chaudhary. Silhouette analysis in k-means clustering. *Medium*, von https://medium.com/@cmukesh8688/silhouette-analysis-in-k-means-clustering-cefa9a7ad111. 2020. Abgerufen: 03. Januar 2023.

[CHX+19] Zewen Chi, Heyan Huang, Heng-Da Xu, Houjin Yu, Wanxuan Yin, and Xian-Ling Mao. Complicated table structure recognition. *arXiv preprint arXiv:1908.04729*, 2019.

[CK93] Surekha Chandran and Rangachar Kasturi. Structural recognition of tabulated data. In *Proceedings of 2nd International Conference on Document Analysis and Recognition (ICDAR'93)*, pages 516–519. IEEE, 1993.

[Cod70] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[Com12] Peltarion Community. Binary crossentropy. *ML Platform - Knowledge Center*, von https://peltarion.com/knowledge-center/modeling-view/build-an-ai-model/loss-functions/binary-crossentropy. 2012. Abgerufen: 25. Oktober 2022.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[CVMBB14] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[Dep22a] Computer Science Department. Fundamentals of programming and computer science. *Carnegie Mellon University*, von https://www.cs.cmu.edu/~112/. 2022. Abgerufen: 26. Oktober 2022.

[Dep22b] Statista Research Department. Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025. *IDC; Statista*, von `https://www.statista.com/statistics/871513/worldwide-data-created/`. 2022. Abgerufen: 13. Oktober 2022.

[DH72] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[DS18] Julian Schrittwieser Demis Hassabis David Silver, Thomas Hubert. Alphazero: Shedding new light on chess, shogi, and go. *DeepMind*, von `https://www.deepmind.com/blog/alphazero-shedding-new-light-on-chess-shogi-and-go`. 2018. Abgerufen: 26. Oktober 2022.

[DT05] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.

[Edu20] IBM Cloud Education. Artificial intelligence (ai). *IBM*, von `https://www.ibm.com/cloud/learn/what-is-artificial-intelligence`. 2020. Abgerufen: 26. Oktober 2022.

[EOW10] Boris Epshtein, Eyal Ofek, and Yonatan Wexler. Detecting text in natural scenes with stroke width transform. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 2963–2970. IEEE, 2010.

[FFRC+17] Waleed Farrukh, Antonio Foncubierta-Rodriguez, Anca-Nicoleta Ciubotaru, Guillaume Jaume, Costas Bejas, Orcun Goksel, and Maria Gabrani. Interpreting data from scanned tables. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 2, pages 5–6. IEEE, 2017.

[FH89] Evelyn Fix and Joseph Lawson Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique*, 57(3):238–247, 1989.

[Fou22] Python Software Foundation. Find, install and publish python packages with the python package index. *Python Package Index*, von `https://pypi.org/`. 2022. Abgerufen: 26. Oktober 2022.

[FS05] Paolo Favaro and Stefano Soatto. A geometric approach to shape from defocus. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(3):406–417, 2005.

[FTT+12] Jing Fang, Xin Tao, Zhi Tang, Ruiheng Qiu, and Ying Liu. Dataset, ground-truth and performance metrics for table detection evaluation. In *2012 10th IAPR International Workshop on Document Analysis Systems*, pages 445–449. IEEE, 2012.

[GAZ19] Amirata Ghorbani, Abubakar Abid, and James Zou. Interpretation of neural networks is fragile. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 3681–3688, 2019.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[GBI+13] Ian J Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *arXiv preprint arXiv:1312.6082*, 2013.

[GCW+19] Yunze Gao, Yingying Chen, Jinqiao Wang, Ming Tang, and Hanqing Lu. Reading scene text with fully convolutional sequence modeling. *Neurocomputing*, 339:161–170, 2019.

[GDPP05] Basilios Gatos, Dimitrios Danatsas, Ioannis Pratikakis, and Stavros J Perantonis. Automatic table detection in document images. In *International Conference on Pattern Recognition and Image Analysis*, pages 609–618. Springer, 2005.

[GFGS06] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376, 2006.

[GHTC13] Katarina Grolinger, Wilson A Higashino, Abhinav Tiwari, and Miriam AM Capretz. Data management in cloud environments: Nosql and newsql data stores. *Journal of Cloud Computing: advances, systems and applications*, 2(1):1–24, 2013.

[Goy11] Megha Goyal. Morphological image processing. *IJCST*, 2(4):59, 2011.

[GT15] Clinton Gormley and Zachary Tong. *Elasticsearch: the definitive guide: a distributed real-time search and analytics engine*. Ö'Reilly Media, Inc.", 2015.

[Ham13] John Hammersley. *Monte carlo methods*. Springer Science & Business Media, 2013.

[Han00] John C Handley. Table analysis for multiline cell identification. In *Document Recognition and Retrieval VIII*, volume 4307, pages 34–43. SPIE, 2000.

[HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[HJ11] Robin Hecht and Stefan Jablonski. Nosql evaluation: A use case oriented survey. In *2011 International Conference on Cloud and Service Computing*, pages 336–341. IEEE, 2011.

[HKLW02] Jianying Hu, Ramanujan S Kashi, Daniel Lopresti, and Gordon T Wilfong. Evaluating the performance of table processing algorithms. *International Journal on Document Analysis and Recognition*, 4(3):140–153, 2002.

[HLVDMW17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[HPT22] Jiawei Han, Jian Pei, and Hanghang Tong. *Data mining: concepts and techniques*. Morgan kaufmann, 2022.

[HS81] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[HS+88] Chris Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[Hu19] Dichao Hu. An introductory survey on attention mechanisms in nlp problems. In *Proceedings of SAI Intelligent Systems Conference*, pages 432–448. Springer, 2019.

[HW79] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979.

[HYL+19] Yilun Huang, Qinqin Yan, Yibo Li, Yifan Chen, Xiong Wang, Liangcai Gao, and Zhi Tang. A yolo-based table detection method. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 813–818. IEEE, 2019.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IH81] Katsushi Ikeuchi and Berthold KP Horn. Numerical shape from shading and occluding boundaries. *Artificial intelligence*, 17(1-3):141–184, 1981.

[Ins22] Robert Koch Insititut. Aktuelle situationsberichte, wochenberichte und pandemieradar. *Robert Koch Insititut*, von `https://www.rki.de/DE/Content/InfAZ/N/Neuartiges_Coronavirus/Situationsberichte/Wochenbericht/Wochenberichte_Tab.html`. 2022. Abgerufen: 24. Oktober 2022.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[Jaw05] Ali Jawad. *Computer processing of remotely sensed images*. PhD thesis, ALIGARH MUSLIM UNIVERSITY ALIGARH, 2005.

[JEP+21] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[JSZ+15] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. *Advances in neural information processing systems*, 28, 2015.

[KEN22] WILL KENTON. Three sigma limits statistical calculation, with an example. *Investopedia*, von `https://www.investopedia.com/terms/t/three-sigma-limits.asp`. 2022. Abgerufen: 16. Januar 2023.

[Kie98] Thomas G Kieninger. Table structure recognition based on robust block segmentation. In *Document Recognition V*, volume 3305, pages 22–32. SPIE, 1998.

[KLW19] Uday Kamath, John Liu, and James Whitaker. *Deep learning for NLP and speech recognition*, volume 84. Springer, 2019.

[KM20] Lazhar Khelifi and Max Mignotte. Deep learning for change detection in remote sensing images: Comprehensive review and meta-analysis. *Ieee Access*, 8:126385–126400, 2020.

[Kof04] Michael Kofler. What is mysql? In *The Definitive Guide to MySQL*, pages 3–19. Springer, 2004.

[KR09] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009.

[KS17] Karambir Kaur and Monika Sachdeva. Performance evaluation of newsql databases. In *2017 International Conference on Inventive Systems and Control (ICISC)*, pages 1–5. IEEE, 2017.

[KW16] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[KWT88] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988.

[Lan22] Niklas Lang. Database basics: Acid transactions. *Medium*, von `https://towardsdatascience.com/database-basics-acid-transactions-bf4d38bd8e26`. 2022. Abgerufen: 08. Oktober 2022.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[LHY+21] Yiren Li, Zheng Huang, Junchi Yan, Yi Zhou, Fan Ye, and Xianhui Liu. Gfte: graph-based financial table extraction. In *International Conference on Pattern Recognition*, pages 644–658. Springer, 2021.

[LKB+17] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[LLY+18] Xuebo Liu, Ding Liang, Shi Yan, Dagui Chen, Yu Qiao, and Junjie Yan. Fots: Fast oriented text spotting with a unified network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5676–5685, 2018.

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[Low99] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[LPH+20] Minghui Liao, Guan Pang, Jing Huang, Tal Hassner, and Xiang Bai. Mask textspotter v3: Segmentation proposal network for robust scene text spotting. In *European Conference on Computer Vision*, pages 706–722. Springer, 2020.

[LSB18] Minghui Liao, Baoguang Shi, and Xiang Bai. Textboxes++: A single-shot oriented scene text detector. *IEEE transactions on image processing*, 27(8):3676–3690, 2018.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[LZW+19] Minghui Liao, Jian Zhang, Zhaoyi Wan, Fengming Xie, Jiajun Liang, Pengyuan Lyu, Cong Yao, and Xiang Bai. Scene text recognition from two-dimensional perspective. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 8714–8721, 2019.

[MAJ12] Anand Mishra, Karteek Alahari, and CV Jawahar. Scene text recognition using higher order language priors. In *BMVC-British machine vision conference*. BMVA, 2012.

[Mat22] MathWorks. Anchor boxes for object detection. *MathWorks*, von `https://de.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html`. 2022. Abgerufen: 03. Januar 2023.

[MCDC06] Sekhar Mandal, SP Chowdhury, Amit K Das, and Bhabatosh Chanda. A simple and effective table detection system from document images. *International Journal of Document Analysis and Recognition (IJDAR)*, 8(2):172–182, 2006.

[Mis20] Divyanshu Mishra. Transposed convolution demystified. *Towards Data Science*, von https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba. 2020. Abgerufen: 11. Dezember 2022.

[MKB+10] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari, 2010.

[MM97] Tom M Mitchell and Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.

[Moh20] Tobias Mohr. Digitization made easy with ocr and icr. *MoreThanDigital*, von https://morethandigital.info/en/digitization-made-easy-with-ocr-and-icr/. 2020. Abgerufen: 18. Oktober 2022.

[OCR09] OCRWizard. How ocr software works. von https://web.archive.org/web/20090816210246/http://ocrwizard.com/ocr-software/how-ocr-software-works.html. 2009. Abgerufen: 11. Oktober 2022.

[Ora18] Oracle. Was ist eine datenbank? *Oracle Deutschland*, von https://www.oracle.com/de/database/what-is-database/. 2018. Abgerufen: 26. Oktober 2022.

[OS88] Stanley Osher and James A Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of computational physics*, 79(1):12–49, 1988.

[OW12] Martijn van Otterlo and Marco Wiering. Reinforcement learning and markov decision processes. In *Reinforcement learning*, pages 3–42. Springer, 2012.

[Pea01] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.

[Pet10] Tim Peters. The zen of python. In *Pro Python*, pages 301–302. Springer, 2010.

[PGK+20] Devashish Prasad, Ayan Gadpal, Kshitij Kapadni, Manish Visave, and Kavita Sultanpure. Cascadetabnet: An approach for end to end table detection and structure recognition from image-based documents. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 572–573, 2020.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[Pra22] Sohom Pramanick. History of python. *GeeksforGeeks*, von `https://www.geeksforgeeks.org/history-of-python/`. 2022. Abgerufen: 2. Oktober 2022.

[PVR+19] Shubham Singh Paliwal, D Vishwanath, Rohit Rahul, Monika Sharma, and Lovekesh Vig. Tablenet: Deep learning model for end-to-end table detection and tabular data extraction from scanned document images. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 128–133. IEEE, 2019.

[QLC+21] Liang Qiao, Zaisheng Li, Zhanzhan Cheng, Peng Zhang, Shiliang Pu, Yi Niu, Wenqi Ren, Wenming Tan, and Fei Wu. Lgpma: Complicated table structure recognition with local and global pyramid mask alignment. In *International Conference on Document Analysis and Recognition*, pages 99–114. Springer, 2021.

[RAA+17] Sheikh Faisal Rashid, Abdullah Akmal, Muhammad Adnan, Ali Adnan Aslam, and Andreas Dengel. Table recognition in heterogeneous documents using machine learning. In *2017 14th IAPR International conference on document analysis and recognition (ICDAR)*, volume 1, pages 777–782. IEEE, 2017.

[Ras15] Sebastian Raschka. *Python machine learning*. Packt publishing ltd, 2015.

[RDGF16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[RHGS15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[Ril17] Jenn Riley. Understanding metadata. *Washington DC, United States: National Information Standards Organization (http://www. niso. org/publications-/press/UnderstandingMetadata. pdf)*, 23, 2017.

[RL14] Martin Riedmiller and A Lernen. Multi layer perceptron. *Machine Learning Lab Special Lecture, University of Freiburg*, pages 7–24, 2014.

[Sag21] Arun Sagar. Tablenet-pytorch. von `https://github.com/asagar60/TableNet-pytorch`. 2021. Abgerufen: 24. Juni 2022.

[Sar22] Pratibha Sarin. How to perform unstructured data to structured data conversion? | 9 easy steps. *HEVO*, von `https://hevodata.com/learn/unstructured-data-to-structured-data-conversion/`. 2022. Abgerufen: 13. Oktober 2022.

[SAW+17] Sebastian Schreiber, Stefan Agne, Ivo Wolf, Andreas Dengel, and Sheraz Ahmed. Deepdesrt: Deep learning for detection and structure recognition of tables in document images. In *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*, volume 1, pages 1162–1167. IEEE, 2017.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[SBY16] Baoguang Shi, Xiang Bai, and Cong Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2298–2304, 2016.

[Ser11] Couchbase Server. Couchbase server redefining database economics. von `https://www.couchbase.com/products/server`, 2011. Abgerufen: 08. Oktober 2022.

[Sha17] Tarang Shah. About train, validation and test sets in machine learning. *Towards Data Science*, von `https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7`. 2017. Abgerufen: 24. Oktober 2022.

[SHK+14a] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[SHK+14b] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[sI22] solid IT. Db-engines ranking. von `https://db-engines.com/en/ranking`, 2022. Abgerufen: 08. Oktober 2022.

[SK19] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[SKDA19] Shoaib Ahmed Siddiqui, Pervaiz Iqbal Khan, Andreas Dengel, and Sheraz Ahmed. Rethinking semantic segmentation for table structure recognition in documents. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1397–1402. IEEE, 2019.

[SL19] Nico Litzel Stefan Luber. Was ist ein rekurrentes neuronales netz (rnn)? *BIGDATA INSIDER*, von `https://www.bigdata-insider.de/was-ist-ein-rekurrentes-neuronales-netz-rnn-a-843274/`. 2019. Abgerufen: 26. Oktober 2022.

[SPP10] Vasilis A Sotiris, W Tse Peter, and Michael G Pecht. Anomaly detection through a bayesian support vector machine. *IEEE Transactions on Reliability*, 59(2):277–286, 2010.

[Sto22] Pure Storage. Cpu vs. gpu for machine learning. *PURESTORAGE*, von `https://blog.purestorage.com/purely-informational/cpu-vs-gpu-for-machine-learning/`. 2022. Abgerufen: 29. Januar 2023.

[SW13] Michael Stonebraker and Ariel Weisberg. The voltdb main memory dbms. *IEEE Data Eng. Bull.*, 36(2):21–27, 2013.

[Swa13] CH Swaroop. *A Byte of Python*. Independent, 2013.

[SWL+16] Baoguang Shi, Xinggang Wang, Pengyuan Lyu, Cong Yao, and Xiang Bai. Robust scene text recognition with automatic rectification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4168–4176, 2016.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[SZH19] Ningning Sun, Yuanping Zhu, and Xiaoming Hu. Faster r-cnn based table detection combining corner locating. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1314–1319. IEEE, 2019.

[TaloJ] Talend. Strukturierte vs. unstrukturierte daten: ein leitfaden. *Talend Data Fabric*, von https://www.talend.com/de/resources/strukturierte-vs-unstrukturierte-daten. o.J. Abgerufen: 01. Oktober 2022.

[THH+16] Zhi Tian, Weilin Huang, Tong He, Pan He, and Yu Qiao. Detecting text in natural image with connectionist text proposal network. In *European conference on computer vision*, pages 56–72. Springer, 2016.

[Tiu19] Ekin Tiu. Metrics to evaluate your semantic segmentation model. *Medium*, von https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2. 2019. Abgerufen: 09. Januar 2023.

[TNS20] Paulius Tumas, Adam Nowosielski, and Arturas Serackis. Pedestrian detection in severe weather conditions. *IEEE Access*, 8:62775–62784, 2020.

[Tor22] Torchvision. Densenet. *Pytorch*, von https://github.com/pytorch/vision/blob/main/torchvision/models/densenet.py. 2022. Abgerufen: 13. November 2022.

[TS19] Domen Tabernik and Danijel Skočaj. Deep learning for large-scale traffic-sign detection and recognition. *IEEE transactions on intelligent transportation systems*, 21(4):1427–1440, 2019.

[TSM+20] Rebecca Taft, Irfan Sharif, Andrei Matei, Nathan VanBenschoten, Jordan Lewis, Tobias Grieger, Kai Niemi, Andy Woods, Anne Birzin, Raphael Poss, et al. Cockroachdb: The resilient geo-distributed sql database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1493–1509, 2020.

[Ven03] Bill Venners. The making of python: A conversation with guido van rossum, part i. *Artima Developer*, von http://www.artima.com/intv/pythonP.html. 2003. Abgerufen: 03. Oktober 2022.

[VGJMR12] Rafael Grompone Von Gioi, Jérémie Jakubowicz, Jean-Michel Morel, and Gregory Randall. Lsd: A line segment detector. *Image Processing On Line*, 2:35–55, 2012.

[Vis22a] Open Source Computer Vision. Image filtering. *Opencv 4.6.0-dev*, von https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html. 2022. Abgerufen: 30. Oktober 2022.

[Vis22b] Open Source Computer Vision. Image thresholding. *Opencv 4.6.0-dev*, von `https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html`. 2022. Abgerufen: 30. Oktober 2022.

[Vis22c] Open Source Computer Vision. Fast line detector. *Opencv 4.6.0-dev*, von `https://docs.opencv.org/4.x/df/ded/group__ximgproc__fast__line__detector.html`. 2022. Abgerufen: 24. Oktober 2022.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[WB10] Kai Wang and Serge Belongie. Word spotting in the wild. In *European conference on computer vision*, pages 591–604. Springer, 2010.

[WBB11] Kai Wang, Boris Babenko, and Serge Belongie. End-to-end scene text recognition. In *2011 International conference on computer vision*, pages 1457–1464. IEEE, 2011.

[WKRQ+08] Xindong Wu, Vipin Kumar, J Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J McLachlan, Angus Ng, Bing Liu, Philip S Yu, et al. Top 10 algorithms in data mining. *Knowledge and information systems*, 14(1):1–37, 2008.

[Woo22] Thomas Wood. Softmax function. *DeepAI*, von `https://deepai.org/machine-learning-glossary-and-terms/softmax-layer`. 2022. Abgerufen: 03. Januar 2023.

[WXL+19] Wenhai Wang, Enze Xie, Xiang Li, Wenbo Hou, Tong Lu, Gang Yu, and Shuai Shao. Shape robust text detection with progressive scale expansion network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9336–9345, 2019.

[YBSL14] Cong Yao, Xiang Bai, Baoguang Shi, and Wenyu Liu. Strokelets: A learned multi-scale representation for scene text recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4042–4049, 2014.

[YKK17] Byoung-Ha Yoon, Seon-Kyu Kim, and Seon-Young Kim. Use of graph database for the integration of heterogeneous biological data. *Genomics & informatics*, 15(1):19–27, 2017.

[YLW+20] Jiayi Yuan, Hongye Li, Meng Wang, Ruyang Liu, Chuanyou Li, and Beilun Wang. An opencv-based framework for table information extraction. In *2020 IEEE International Conference on Knowledge Graph (ICKG)*, pages 621–628. IEEE, 2020.

[Yon19] Yonke. Document recognition technology in the ai era: Recognition of table in image. *Die CV-Anwendungsforschungsgruppe des Tencent QQ R&D Center*, von `https://bit.ly/3yyEz07`. 2019. Abgerufen: 10. Oktober 2022.

[Zah09] Alexander Zahariev. Google app engine. *Helsinki University of Technology*, pages 1–5, 2009.

[ZXZL19] Zitao Zeng, Weihao Xie, Yunzhe Zhang, and Yao Lu. Ric-unet: An improved neural network based on unet for nuclei segmentation in histology images. *Ieee Access*, 7:21420–21428, 2019.

[ZYW+17] Xinyu Zhou, Cong Yao, He Wen, Yuzhi Wang, Shuchang Zhou, Weiran He, and Jiajun Liang. East: an efficient and accurate scene text detector. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 5551–5560, 2017.

A. Anhang