

Bachelorarbeit

Extraktion von Diagrammen aus Texten und Auswertung von Liniendiagrammen mit Deep-Learning Methoden

Luzian Uihlein

Würzburg, 18. August 2024



Julius-Maximilians-Universität Würzburg

Lehrstuhl für Informatik VI

Betreuer: Prof. Dr. Frank Puppe

Norbert Fischer

Alexander Hartelt

Abstract

Hallo. Ich bin ein kleiner Blindtext. Und zwar schon so lange ich denken kann. Es war nicht leicht zu verstehen, was es bedeutet, ein blinder Text zu sein: Man ergibt keinen Sinn. Wirklich keinen Sinn. Man wird zusammenhangslos eingeschoben und rumgedreht – und oftmals gar nicht erst gelesen. Aber bin ich allein deshalb ein schlechterer Text als andere? Na gut, ich werde nie in den Bestsellerlisten stehen. Aber andere Texte schaffen das auch nicht. Und darum stört es mich nicht besonders blind zu sein. Und sollten Sie diese Zeilen noch immer lesen, so habe ich als kleiner Blindtext etwas geschafft, wovon all die richtigen und wichtigen Texte meist nur träumen.

Inhaltsverzeichnis

1 Einleitung	6
2 Literaturübersicht	7
3 Methodik	9
3.1 Extraktion von Diagrammen aus Texten	9
3.1.1 Datensatz DocBank zur Objekterkennung	9
3.1.2 Datensatz historischer Wirtschaftsscans zur Objekterkennung	11
3.2 Schwierigkeitsklassifizierung von Liniendiagrammen	12
3.3 Auswertung von Liniendiagrammen	13
3.3.1 Datensatz von synthetischen Liniendiagrammen zur Segmentation	14
3.3.2 Datensatz von historischen Liniendiagrammen zur Segmentation	15
4 Implementation	16
4.1 Ultralytics YOLO	16
4.1.1 Objekterkennung zur Extraktion von Diagrammen aus Texten	17
4.1.2 Klassifizierung zur Schwierigkeitsbestimmung von Liniendiagrammen	18
4.1.3 Instanzsegmentation von Wertelinien in Liniendiagrammen	19
4.2 U-Net: Semantische Segmentation von Wertelinien	20
4.3 Numerische Auswertung von Liniendiagrammen in Tabellenform	22
4.3.1 Extraktion der segmentierten Wertelinien	22
4.3.2 Achsenerkennung durch optische Schriftzeichenerkennung	24
4.3.3 Grafische Darstellung und tabellarische Auswertung	27
5 Experimente	30
5.1 Objekterkennung	31
5.1.1 Vortrainiertes Modell auf DocBank	31
5.1.2 Feintraining auf historische Wirtschaftsscans	35
5.2 Liniendiagrammsklassifizierung	37
5.3 Segmentation	38
5.3.1 Instanzsegmentation durch Ultralytics YOLO	38
5.3.2 Semantische Segmentation durch das U-Net	38

5.4	Diagrammauswertung	40
5.4.1	Achsenerkennung durch OCR	40
5.4.2	Numerische Tabellenformextraktion	40
6	Zusammenfassung	43

Kapitel 1

Einleitung

Kapitel 2

Literaturübersicht

Die automatische Transkription von Liniendiagrammen ist weit weniger erforscht als die von Tabellen, z.B. gibt es auf den ICDAR-Konferenzen (International Conference on Document Analysis and Recognition) keine Wettbewerbe (Challenges) mit annotierten Datensätzen, im Gegensatz zu Tabellen und vielen anderen Bereichen. Es gibt nur wenige Publikationen, die sich mit diesem Problem beschäftigen, wobei aktuelle Ansätze [18, 19] Deep-Learning-Techniken verwenden, die mangels annotierter realer Daten überwiegend mit synthetischen Daten trainiert werden. In der Literatur wird die Erkennung von Liniendiagrammen meist in folgende Schritte unterteilt:

1. Erkennen und Klassifizieren des Diagramms
2. Erkennen der x- und y-Achse des Liniendiagramms
3. Erkennen der Linien
4. Erkennen der Beschriftungen
5. Extraktion der Datenpunkte auf den Linien
6. Zuordnung der Datenpunkte zu den semantischen x- und y-Werten
7. Darstellung des Ergebnisses als Tabelle.

Während einfache Linien gut erkannt werden, wird bei überlappenden Linien oft angenommen, dass diese farbig gezeichnet werden, um sie zu unterscheiden. Dies gilt jedoch nicht für historische Liniendiagramme, die in der Regel durch verschiedene gestrichelte Linien unterschieden werden, was automatisch schwer zu erkennen ist. Dafür eignen sich semiautomatische Ansätze wie z.B. in [20] beschrieben. Hierbei werden die automatischen Schritte von den Anwendern sofort manuell überprüft und korrigiert, was bei einer Massentranskription nicht praktikabel, aber bei einer begrenzten Anzahl von Diagrammen realistisch ist, zumal eine Qualitätskontrolle für die GT-Erstellung ohnehin notwendig ist. Erforschte Herangehensweisen [21] zur Linienerkennung und Datenextraktion bestehen

unter anderem aus der Erkennung von Schlüsselpunkten (key point detection) der jeweiligen Wertelinien, welche hier durch Steigungsänderungen (pivot points) festgelegt werden. Nach deren Erkennung durch ein neurales Netzwerk werden diese mit Hilfe einer zusätzlichen Faltungsschicht (convolution layer) zu einzelnen Linieninstanzen gruppiert. Andere Linieninstanzgruppierungsalgorithmen [22] bestehen in der Optimierung einer Kostenfunktion mithilfe der linearen Programmierung über ein Minimum-Kosten-Fluss-Problem (minimum-cost-flow problem). Im Vergleich zu handgeschriebenen, historischen Liniendiagrammen allerdings, bestehen die Datensätze exklusiv aus computergenerierten Textbeschriftungen, sodass die optische Schriftzeichenerkennung (optical character recognition) erfolgreicher durchgeführt werden kann. Die Zuordnung der Datenpunkte zu den semantischen x- und y-Werten erfolgt dadurch fehlerfreier, was wie bei allen Zwischenschritten die Effizienz des Endergebnisses direkt beeinflusst.

Zur Evaluation werden die Linien als kontinuierliches Ähnlichkeitsproblem (continuous similarity problem) behandelt. Die Punktsequenz der Vorhersage des Modells und eine definierte Grundwahrheitsmenge werden verglichen, sodass Präzision (precision), Erinnerung (recall) und F1-Wert (F1-Score) berechnet werden können.

Kapitel 3

Methodik

3.1 Extraktion von Diagrammen aus Texten

Ziel des ersten Teils ist die Extraktion der Diagramme aus den historischen Textscans, welche dann im folgenden Teil in eine gewünschte Form ausgewertet werden können.

Die wesentlichen Schritte des Extraktionssteils beinhalten die Objekterkennung, also die Bestimmung des Begrenzungsrechtecks (bounding box) der Diagramme innerhalb der vorliegenden Vollseitescans und deren Unterscheidung in verschiedene Diagrammtypen, beispielsweise Linien- und Balkendiagramme. Die erkannten Liniendiagramme werden anschließend anhand ihrer Auswertungsschwierigkeit klassifiziert, etwa durch Kennzeichnung der Diagramme, welche kontextbedingt gruppiert wurden, zum Beispiel aufgrund gemeinsamer Graphachsen.

Um mit Hilfe von Deep-Learning Modelle zu trainieren, werden annotierte Grundwahrheiten (ground truth) benötigt.

3.1.1 Datensatz DocBank zur Objekterkennung

Für die Erkennung von Diagrammen in Texten wurden DocBank [1] und ein Anteil der historischen Wirtschaftsscans verwendet. DocBank besteht aus wissenschaftlichen Publikationen mit computergenerierten Grafiken, weshalb DocBanks' Dokumentenseiten lediglich zum Vortrainieren des Detektionsmodells gedacht sind. Beabsichtigt wurde dieser Prozess des Vortrainierens, um das System schneller und allgemeingültiger, also mit besseren Voraussagen, trainieren zu können. Spätere Experimente untersuchen diese Annahme.

An die Vorkommenshäufigkeit angepasst, wurde die Differenzierung zuerst in fünf Objektklassen beschlossen: Linien, Balken, Histogramme, Sonstige und Gemischte. Da jedoch im Verlauf der Experimente nicht genügend Histogramm-Klassen vorhanden waren, wurden die Datensätze auf vier Klassen reduziert, indem Balkendiagramme und Histogramme vereinigt wurden.

Die Überlegung, zwischen Balkendiagrammen und Histogrammen zu unterscheiden, beruhte darauf, dass Balkendiagramme kategorische Datenvergleiche anschaulich machen, bei denen die Balkenanordnung irrelevant ist, während Histogramme kontinuierliche, numerische Daten darstellen. Die Differenz liegt allerdings lediglich an der Achsenbeschreibung und nicht an visuellen Hinweisen, oftmals werden Balkendiagramme jedoch mit Lücken zwischen den Balken dargestellt, während Histogramme lückenlos abgebildet werden; dies ist jedoch nicht ausschlaggebend zur Bestimmung des Diagrammtyps.

Für die manuelle GT-Annotation der DocBank Dokumentenseiten, sowie folgender anderer Datensätze, wurde die Annotationssoftware CVAT [2] verwendet.

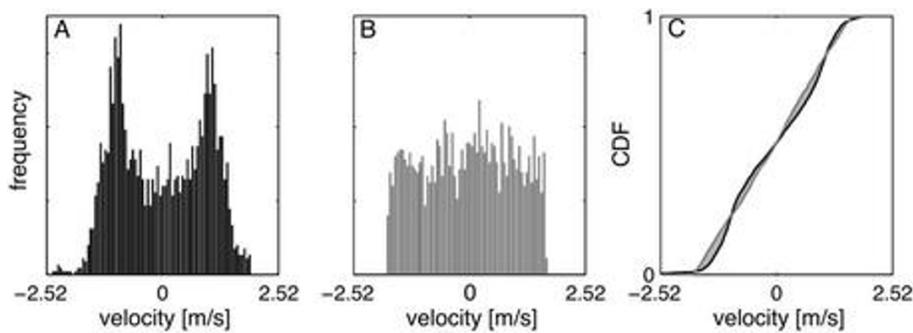


Figure 5: Panel A shows a velocity profile - histogram (h_A ; black) of the

Abbildung 3.1: Beispiel kontextbedingter Gruppierung wegen gemeinsamer Y-Achsenbeschreibung eines gemischten Diagrammtyps (Histogramm und Liniendiagramm)

Da der Datensatz aus einer beträchtlich diversen Menge verschiedener wissenschaftlicher Publikationen besteht, beinhalten diese auch zahlreich verschiedene Diagrammlayouts. Um eine bestmögliche Konsistenz und Nützlichkeit in der Handannotation zu gewährleisten, wurden einige Überlegungen gemacht: Da einige Abbildungen als Gruppe von Diagrammen fungieren, siehe Abbildung 3.1, muss die generelle Entscheidung getroffen werden, jedes Diagramm der Gruppe einzeln zu annotieren oder lediglich die gesamte Gruppe zusammen. Beide Möglichkeiten liefern Vor- und Nachteile; beim getrennten Annotieren muss die Gruppe in einem späteren Schritt nicht mehr in die einzelnen Diagramme aufgeteilt werden, jedoch können auch kontextbedingte Informationen verloren gehen, wie in dem abgebildeten Beispiel die Y-Achsenbeschreibung des mittleren Diagramms (B), welches sich eine gemeinsame Y-Achsenbeschriftung mit dem linken Diagramm (A) teilt. Ebenfalls können Diagrammgruppen aus verschiedenen Diagrammtypen bestehen, etwa Histogramme und Liniendiagramme beieinander, weswegen dementsprechend für genau diesen Fall die gemischte Diagrammkategorie eingeführt wurde. Bei weiteren Unklarheiten des Gruppenumfangs wurde sich sonst immer an die darunterliegende Abbildungsunterschrift gehalten.

Insgesamt wurden 321 Seiten annotiert, beinhaltend 105 Liniendiagramme, 115 Balkendiagramme (vereinigt mit Histogrammen), 79 sonstige und 66 gemischte Diagramme.

3.1.2 Datensatz historischer Wirtschaftsscans zur Objekterkennung

Die Scans der geschichtlichen Wirtschaftsmagazine wurden mit ähnlichen Überlegungen annotiert. Hier befinden sich ebenfalls Diagrammgruppen, teilweise auch mit mehreren verschiedenen Diagrammtypen, siehe Abbildung 3.2, welche alle wieder als gesamte Gruppe annotiert wurden. Bis auf sehr wenige Ausnahmen befinden sich alle Abbildungen in den Scans visuell eingeraumt. Da die Ausrichtung dieser jedoch nie wirklich perfekt gerade dargestellt wurde, und somit, der Ausrichtung verschuldet, kein Annotationsrechteck mit ausgeschlossenem Abbildungsrahmen gezeichnet werden kann, wurde die Entscheidung getroffen, jede Annotation mit allen Ecken der Diagrammrahmen zu beinhalten. Grundsätzlich wurden alle Abbildungen, Diagramme oder nicht, wie etwa vereinzelte Karikaturen oder Landkarten mit in die Klasse der sonstigen Diagramme eingeschlossen, um so die allgemeine Erkennung von seltenen Diagrammtypen zu verstärken. Es wurden insgesamt 2391 zufällige Seiten ausgewählt und manuell annotiert, woraus sich 343 Liniendiagramme, 102 Balkendiagramme, 77 sonstige und 52 gemischte Diagramme ergaben.



Abbildung 3.2: Diagrammbeispiel historischer Scans

3.2 Schwierigkeitsklassifizierung von Liniendiagrammen

Aufgrund der überwiegend vorkommenden Liniendiagramme in den historischen Wirtschaftsscans wurde sich im Folgenden primär auf die Auswertung der Liniendiagramme fokussiert.

Für genau diese Auswertung wurde der Vorverarbeitungsschritt überlegt, die extrahierten Liniendiagramme in verschiedene Untergruppen zu unterteilen. Es wurden vier Klassifikationen gewählt; Liniendiagramme mit nur einer Wertelinie, aus zusammengesetzten Diagrammen, also Liniendiagrammsgruppen, sich nicht überlappenden Wertelinien und sich überlappenden Wertelinien.

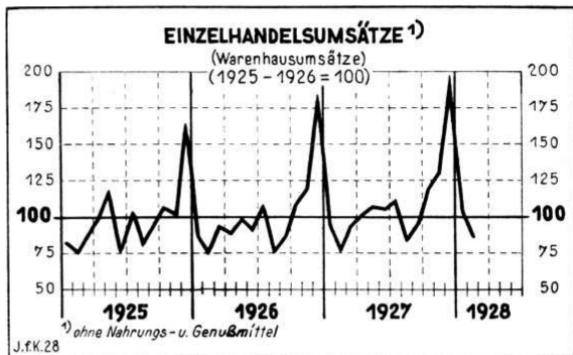


Abbildung 3.3: Liniendiagramm mit einer Wertelinie

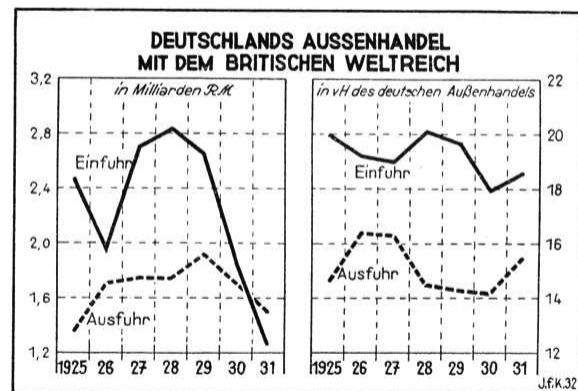


Abbildung 3.4: Zusammengesetzte Liniendiagrammsgruppe

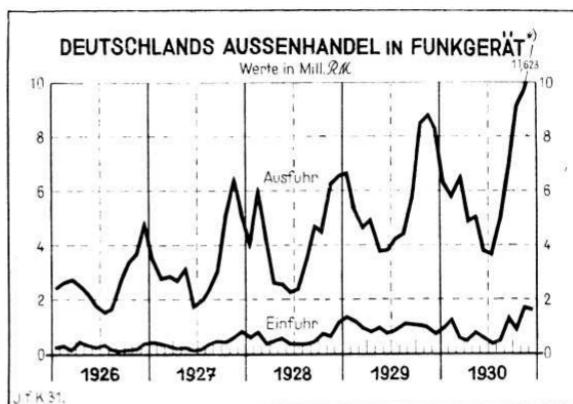


Abbildung 3.5: Liniendiagramm mit sich nicht überlappenden Wertelinien

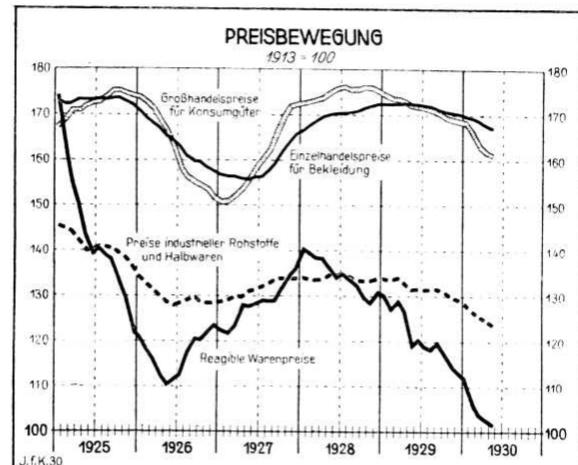


Abbildung 3.6: Liniendiagramm mit sich überlappenden Wertelinien

Es existieren nämlich Liniendiagrammsgruppen mit mehreren eigenständigen Unterdiagrammen, welche möglicherweise jeweils ihre eigene Achsenbeschreibung haben, oder auch sich kontextbedingt diese Achsenbeschriftungen teilen. Diese müssen also im Vergleich zu

einfachen Liniendiagrammen speziell behandelt werden. Aber auch wenn für Liniendiagramme mit nur einer oder sich nicht überschneidenden Wertelinien ein eher primitiver Extraktionsalgorithmus ausreichen würde, tritt bei komplexeren, sich überlappenden oder überschneidenden Wertelinien schnell das Problem der Linientrennung bzw. Liniengruppierung auf.

Der erstellte Datensatz für die Schwierigkeitsklassifizierung besteht aus 807 klassifizierten Liniendiagrammen, unterteilt auf 93 mit einer Wertelinie, 284 zusammengesetzte, 94 nicht überlappende und 336 überlappende Liniendiagramme.

Eine zweite Version des Datensatzes wurde ebenfalls erstellt. Bei dieser wurde aufgrund späterer Klassifizierungsprobleme von sich nicht überlappenden mit überlappenden Wertelinien beide Klassen in die gemeinsame Differenzierung der Liniendiagramme mit mehreren Wertelinien vereinigt. Dementsprechend besteht die zweite Version des Datensatzes aus 430 Instanzen dieser Klasse.

3.3 Auswertung von Liniendiagrammen

Für die Auswertung der historischen Liniendiagramme wurden Überlegungen gemacht, Beschriftungen und vor allem das Hintergrundgitter, welches sich in jedem Diagramm zu finden lässt, zu entfernen, jedoch wurde schnell klar, dass diese primitive Herangehensweise grundsätzlich eher impraktikabel ist. Zum einen führen die nicht genau senkrecht und waagerecht verlaufenden Gitterlinien die korrekte Erkennung dieser zu einem nichttrivialen Erkennungsproblem und zum anderen überlappen und verlaufen viele Wertelinien auf dem Gitter, sodass die einfache Entfernung der Gitterlinienpixel das Diagramm mit unzähligen Lücken verbleiben lässt. Dementsprechend wurde beschlossen, statt aus dem Diagramm alles bis auf die Wertelinien zu entfernen, die Wertelinien selbst zu extrahieren, also sie durch Segmentation vom Hintergrundgitter und allen anderen Elementen zu trennen.

Die manuelle Erstellung der Grundwahrheiten für die Werteliensegmentation ist allerdings recht arbeitsaufwendig, weswegen zusätzlich ein synthetisch erstellter Datensatz generiert wurde, bei dem die Erstellung von Binärmasken der Wertelinien trivial ausfällt. Im Folgenden wird die Datensatzerstellung für die sowohl semantische Segmentation als auch Instanzsegmentation beschrieben. Die semantische Segmentation benötigt pro Klasse nur eine gemeinsame Binärmaske, unabhängig von der Anzahl der Objekte, also in dem Fall der Werteliensegmentation eine Maske pro Liniendiagramm. In dem Fall der Instanzsegmentation dagegen wird nicht nur eine eigene Maske pro jeweiliges Objektaufkommen - pro Objektinstanz - erforderlich.

3.3.1 Datensatz von synthetischen Liniendiagrammen zur Segmentation

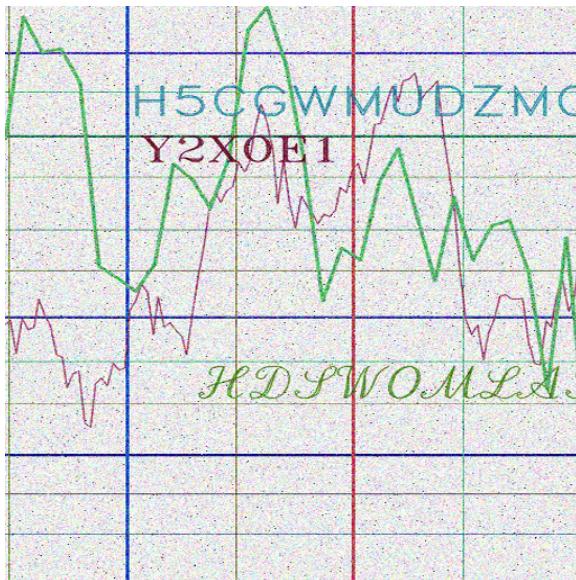


Abbildung 3.7: Synthetisch erstelltes Liniendiagramm

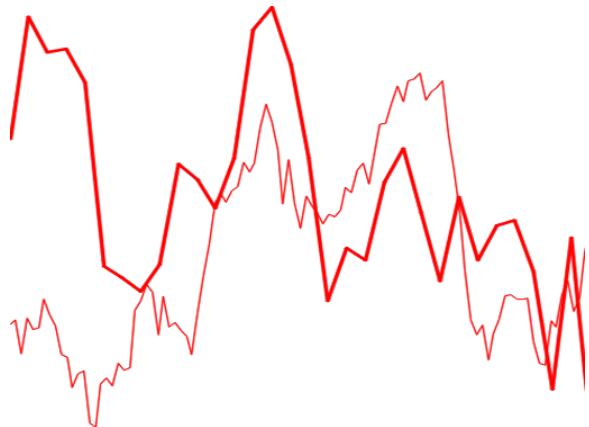


Abbildung 3.8: Zugehörige generierte Binärmaske der Wertelinien

Der synthetische Datensatz besteht aus 2000 verschiedenen, zufällig generierten Liniendiagrammen. Diese beinhalten zufällige Wertelinien und Gitterlinien, sowohl in Position als auch in Liniendicke und beliebige, teils den Wertelinien überlappende, Textbeschriftungen vielfältiger Schriftgrößen und Schriftarten. Da beim Generierungsprozess alle Diagrammwerte natürlicherweise bekannt sind, können diese einfach auf einem zweiten, leeren Bild übertragen werden, um so die zugehörige Wertelinien-Binärmaske der semantischen Segmentation zu erstellen. Für die Instanzsegmentation dagegen können diese auf getrennte Bilder gezeichnet werden. Je nach Implementation werden oftmals auch keine Binärmasken bei der Instanzsegmentation verwendet, sondern stattdessen Annotationen im Format von Polygonumrissen. Ist dies der Fall, können die getrennten Wertelinien-Binärmasken unter anderem mit Hilfe von Konturenerkennung in das gewünschte Polygonannotationsformat gebracht werden. Nachbearbeitet wurden die generierten Liniendiagramme am Ende mit unterschiedlichem Bildrauschen, um so näher an die Scanqualität und Diversität der historischen Diagramme herankommen zu können.

3.3.2 Datensatz von historischen Liniendiagrammen zur Segmentation



Abbildung 3.9: Historisches Liniendiagramm

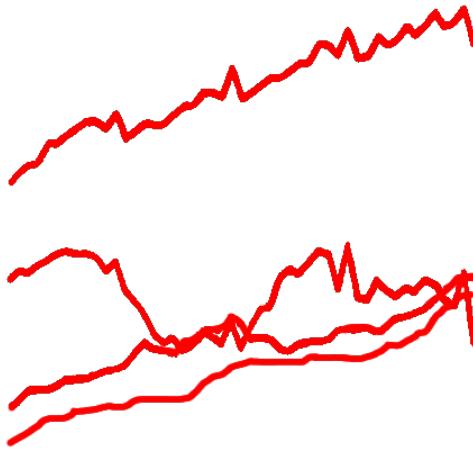


Abbildung 3.10: Zugehörige manuell erstellte Binärmasken der Wertelinien

Die manuelle Erstellung der Wertelinien-Binärmasken der historischen Liniendiagramme fällt dagegen nicht ganz so leicht aus. Für die Anfertigung der Werteliengrundwahrheiten wurde das Originalbild in einem Bildbearbeitungsprogramm [3] geöffnet und pro Wertelinie eine eigene Bildschicht (layer) hinzugefügt. In jeder dieser einzelnen Schichten kann dann die jeweilige Wertelinie überzeichnet - abgepaust - werden. Der Grund, jede Wertelinie in ihre eigene Maskenschicht zu übertragen, ist der, dass am Ende einfach alle Schichten getrennt exportiert werden können. Für die semantische Segmentation ist dies allerdings nicht nötig, da pro Klasse nur eine gemeinsame Binärmaske verwendet wird. Hier können jedoch dann ganz einfach alle exportierten Schichten in eine gemeinsame Maske vereinigt werden. In dem Fall der Instanzsegmentation dagegen wird eben nicht nur eine vereinigte Binärmaske pro Klasse benötigt, sondern eine eigene Maske pro Objektinstanz - pro Wertelinie. Dementsprechend können hierfür die getrennten Maskenschichten verwendet werden. Werden hierfür wieder die Grundwahrheiten im Polygonannotationsformat erforderlich, können diese, wie zuvor beschrieben, durch Konturenerkennung von der Binärmaske konvertiert werden.

Kapitel 4

Implementation

Die Implementation der verschiedenen Deep-Learning-Bilderkennungsmethoden erfolgte durch die Verwendung des Ultralytics YOLO [4] Frameworks und der Eigenimplementati-
on der U-Net [5] Architektur. Alle folgenden Modelle wurden auf einer NVIDIA GeForce RTX 3090 mit 24252 MiB Grafikkartenspeicher trainiert.

4.1 Ultralytics YOLO

Für die Extraktion von Diagrammen aus Texten, Schwierigkeitsklassifizierung von Liniendiagrammen und Instanzsegmentation der Wertelinien wurde das Ultralytics YOLO Framework verwendet. Es basiert auf der YOLO (You Only Look Once) Architektur, welche erstmals 2015 [6] veröffentlicht wurde und seitdem zehn Versionsiterationen durchlief. Unterstützt werden verschiedene Bild- und Videoerkennungsaufgaben, wie die Erkennung (detection), Segmentierung (segmentation), Posenschätzung (pose detection), Verfolgung (tracking) und Klassifizierung (classification). Das Ultralytics YOLO Framework ist anfängerfreundlich, die Verwendung erfolgt einfach, verfügt man bereits über einen annotierten Datensatz, kann mit lediglich einem Konsolenbefehl der Trainingsprozess des eigenen Modells gestartet werden. Ebenfalls verfügt es über die automatische Datenaugmentation während des Trainingsvorgangs und die Evaluation verschiedener Metriken des trainie-
renden und trainierten Modells.

Ultralytics stellt zu dem Großteil der zehn YOLO Architekturen bereits vortrainierte herunterladbare Modelle bereit. Für das Vortrainieren der Objekterkennung, Klassifizierung und Instanzsegmentation wurde der COCO [7] Datensatz verwendet, welcher aus über 200.000 Bildern besteht und in 80 Objektklassen eingeteilt wurde.

Diese werden außerdem in verschiedene Modellgrößen angeboten, sodass die Möglichkeit besteht, zwischen Invarianzgeschwindigkeit, benötigter Gleitkommaoperationsleistungsfähigkeit und verwendbarem Grafikkartenspeicher abwägen zu können.

Es wurden verschiedene YOLO Modelle verwendet, an denen jedoch keine Architekturänderungen vorgenommen wurden. Sofern im Weiteren nicht explizit angegeben, wurden die Grundeinstellungen des Ultralytics YOLO Frameworks benutzt. Die genaue Anzahl der Trainingsepochen variierte zwischen den einzelnen Trainingsvorgängen, da bei allen die Geduldseinstellung (patience) von 100 Epochen verwendet wurde. Um die Überanpassung (over-fitting) an den Trainingsdatensatz zu vermeiden, wird mit diesem Geduldsparameter das Training vorzeitig abgebrochen, solange in den letzten beliebigen Epochen das trainierende Modell keine Verbesserungen der Validationsmetriken aufweisen konnte.

4.1.1 Objekterkennung zur Extraktion von Diagrammen aus Texten

Für die Erkennung aller Diagramme in den Vollseitscans wurde das YOLov9e Modell verwendet.

Zuerst wurde es auf den in 3.1.1 beschriebenen, selbst erstellten DocBank Datensatz vortrainiert (pre-trained) und danach auf den Datensatz aus 3.1.2, der historischen Wirtschaftsscans, feintrainiert (fine-tuned). Beide Datensätze wurden mit der häufig verwendeten 80-20 Aufteilung differenziert. In diesem Fall werden 80% des Datensatzes für das Trainieren und 20% für das Evaluieren des Modells benutzt. Die hierbei 20% des Evaluationssets bestehen aus Daten, welche das Modell vor dem Zeitpunkt noch nie gesehen hat und dementsprechend unbekannt sind.

Es wurde die Bildgröße von 1024x1024 Pixel verwendet und eine Batchgröße von 7 gewählt, da für eine höhere Batchgröße die verwendete Grafikkarte über ungenügend viel Speicher verfügte. Die von den Grundeinstellungen geänderten Parameter der Datenaugmentierung, mitsamt ihrer jeweiligen Wahrscheinlichkeiten, bestanden aus:

- Farbmanipulation im HSV-Farbraum: Hue (1.0), Sättigung (1.0) und Helligkeit (0.5)
- BGR-Kanaländerung (0.5)
- Skalierung (1.0)
- Vertikales Spiegeln (0.1)
- Überlagern von Bildern (Mixup, 1.0)

Gewählt wurden diese Augmentierungstechniken mit dem Ziel, die universellen Erkennungsfähigkeiten des trainierten Modells zu stärken. Farbmanipulationen helfen dem Modell, zuversichtlichere Vorhersagen auf einer breiten Spanne unterschiedlicher Papierhintergründe zu treffen beziehungsweise Diagramme mit willkürlichen Farben besser zu erkennen.

4.1.2 Klassifizierung zur Schwierigkeitsbestimmung von Liniendiagrammen

Der Trainingsvorgang der Schwierigkeitsklassifizierung der Liniendiagramme folgte ähnlich der Objekterkennung aus 4.1.1. Da hier das gewählte YOLOv8m-cls Modell nur Voraussagen über bereits extrahierte Diagramme machen muss und nicht mehr über Vollseitescans, wurde die Bildgröße auf 640x640 Pixel reduziert. Im Vergleich zu der vorherigen Objekterkennung konnten keine Verbesserungen der Verwendung eines größeren Modells beobachtet werden, weshalb für die Klassifizierung lediglich das Modell mittlerer Größe gewählt wurde. Die Batchgröße dagegen wurde auf 16 erhöht und ebenfalls wurde wieder der Trainingsprozess durch den Geduldsparameter von 100 beendet, anstatt durch eine festgelegte Epochenbegrenzung. Die veränderten Augmentierungstechniken sowie deren Parameter bestanden aus:

- Farbmanipulation im HSV-Farbraum: Hue (1.0), Sättigung (1.0) und Helligkeit (0.5)
- BGR-Kanaländerung (0.5)
- Vertikales Spiegeln (0.1)
- Rotation ($\pm 5^\circ$)
- Scherung ($\pm 5^\circ$)
- Zufälliges Löschen (1.0)

Auch wenn das Löschen zufälliger Bereiche im Bild möglicherweise kontraproduktiv wirkt, erzielte die Verwendung dieser Technik bessere Ergebnisse, weshalb sie mit einbezogen wurde. Die restlichen Augmentierungen wurden für die breitere Diagrammvarianz gewählt.

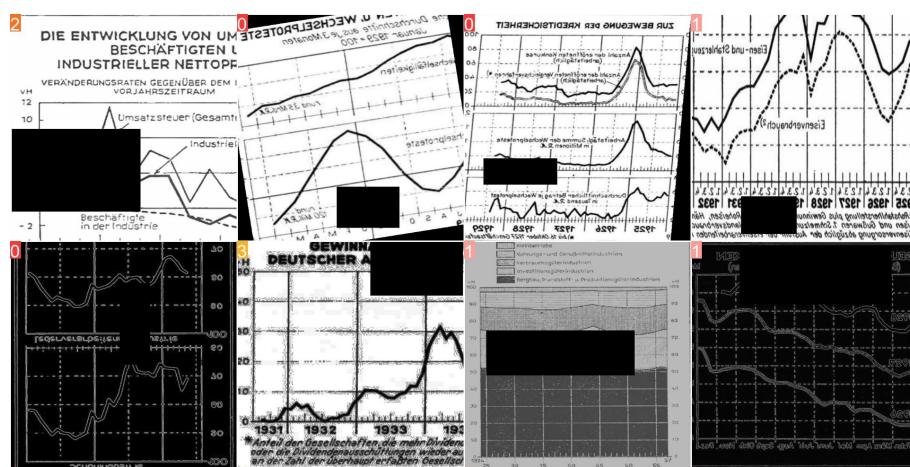


Abbildung 4.1: Augmentierter Trainingsbatch der Klassifizierung

Des Weiteren wurde dieser beschriebene Trainingsprozess erneut auf der anderen Version des in 3.2 beschriebenen Datensatzes mit der Vereinigung der sich überlappenden und

nicht überlappenden Wertelinien in die gemeinsame Klasse mehrerer Wertelinien trainiert.

4.1.3 Instanzsegmentation von Wertelinien in Liniendiagrammen

Die Instanzsegmentation durch Ultralytics YOLO erfolgte mithilfe der in 3.3.1 und 3.3.2 definierten Datensätze der Wertelinien synthetischer und historischer Liniendiagramme. Wie zuvor beschrieben, wurden diese zuerst in das vom Framework benötigte Polygonannotationsformat konvertiert. Hierzu wurde Python 3.10.12 in Verbindung mit der Bildverarbeitungsbibliothek OpenCV [8] verwendet. Die Korrektheit der Konvertierung von den Binärmasken in Polygonform wurde durch die von Ultralytics erstellten Trainingsbatches als auch durch Auswertung mithilfe der Annotationswebseite Roboflow [9] überprüft. Im Anschluss konnte das YOLOv8x-seg Modell mit einer Bildgröße von 512x512 Pixel und Batchgröße von 16 trainiert werden. Der Trainingsprozess verlief erneut in zwei Schritten. Diese Parameter wurden für sowohl das Vortrainieren auf den synthetischen als auch das Feintrainieren auf den historischen Liniendiagrammen gewählt. Ebenfalls wurden für beide Trainingsprozesse dieselben Augmentierungstechniken verwendet:

- Farbmanipulation im HSV-Farbraum: Hue (1.0), Sättigung (1.0) und Helligkeit (0.5)
- BGR-Kanaländerung (0.5)
- Vertikales Spiegeln (0.5)
- Rotation ($\pm 90^\circ$)
- Zufälliges Löschen (0.7)
- Überlagern von Bildern (Mixup, 0.5)

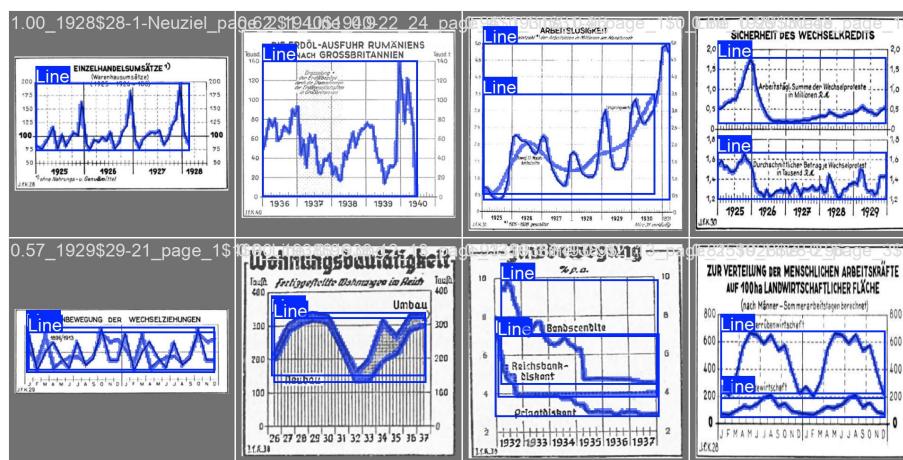


Abbildung 4.2: Annotierte Wertelinien der Liniendiagramme zur Instanzsegmentation

Hier wurden stärkere Augmentierungstechniken gewählt, da Wertelinien unabhängig von Positionskontexten ausfindig gemacht werden müssen, weshalb zusätzlich höhere Rotations- und Spiegelungsparameter verwendet wurden.

4.2 U-Net: Semantische Segmentation von Wertelinien

Als Alternative zur Instanzsegmentation wurde ebenfalls die semantische Segmentation durch die Eigenimplementation der U-Net Architektur verwirklicht. Diese wurde speziell für die medizinische Bildsegmentierung entwickelt und zeichnet sich besonders bei einer limitierten Anzahl von Trainingsdaten aus. Die Architektur des CNNs (convolutional neural network) kombiniert Encoder- und Decoderpfade mit Skip-Verbindungen, um so präzise Segmentationsergebnisse zu erzielen.

Implementiert wurde die Datenvorverarbeitung, das Modelltraining und die Evaluation mithilfe der Keras 3 API [10]. Für die Augmentierung der Trainingsdaten wurde die Bibliothek Albumentations [11] genutzt. Lediglich der Code der U-Net Architektur selbst wurde in Form des Vanilla U-Nets [12] übernommen.

Der Ablauf des U-Net Trainings durchlief wie folgt: Zuerst wurde der Datensatz mithilfe verfügbarer Bibliotheksfunktionen eingelesen. Im Fall der Binärmasken wurde der Schwellenwert (threshold) von 30% genutzt. Nach der Deklaration der Modellarchitektur wurde die Datenaugmentation definiert. Folgende Augmentierungstechniken wurden verwendet:

- Spiegeln entlang der horizontalen Achse (`A.HorizontalFlip, p=0.5`)
- Spiegeln entlang der vertikalen Achse (`A.VerticalFlip, p=0.5`)
- Drehung des Bildes (`A.RandomRotate90, p=0.5`)
- Vertauschen von Breite und Höhe (`ATranspose, p=0.5`)
- Ausschneiden und Skalieren auf eine feste Größe (`A.RandomResizedCrop, height=512, width=512, scale=(0.8, 1.0), p=0.5`)
- Zuschneiden auf eine feste Größe vom Zentrum aus (`A.CenterCrop, height=512, width=512, p=0.5`)
- Ausschneiden eines Bereichs des Bildes basierend auf der Maske (`A.CropNonEmptyMaskIfExists, height=512, width=512, p=0.5`)
- Zufälliges Löschen von Bereichen (`A.CoarseDropout, max_holes=8, max_height=64, max_width=64, p=0.5`)
- Anpassung von Helligkeit und Kontrast (`A.RandomBrightnessContrast, p=0.5`)
- Anwendung von Gamma-Korrektur (`A.RandomGamma, p=0.5`)
- Zufälliges Mischen der Farbkanäle (`A.ChannelShuffle, p=0.5`)

- Umkehren der Pixelwerte (A.Solarize, threshold=128, p=0.1)
- Vollständiges Invertieren der Farbwerte (A.InvertImg, p=0.1)
- Skalieren auf die maximale Größe entlang der längsten Kante (A.LongestAxisSize, max_size=512, p=1.0)
- Auffüllen auf eine Mindestgröße mit konstantem Rand (A.PadIfNeeded, min_height=512, min_width=512, border_mode=cv2.BORDER_CONSTANT, value=0, p=1.0)
- Überlagern mit einem anderen Bild (A.MixUp, p=1.0, alpha=0.7)

Es wurde eingerichtet, dass die Trainingsbilder samt ihrer zugehörigen Binärmasken jeweils pro Batch neu augmentiert werden. Der Validationsdatensatz dagegen wurde unaugmentiert verarbeitet. Durch Abspeichern der einzelnen Bilderbatches konnte der fehlerfreie Augmentierungsablauf manuell verifiziert werden.

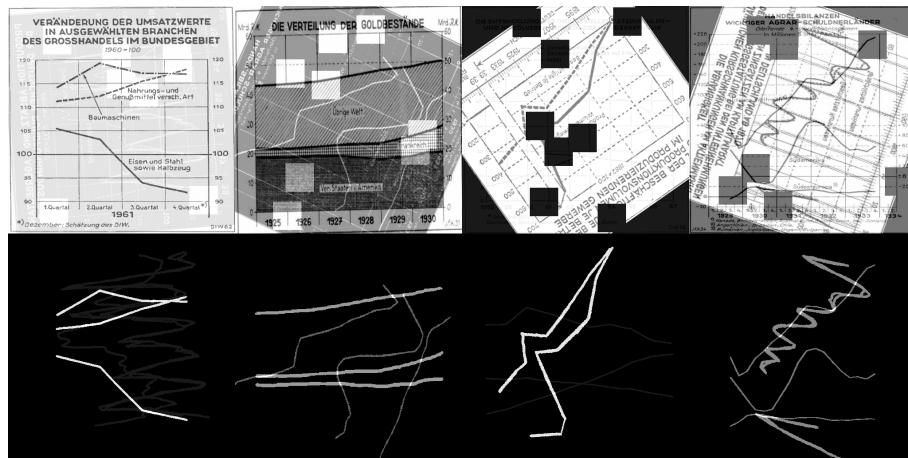


Abbildung 4.3: Augmentierter Trainingsbatch inklusive der zugehörigen Binärmasken

Nach diesem Implementationsschritt konnte der Trainingsprozess gestartet werden. Hierfür wurde ebenfalls die für die Grafikkarte größtmögliche Batchgröße von 7 verwendet und ein Geduldsparameter von 100 Epochen genutzt, welcher das Training zum frühzeitigen Abbrechen (early stopping) führt. Als Überwachungsmetrik (monitor) dafür wurde der Trainingsverlust (training loss) gewählt; sobald dieser über die bestimmte Anzahl an Epochen nicht unterschritten wurde, beendet das Training vorzeitig. Aufgrund der Minimierungsoptimierung des verwendeten Adam Optimierers (optimizer) [13] wurde dieser Trainings Loss als Negativ des Dice-Sørensen-Koeffizient (dice score) festgelegt, welcher ebenfalls als Validationsfunktion dient. Der Dice Score misst die Überlappung zwischen der Vorhersage und der Ground Truth, wobei ein Wert von 1 eine perfekte Übereinstimmung und 0 keine Übereinstimmung bedeutet. Als Lernrate (learning rate) wurde 0.0001

und als Dropout-Rate 0.0 gewählt. Für die genannten Validations- und Verlustfunktionen wurden folgende Formeln verwendet:

$$\text{Dice Score}(y_{\text{true}}, y_{\text{pred}}) = \frac{2 \cdot \sum_i (y_{\text{true}_i} \cdot y_{\text{pred}_i}) + 1}{\sum_i y_{\text{true}_i} + \sum_i y_{\text{pred}_i} + 1}$$

$$\text{Dice Loss}(y_{\text{true}}, y_{\text{pred}}) = -\text{Dice Score}(y_{\text{true}}, y_{\text{pred}})$$

Hierbei steht y_{true} für die Grundwahrheiten und y_{pred} für die Modellvorhersagen. Der Index i läuft über alle Pixel der Bilder. Die Addition von 1 im Zähler und Nenner dient der numerischen Stabilität.

4.3 Numerische Auswertung von Liniendiagrammen in Tabellenform

Um die historischen Liniendiagramme in Tabellenform auswerten zu können, wurde die vorherig beschriebene U-Net Segmentierung mitbenutzt. Die generelle Idee des Auswertungsalgorithmus besteht aus der Kombination der durch die implementierte Segmentierung möglich gemachten Wertelinienerkennung und einer optischen Schriftzeichenerkennung (optical character recognition; OCR), aus welcher unter anderem Achsenbeschriftungen und deren numerischer Kontext extrahiert werden können. Umgesetzt wurde dieser Algorithmus ebenfalls in der Programmiersprache Python. Im Folgenden wird die Funktionsweise des zweigeteilten Algorithmus im Näheren beschrieben.

4.3.1 Extraktion der segmentierten Wertelinien

Aufgrund der später sichtlich gemachten souveräneren Ergebnisse des U-Nets im Vergleich mit dem Ultralytics YOLO Segmentierungsmodell wurde dieses für die Wertelinienextraktion verwendet. Allerdings verfügt das U-Net lediglich über die semantische Segmentierung, weshalb dies im Kontrast zu der Instanzsegmentation ein nicht triviales Hindernis aufwirft: Die semantische Segmentation kann nicht zwischen mehreren Wertelinien unterscheiden. Überschneiden oder überlappen sich diese, können die erkannten Linien vom Modell aus nicht differenziert werden. Dementsprechend muss die Wertelinientrennung in der Nachverarbeitung bewältigt werden.

Der erste Schritt der Wertelinienextraktion ist die semantisch segmentierte Binärmaskenvorhersage des U-Nets zu skelettieren (skeletonize). Sowie alle anderen folgenden Bildbearbeitungsfunktionen wurde dieses Ausdünnen der Linien mit Hilfe der OpenCV Bibliothek erledigt. Um mögliche kleine Lücken zu füllen, durchläuft die resultierende Maske eine

schließende Morphologie (closing morphology) mit einer Kerngröße (kernel size) von 3x3 Pixel.



Abbildung 4.4: Zu auswertendes Liniendiagramm

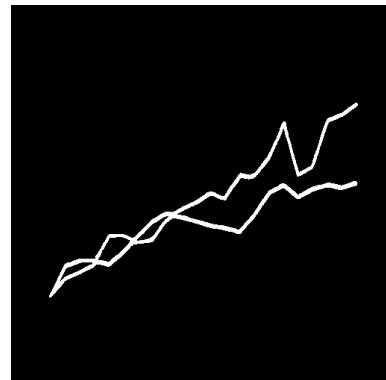


Abbildung 4.5: Semantische U-Net Vorhersage

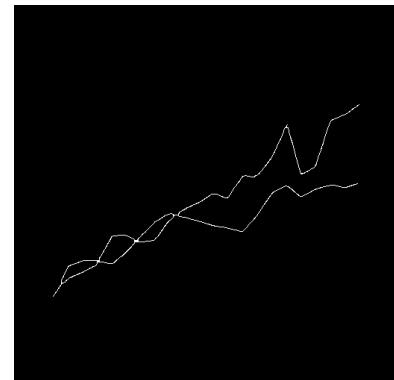


Abbildung 4.6: Skelletierte Binärmasken

Um das Problem der Linientrennung zu überwinden, wurde die Entscheidung getroffen, einen primitiven Algorithmus zu verwenden, welcher die Wertelinien anhand ihrer räumlichen Position im Diagramm willkürlich differenziert. Im Folgenden werden alle sich überschneidenden Wertelinien nach dieser Approximation getrennt. Dementsprechend wurde das anspruchsvolle Problem der Wertelinientrennung im Weiteren aufgrund seiner Komplexität und des begrenzten Rahmens dieser Arbeit nicht weiter eingehend behandelt.

Dieser Linientrennungsalgorithmus funktioniert wie folgt: Zuerst wird die gesamte Anzahl der Wertelinien ermittelt. Dazu wird für jede vertikale Pixelspalte des Bildes das Aufkommen von nicht-schwarzen Pixeln gezählt. Da im Fall von vertikal verlaufenden Wertelinien mehrere von diesen aufeinanderfolgen können, muss eine Gruppe an sequentiellen nicht-schwarzen Pixeln als nur ein Aufkommen gewertet werden. Die berechneten Werte pro Pixelspalte werden daraufhin der Aufkommensgröße nach sortiert und die größten 5% entfernt. Diese enthalten mögliche fehlerbehaftete Werte aufgrund eventueller Artefakte in der Werteliensegmentation. Anschließend wird der verbleibende Maximalwert als gesamte Linienanzahl definiert.

Nach Erhalt dieser berechneten Linienanzahl des Diagramms werden erneut alle Pixelspalten der vorhergesagten Binärmaske durchlaufen. Ebenfalls wird wieder die Anzahl der aufkommenden Wertelinien gezählt. Besitzt die jeweilige Pixelspalte nun dieselbe Anzahl an Linien wie die berechnete gesamte Linienanzahl, dann wird die *i-te* sequentielle Gruppe an nicht-schwarzen Pixeln der *i-ten disjunkten Wertelinienbinärmaske* zugeordnet. Sollte dies allerdings nicht der Fall sein, wird die gesamte Pixelspalte in eine *gemeinsam genutzte Binärmaske* übertragen.

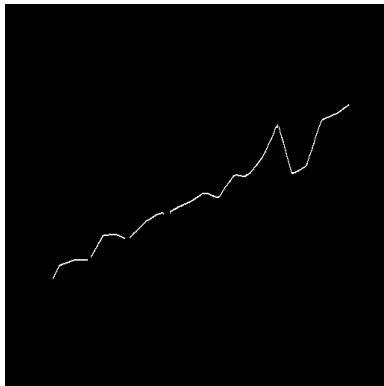


Abbildung 4.7: Disjunkte Binärmaske der ersten Linie

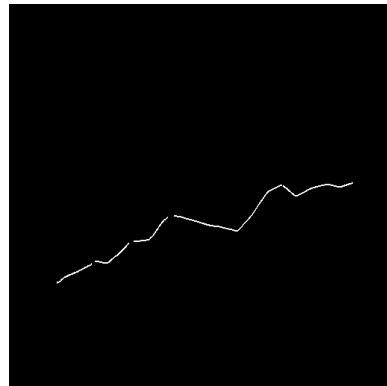


Abbildung 4.8: Disjunkte Binärmaske der zweiten Linie



Abbildung 4.9: Gemeinsam genutzte Binärmaske

Somit können die Wertelinien anhand ihrer Position differenziert werden. Um nun die einzelnen kontinuierlichen Linienwerte zu extrahieren, müssen die disjunkteten Binärmasken lediglich jeweils mit der gemeinsam genutzten Maske bitweise vereinigt werden. Durch die OpenCV Funktion des Auffindens verbundener Komponenten (connected components) wird anschließend die räumlich längste ausgewählt und der Mittelwert aller nicht-schwarzen Pixel pro Pixelspalte dieser Komponente berechnet. Um möglichem Rauschen entgegenzuwirken, wird nach Gaußscher Weichzeichnung (gaussian blur) mit einem Kern von 5x5 Pixel die Positionssequenz jeder getrennten Wertelinie im Diagramm ermittelt.

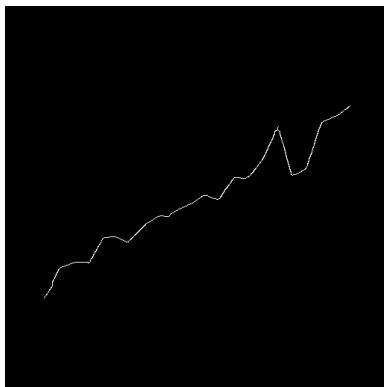


Abbildung 4.10: Vereinigte Binärmaske der ersten Linie

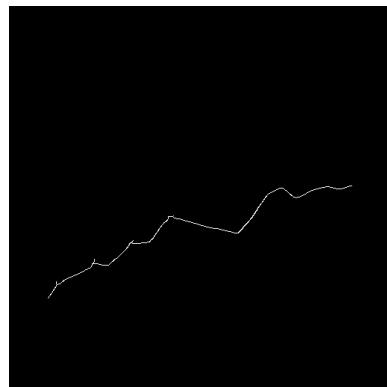


Abbildung 4.11: Vereinigte Binärmaske der zweiten Linie



Abbildung 4.12: Getrennte Wertelinien

4.3.2 Achsenerkennung durch optische Schriftzeichenerkennung

Der andere Teil der Liniendiagrammsauswertung setzt sich aus optischer Schriftzeichenerkennung zusammen, aus welcher Titel, Linienbezeichnungen und Achsenbeschriftungen (label) ausgearbeitet werden können. Für die Label der Achsen wird außerdem deren Mittelpunktposition im Diagramm für die numerische Tabellenformauswertung verwendet.

Bis auf limitierte Ausnahmen befinden sich so alle Datenpunkte, beispielsweise bei Diagrammen über Jahresentwicklungen, in der Jahresmitte.

Für die optische Schriftzeichenerkennung wurde die JavaScript Bibliothek Chrome Lens OCR [14] verwendet, während der restliche Algorithmus weiterhin in Python implementiert wurde. Diese OCR-Bibliothek verwendet die im Chromium Browser eingebettete Google Lens OCR und kann ohne weiteren Authentifizierungsprozess genutzt werden, weshalb sich für sie entschieden wurde. Die Brücke zwischen Python und JavaScript erfolgt durch das programmatische Aufrufen des JavaScript Codes über die Konsole in Python, welcher daraufhin von ihr in Textform ausgelesen und verarbeitet werden kann. Zur Verfügung gestellt werden die erkannten Textbezeichnungen selbst, aber auch ihre jeweiligen absoluten und relativen Positionen innerhalb des Diagrammbilds.

Nachdem alle erkannten OCR-Labels bereitgestellt wurden, werden diese mithilfe ihrer Mittelpunktpositionen in fünf disjunkte Gruppen eingeteilt: Titel, Linienbezeichnungen, linke Y-Achse, rechte Y-Achse und untere X-Achse.

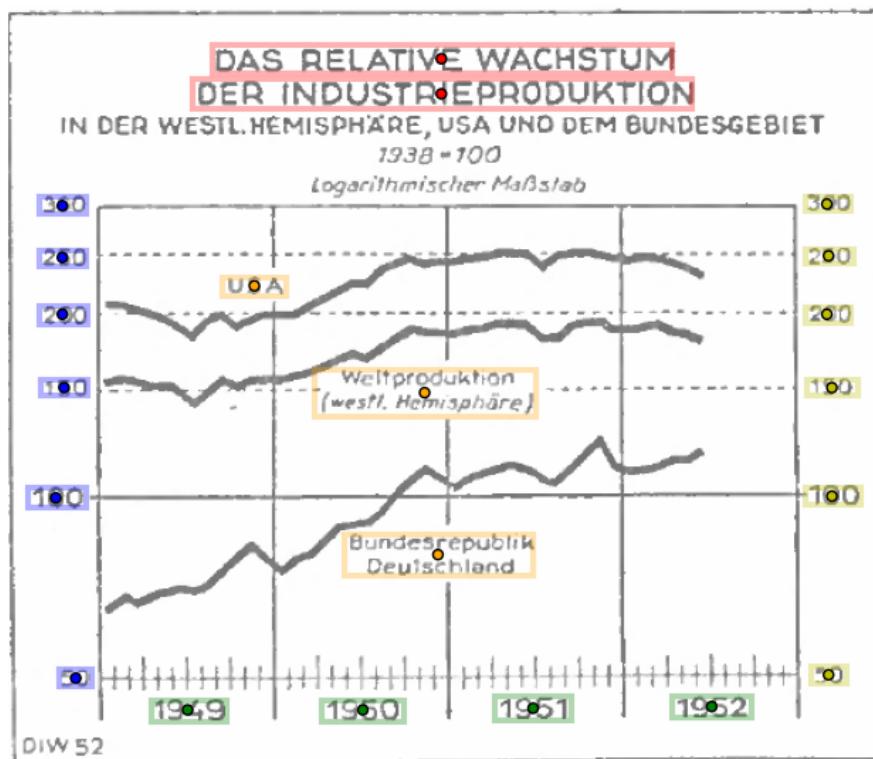


Abbildung 4.13: Zuordnung der erkannten Textbeschriftungen in die fünf disjunkten Gruppen samt ihrer jeweiligen Mittelpunktpositionen

Für die drei Achsenbeschriftungen werden nur jene Labels untersucht, welche sich auf dem äußeren Rand von 15% des Bildes befinden. Die horizontale und vertikale Mittelpunktposition aller Labels für die drei Achsen wird bestimmt und mit dieser werden mögliche

Beschriftungen, welche sich nicht innerhalb 2% der jeweiligen Mittelwertlinie befinden, aussortiert. Ebenfalls werden alle Beschriftungen unterhalb der erkannten X-Achse, unabhängig dieser vorherigen Einschränkung, ignoriert. Um mögliche Schriftzeichenerkennungsfehler entgegenzuwirken, wird der Text der erkannten Achsenlabels gefiltert. Erkannte Kommas werden in Punkte verwandelt und danach alle Buchstaben und andere Zeichen entfernt, sodass der Text am Ende nur noch aus den Zeichen 0-9, Punkten und Minuszeichen besteht. Diese Entscheidung wurde getroffen, um fälschlicherweise miterkannte Einheitsangaben oder andere OCR-Fehler zu verbessern. Das Aufkommen von falscher OCR bei Ziffern, z.B. die Erkennung von einem B statt einer 8, kam nur sehr selten vor, weshalb dieser Schritt gewählt wurde. Um den Diagrammtitel zu extrahieren, werden die oberen Labels ausgewählt, die sich in horizontaler Mitte befinden. Für die Wertelinienbeschriftungen werden alle Textbezeichnungen in Auswahl gezogen, welche sich innerhalb des definierten Diagrammrands befinden. Zur Lösung des aufkommenden Minimum-Distanz-Problems werden in Verbindung mit den getrennten Wertelinienmasken aus dem vorherigen Schritt diese Labels den jeweils nächsten Linien mithilfe der Ungarischen Methode bestmöglich zugeordnet.

4.3.3 Grafische Darstellung und tabellarische Auswertung

Im Anschluss an die Wertelinien- und Achsenbeschriftungsextraktion kann das zu auswertende Liniendiagramm visualisiert und in Tabellenform ausgewertet werden. Für die grafische Darstellung wurde die Datenvisualisierungsbibliothek Matplotlib [15] verwendet.



Abbildung 4.14: Zu auswertendes Liniendiagramm

Die extrahierten kontinuierlichen Linienwerte können nach vorherigem Skalieren, basierend auf der Bildgröße des Diagramms, durch Matplotlib dargestellt werden. Jeder dieser Linien kann eine Bezeichnung zugeordnet werden, welche in der Legende aufgezählt wird. Zudem kann der Titel des Graphs (plot) gesetzt werden. Die Position der jeweiligen Achsenticks kann, wie vermerkt, durch die Mittelpunktpositionen der einzelnen eingeteilten Achsenbeschriftungen eingestellt werden, sowie deren jeweilige Labels. Pro X-Achsen Label wurde außerdem eine vertikale, gestrichelte Linie visualisiert.

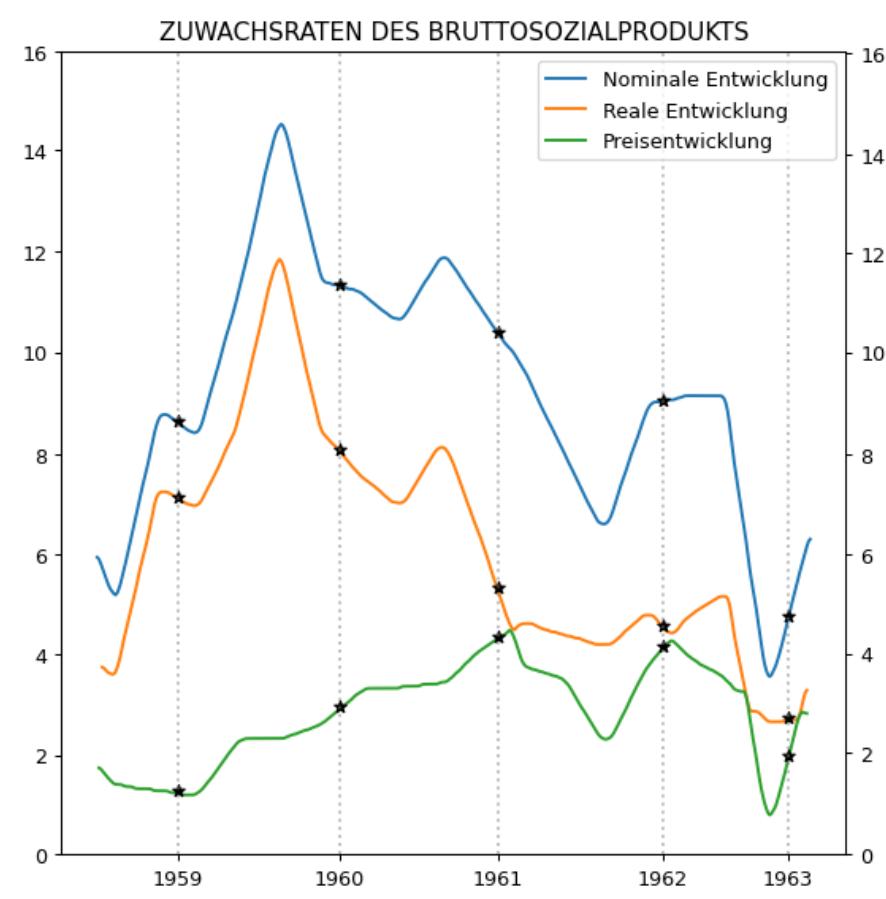


Abbildung 4.15: Durch Matplotlib visualisiertes, extrahiertes Liniendiagramm

Die eigentliche numerische Auswertung erfolgt nun ohne weiteren großen Aufwand. Für jedes Label der X-Achse wird dessen Schnittpunkt einer von seinem Mittelpunkt aus verlaufenden, vertikalen Linie mit allen Werteliniensequenzen berechnet und die räumliche Y-Position bestimmt. Anschließend wird durch lineare Inter- und Extrapolation mithilfe der SciPy-Bibliothek [16] der genaue Y-Wert zwischen den Labelwerten der linken Y-Achse berechnet. Zu notieren ist hier, dass diese lineare Interpolation bei Diagrammen mit beispielsweise logarithmischer Skala zu Evaluationsfehlern führt. Diese Diagrammform ist jedoch so selten vorkommend, dass diese fehlerbehaftete Herangehensweise akzeptabel erscheint. Zu visuellen Zwecken wurden diese Schnittpunkte ebenfalls in dem grafisch dargestellten Graphen veranschaulicht.

	1959	1960	1961	1962	1963
Nominale Entwicklung	8.63	11.33	10.39	9.04	4.73
Reale Entwicklung	7.12	8.08	5.29	4.53	2.70
Preisentwicklung	1.23	2.91	4.32	4.14	1.94

Tabelle 4.1: In Tabellenform ausgewertetes Liniendiagramm

Kapitel 5

Experimente

Die verwendeten Metriken der Experimente fassen sich aus Präzision (precision), Erinnerung (recall), F1-Wert (F1-Score) und Genauigkeit (accuracy) zusammen. Zum Berechnen dieser wird das Aufkommen der Richtig Positiven (TP), Falsch Positiven (FP), Falsch Negativen (FN) und Richtig Negativen (TN) Vorhersagen (predictions) des Modells benutzt. Die Kategorie TP zeigt vom Modell richtig erkannte Objekte, welche tatsächlich vorhanden sind, FP bestimmt die falsche Erkennung des Modells von Objekten, die in Wirklichkeit nicht vorhanden sind, FN gibt Auskunft über Objekte, die in der Realität vorhanden sind, das Modell sie allerdings nicht erkannt hat, und TN beschreibt Objekte, die korrekt als nicht vorhanden erkannt wurden.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Precision misst den Anteil der korrekten positiven Vorhersagen an allen positiven Vorhersagen des Modells. Sie zeigt, wie genau das Modell bei der Erkennung von Objekten ist und wie gut es falsche positive Ergebnisse vermeidet. Ein hoher Precision-Wert bedeutet, dass wenn das Modell ein Objekt erkennt, es mit hoher Wahrscheinlichkeit tatsächlich vorhanden ist.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Recall misst den Anteil der korrekt erkannten positiven Instanzen an allen tatsächlichen positiven Instanzen. Es zeigt, wie gut das Modell alle vorhandenen Objekte einer Klasse findet. Ein hoher Recall-Wert bedeutet, dass das Modell die meisten der tatsächlich vorhandenen Objekte erkennt.

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Der F1-Wert ist das harmonische Mittel aus Precision und Recall. Ein hoher F1-Wert deutet darauf hin, dass das Modell sowohl präzise als auch umfassend in seinen Vorhersa-

gen ist. Der F1-Wert ist besonders nützlich, wenn ein ausgewogenes Verhältnis zwischen Precision und Recall wichtig ist.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Die Richtigkeit (Accuracy) misst den Anteil aller korrekten Vorhersagen an der Gesamtzahl der Vorhersagen. Sie gibt nur an, wie oft das Modell insgesamt richtig liegt, weshalb sie nur bei der Klassifizierung verwendet wurde.

Zum Visualisieren der Konfusionsmatrizen in diesem Kapitel wurde eine Kombination von [16, 17] verwendet.

5.1 Objekterkennung

Für das Unterkapitel der Objekterkennung wurde die Effizienz, verschiedene Diagrammarten aus Texten mithilfe der Ultralytics YOLO Objekterkennungsmodelle zu extrahieren, untersucht. Hierfür wurden die erstellten Datensätze aus 3.1.1 zum Vortrainieren und aus 3.1.2 zum Feintrainieren verwendet.

Ultralytics YOLO generiert nach abgeschlossenem Modelltraining automatisch verschiedene Metrikseinblicke. Insbesondere das, eines F1-Confidence Diagramms. Aus diesem kann abgelesen werden, wie ein beliebiger Konfidenz Schwellenwert (confidence threshold) der Modellvorhersagen die Evaluationsresultate des F1-Scores beinflusst. Im Folgenden werden für alle Objekterkennungsexperimente jedene Konfidenzwerte gewählt, welche den F1-Wert maximieren.

5.1.1 Vortrainiertes Modell auf DocBank

Um im Verlauf der Datensatzerstellung zu erforschen ob die visuell ähnliche Klassen der Balkendiagramme und Histogramme weiterhin differenziert werden sollen, wurde nach nach dem Erreichen einer Datensatzgröße von 200 Bildern der Datensatz mit Histogrammen, sowie derselbe mit reduzierten Klassen evaluiert.

200 Trainingsbilder (mit Histogramme)

Aufgrund der limitierten Anzahl an verfügbaren Validationsbildern bei einem so kleinen Datensatz können allerdings schwer überzeugende Aussagen getroffen werden. Jedoch fällt die Verwechslung zwischen Balken und Histogramme in der Konfusionsmatrix trotzdem schnell auf. Grundsätzlich fallen die Evaluationswerte der Histogramm Klasse vergleichsweise schlechter aus, weshalb im Weiteren die manuelle Annotation ohne Unterscheidung

zwischen Histogrammen und Balkendiagrammen fortgeführt wurde.

Die Evaluation wurde mit einem Confidence Threshold von 40.4% durchgeführt.

Konfusionsmatrix							
OUTPUT \ TARGET	Sonstige	Linien	Balken	Histogramme	Gemischt	Hintergrund	SUM
Sonstige	5 10.64%	0 0.00%	0 0.00%	1 2.13%	0 0.00%	0 0.00%	6 83.33% 16.67%
Linien	2 4.26%	6 12.77%	0 0.00%	0 0.00%	1 2.13%	2 4.26%	11 54.55% 45.45%
Balken	0 0.00%	1 2.13%	10 21.28%	1 2.13%	0 0.00%	1 2.13%	13 76.92% 23.08%
Histogramme	0 0.00%	0 0.00%	3 6.38%	4 8.51%	2 4.26%	1 2.13%	10 40.00% 60.00%
Gemischt	1 2.13%	0 0.00%	0 0.00%	0 0.00%	3 6.38%	1 2.13%	5 60.00% 40.00%
Hintergrund	1 2.13%	0 0.00%	0 0.00%	1 2.13%	0 0.00%	0 0.00%	2 0.00% 100.00%
SUM	9 55.56% 44.44%	7 85.71% 14.29%	13 76.92% 23.08%	7 57.14% 42.86%	6 50.00% 50.00%	5 100.00% 0.00%	28 / 47 59.57% 40.43%

Abbildung 5.1: Konfusionsmatrix des Modells, trainiert auf 200 Trainingsbildern mit Histogramme

	Precision	Recall	F1-Score
Sonstige	83.33%	55.56%	66.67%
Linien	54.55%	85.71%	66.67%
Balken	76.92%	76.92%	76.92%
Histogramme	40.00%	57.14%	47.06%
Gemischt	60.00%	50.00%	54.55%
Alle	62.96%	65.07%	62.37%

Tabelle 5.1: Evaluationswerte des Modells, trainiert auf 200 Trainingsbildern mit Histogramme

200 Trainingsbilder (ohne Histogramme)

Mit einem Konfidenzwert von 51.1% wurden folgende Ergebnisse des trainierten Modells auf den Datensatz der reduzierten Klassenanzahl erzielt. Auffallend sind nicht nur, der limitierten Evaluationsdatensatzgröße verschuldet, leichte Abweichungen der Linien und gemischten Klassen, sondern auch deutliche Verbesserungen der nun zusammengesetzten Balkendiagramme.

Konfusionsmatrix						
TARGET OUTPUT \	Sonstige	Linien	Balken	Gemischt	Hintergrund	SUM
Sonstige	5 11.63%	0 0.00%	1 2.33%	0 0.00%	0 0.00%	6 83.33% 16.67%
Linien	1 2.33%	6 13.95%	0 0.00%	1 2.33%	1 2.33%	9 66.67% 33.33%
Balken	0 0.00%	1 2.33%	17 39.53%	1 2.33%	0 0.00%	19 89.47% 10.53%
Gemischt	2 4.65%	0 0.00%	0 0.00%	2 4.65%	0 0.00%	4 50.00% 50.00%
Hintergrund	1 2.33%	0 0.00%	2 4.65%	2 4.65%	0 0.00%	5 0.00% 100.00%
SUM	9 55.56% 44.44%	7 85.71% 14.29%	20 85.00% 15.00%	6 33.33% 66.67%	1 100.00% 0.00%	30 / 43 69.77% 30.23%

Abbildung 5.2: Konfusionsmatrix des Modells, trainiert auf 200 Trainingsbildern ohne Histogramme

	Precision	Recall	F1-Score
Sonstige	83.33%	55.56%	66.67%
Linien	66.67%	85.71%	75.00%
Balken	89.47%	85.00%	87.18%
Gemischt	50.00%	33.33%	40.00%
Alle	72.37%	64.90%	67.21%

Tabelle 5.2: Evaluationswerte des Modells, trainiert auf 200 Trainingsbildern ohne Histogramme

321 Trainingsbilder

Nach der Entscheidung Histogramme und Balkendiagramme zu kombinieren, wurde mit dem finalen Datensatz von 321 annotierten Bildern schließlich das endgültige Modell vortrainiert. Da mit einem größeren Datensatz die Anzahl der Validationsbilder auch ansteigt, sind diese Evaluationen nun vertraulicher. Auffallend ist jedoch die deutlich schlechter ausfallende Recall Metrik bei dem gewählten Confidence Threshold von 65.3%. Das Hauptziel dieses Modells ist jedoch lediglich das Vortrainieren, weshalb die vorliegenden Ergebnisse weniger kritisch begutachtet werden können als das eigentliche Feintrainierte Modell. Grundsätzlich ist jedoch herauszunehmen, dass eine Trainingsbildgröße von 321 ungenügend ist.

Konfusionsmatrix						
TARGET \ OUTPUT	Sonstige	Linien	Balken	Gemischt	Hintergrund	SUM
Sonstige	8 10.81%	3 4.05%	0 0.00%	1 1.35%	0 0.00%	12 66.67% 33.33%
Linien	0 0.00%	13 17.57%	1 1.35%	3 4.05%	0 0.00%	17 76.47% 23.53%
Balken	2 2.70%	0 0.00%	20 27.03%	0 0.00%	0 0.00%	22 90.91% 9.09%
Gemischt	1 1.35%	1 1.35%	0 0.00%	7 9.46%	0 0.00%	9 77.78% 22.22%
Hintergrund	5 6.76%	4 5.41%	2 2.70%	3 4.05%	0 0.00%	14 0.00% 100.00%
SUM	16 50.00% 50.00%	21 61.90% 38.10%	23 86.96% 13.04%	14 50.00% 50.00%	0 NaN% NaN%	48 / 74 64.86% 35.14%

Abbildung 5.3: Konfusionsmatrix des Modells, trainiert auf 321 Trainingsbildern

	Precision	Recall	F1-Score
Sonstige	66.67%	50.00%	57.14%
Linien	76.47%	61.90%	68.42%
Balken	90.91%	86.96%	88.89%
Gemischt	77.78%	50.00%	60.87%
Alle	77.96%	62.22%	68.83%

Tabelle 5.3: Evaluationswerte des Modells, trainiert auf 321 Trainingsbildern

5.1.2 Feintraining auf historische Wirtschaftsscans

Im Anschluss wurde das vortrainierte Modell verwendet um das Feintrainieren auf dem erstellten Datensatz der historischen Wirtschaftsmagazine auszuführen.

2444 Trainingsbilder

Die Metriken wurden mit einem Konfidenzwert von 89.1% evaluiert. Aus der Konfusionsmatrix lassen sich die Klassenverteilungen ablesen, wobei die Liniendiagrammsklasse hier eindeutig überwiegt. Dies spiegelt sich auch in der Tabelle wieder, bei der nur bei der Klasse der Linien sich ein ausgewogener Precision und Recall Wert auslesen lässt. Die geringe Existenz der Nicht-Linien Diagramme in den Wirtschaftsscans lassen so nur die glaubwürdige Evaluation der Liniendiagramme selbst zu, welche mit einer Precision von 95.83% und Recall von 94.52% auch gute Ergebnisse liefert, weshalb im weiteren Verlauf der Arbeit sich nur auf die Auswertung dieser fokussiert wurde.

Konfusionsmatrix						
TARGET OUTPUT \	Sonstige	Linien	Balken	Gemischt	Hintergrund	SUM
Sonstige	15 11.81%	0 0.00%	0 0.00%	1 0.79%	0 0.00%	16 93.75% 6.25%
Linien	1 0.79%	69 54.33%	1 0.79%	1 0.79%	0 0.00%	72 95.83% 4.17%
Balken	0 0.00%	0 0.00%	16 12.60%	0 0.00%	0 0.00%	16 100.00% 0.00%
Gemischt	0 0.00%	0 0.00%	0 0.00%	10 7.87%	0 0.00%	10 100.00% 0.00%
Hintergrund	2 1.57%	4 3.15%	6 4.72%	1 0.79%	0 0.00%	13 0.00% 100.00%
SUM	18 83.33% 16.67%	73 94.52% 5.48%	23 69.57% 30.43%	13 76.92% 23.08%	0 NaN% NaN%	110 / 127 86.61% 13.39%

Abbildung 5.4: Konfusionsmatrix des feintrainierten Modells

	Precision	Recall	F1-Score
Sonstige	93.75%	83.33%	88.24%
Linien	95.83%	94.52%	95.17%
Balken	100.0%	69.57%	82.05%
Gemischt	100.0%	76.92%	86.96%
Alle	97.40%	81.09%	88.11%

Tabelle 5.4: Evaluationswerte des feintrainierten Modells

5.2 Liniendiagrammsklassifizierung

Für die Liniendiagrammsklassifizierung wurden zwei Modelle jeweils auf die erstellten Datensätze aus 3.2 trainiert. Die Konfusionsmatrix 5.5 macht die Schwierigkeit des Modells deutlich, überlappende und nicht überlappende Liniendiagramme zu differenzieren. Im Gegensatz zu 5.6 werden hier 7 der insgesamt 19 nicht überlappenden Aufkommen falsch als überlappende Diagramme klassifiziert.

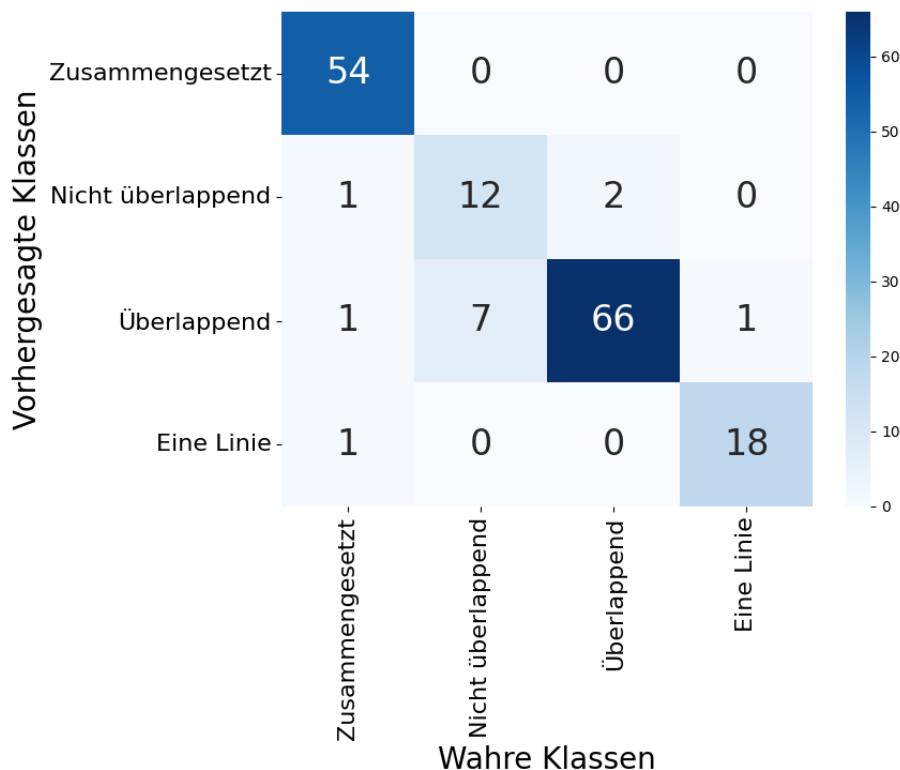


Abbildung 5.5: x

Die besseren Ergebnisse der Unterscheidung zwischen einer und mehreren Wertelinien dagegen, spiegelt sich aber auch in der Accuracy wider: Statt 92.0% des ersten Modells

erzielte dieses eine Klassifikationsgenauigkeit von 98.1%.

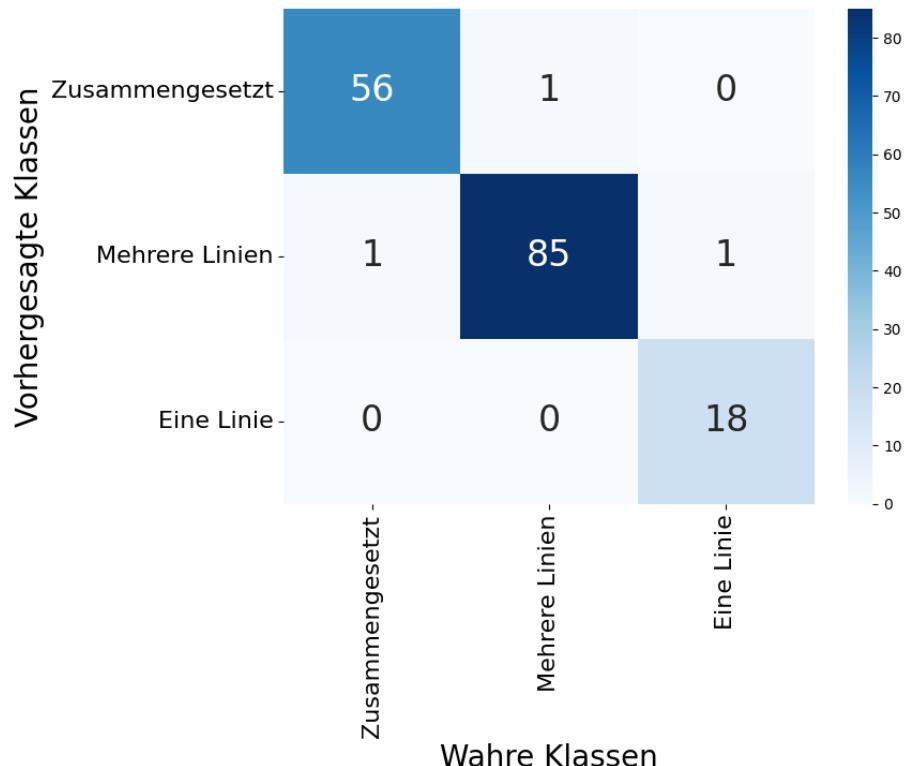


Abbildung 5.6: x

5.3 Segmentation

5.3.1 Instanzsegmentation durch Ultralytics YOLO

	Precision	Recall	F1-Score
Box	95.6%	78.8%	86.4%
Maske	93.3%	76.4%	84.0%

Tabelle 5.5: x

5.3.2 Semantische Segmentation durch das U-Net

Mit den genannten Parametern aus 4.2 wurden insgesamt fünf verschiedene Modelle trainiert: Eines wurde auf den in 3.3.2 beschriebenen Binärmaskendatensatz der Wertelinien von synthetischen Liniendiagrammen vorgenommen. Die restlichen vier Modelle wurden auf den Werteliniendatensatz der historischen Liniendiagrammen so trainiert, um jeweils den

Vergleich zwischen dem Fein- und nicht Feintrainieren und der selbst eingeführten Beschreibung des *32x Batch-Padding* darzustellen. Was dieses Batch-Padding beschreibt ist lediglich die 32-fache Duplikation aller einzelnen Trainingsdatenbilder. Da jeder einzelne Bilderbatch während des Trainingsprozesses zufällig neu augmentiert wird, und so durch das Batch-Padding die limitierten Trainingsdaten möglicherweise künstlich vergrößern werden könnten, wurde dies untersucht.

Alle im folgenden dargestellte Ergebnisse wurden auf denselben, den der historischen Diagrammwertelinien, Validationsdatensatz evaluiert. Um die Effizienz des erstellten Datensatzgenerators anschaulich zu machen wurde das auf den synthetischen Bildern vortrainierte Modell ebenfalls auf diesem Validationsdatensatz gemessen. Allerdings wurden hier keine herausragenden Evaluationswerte erwartet, da synthetische Datensätze in der Regel nicht die Qualität von realen Daten erreichen können.

	Precision	Recall	F1-Score
Maske	75.55%	66.19%	70.56%

Tabelle 5.6: Training auf synthetischem Datensatz

Die Tauglichkeit dieser künstlichen Herangehensweise zum lediglichen Vortrainieren spiegelt sich jedoch in den unteren Tabellen wider. Besonders der Precision Wert zwischen vor- und nicht vortrainierten Modellen zeigt Verbesserungen, hier in beiden Fällen von fast 1.5%. Recall scheint davon nicht ganz betroffen zu sein, die Verbesserungen bei dem einen und Verschlechterungen bei dem anderen können allerdings der Zufälligkeitfaktoren des Trainings verschuldet sein. Die gleichen Auffälligkeiten können im Fall des Batch-Paddings gefunden werden.

	Precision	Recall	F1-Score
Maske	91.52%	89.36%	90.43%

Tabelle 5.7: Training auf vortrainiertem Modell mit 32x Batch-Padding

	Precision	Recall	F1-Score
Maske	90.41%	89.57%	89.99%

Tabelle 5.8: Training auf vortrainiertem Modell ohne 32x Batch-Padding

	Precision	Recall	F1-Score
Maske	89.07%	89.64%	89.35%

Tabelle 5.9: Training auf nicht vortrainiertem Modell mit 32x Batch-Padding

	Precision	Recall	F1-Score
Maske	88.05%	88.88%	88.46%

Tabelle 5.10: Training auf nicht vortrainiertem Modell ohne 32x Batch-Padding

5.4 Diagrammauswertung

Da sich die Diagrammauswertung aus der Werteliniensegmentation und Achsenerkennung durch OCR zusammensetzt wurde nach obiger Evaluation des U-Nets ebenfalls die Effizienz des anderen Algorithmusteils beurteilt. Abgerundet wird das Kapitel der Experimente mit der Analyse des Diagrammauswertungsalgorithmus im Ganzen. Als Evaluationsdatensatz wurde für dieses gesamte Kapitel die durch 4.1.2 implementierte und in 5.2 evaluierten Liniendiagrammsklassen der einer und mehreren Wertelinien verwendet. Wurden Instanzen dieser falsch klassifiziert, so wurden diese bei den folgenden Auswertungen ignoriert.

5.4.1 Achsenerkennung durch OCR

Die Funktionsweise der Achsenerkennung durch optischer Schriftzeichenerkennung ist stark von der OCR selbst abhängig. Die Qualität der Erkennungen variiert stark durch die jeweilig verwendete OCR-Bibliothek, weswegen im Folgenden evaluierte Fehler in zwei, möglicherweise überlappende, Kategorien eingeteilt wurden. Diese bestehen aus den Fehlern des Algorithmus selbst, welche auch bei state-of-the-art Schriftzeichenerkennungsmethoden aufkommen würden. Da der Algorithmus das Auffinden der X-Achse im unteren Bildrand annimt, kommen eventuelle Fehler dieser Kategorie durch Diagramme mit anderen X-Achsen Positionen auf. Die andere Fehlergruppe sind der optischen Schriftzeichenerkennung selbst verschuldet, welche in Theorie durch eine bessere OCR-Bibliothek negiert werden kann.

Es wurden 53 zufällige Diagramme manuell evaluiert, bei denen insgesamt 26 davon die korrekte Achsen- und Achsenbeschriftungserkennung erfüllten. Die Fehlerquellen der restlichen 27 Bilder werden in den unteren Tabellen dargestellt.

Titel	Linke Y-Achse	Rechte Y-Achse	X-Achse
3	0	1	7

Tabelle 5.11: Von den 27 Fehlern wegen fehlerhaftem Achsenerkennungsalgorithmus

Titel	Linke Y-Achse	Rechte Y-Achse	X-Achse
6	6	4	10

Tabelle 5.12: Von den 27 Fehlern wegen fehlerhafter optischer Schriftzeichenerkennung

5.4.2 Numerische Tabellenformextraktion

Letztendlich wurde der kombinierte und gesamte Tabellenformextraktionsalgorithmus evaluiert. Da der Algorithmus jedoch mit einigen Einschränkungen entworfen und sei-

ne einzigen Komponenten bereits oben in 5.3.2 und 5.4.1 ausgewertet wurden, wurden einige Entscheidungen für die Algorithmusevaluation getroffen. Es soll die Effizienz des Algorithmus selbst bewertet werden, Fehler die aufgrund von ihm unausgelegter Problemen entstehen werden nicht in die Evaluation miteinbezogen. Folgende Annahmen wurden vorausgesetzt:

Inkorrektheiten verursacht durch die primitive Wertelinientrennung bleiben unbeachtet, Fehler und Folgefehler der in 5.4.1 beschriebener optischer Schriftzeichenerkennung werden ignoriert und die Bewertung wird unabhängig des korrekten Linientitels der Legende durchgeführt. Außerdem werden alle Werte anhand der Mitte des jeweiligen X-Achsen Labels abgelesen und nicht, wie korrekt, am eigentlichen respektiven Jahresende.

Fehler werden durch den in 5.4.1 genannten, fehlerhaften Achsenerkennungsalgorithmus gezählt, sowie bei falscher Y-Wert Auswertung, die möglicherweise durch falscher Interpolation der Y-Achse bei Diagrammen mit beispielweiser logarithmischen Skala auftreten kann. Des Weiteren werden Fehler bei der falschen Werteliniensegmentation bestimmt, wie etwa beim Auftreten von Lücken oder irrtümlicher Erkennung in der Segmentierungsmaske.

Insgesamt wurden 42 zufällig ausgewählte Liniendiagramme basierend auf den zuvor genannten Umständen manuell evaluiert.

Die durchschnittliche Precision dabei liegt bei 91.64% und die des Recalls bei 90.41%.

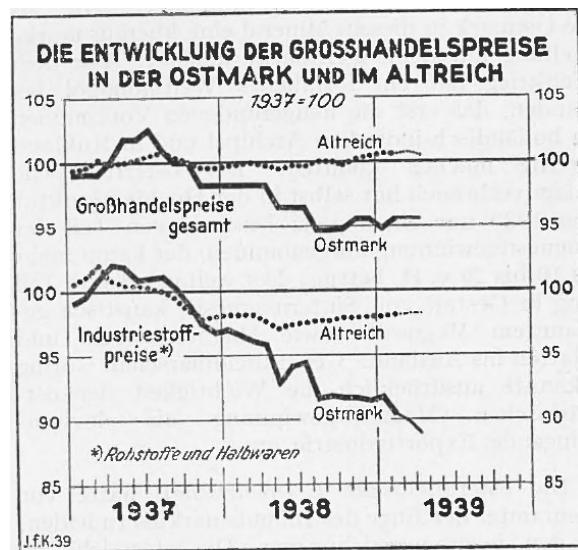


Abbildung 5.7: Liniendiagramm mit komplexer Y-Achse

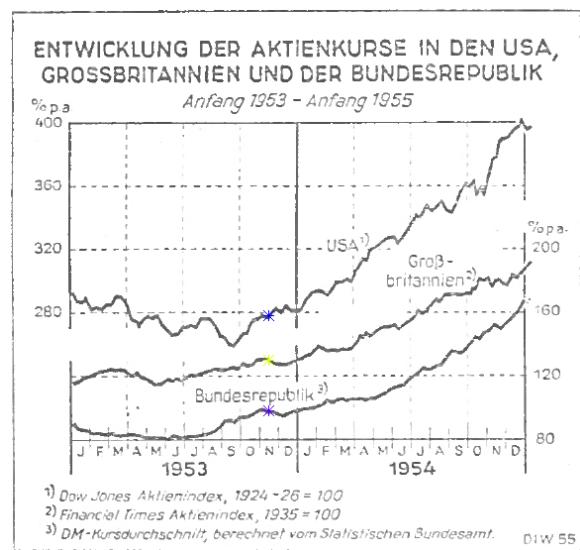


Abbildung 5.8: Liniendiagramm mit unüblicher X-Achse

Fehlergründe lagen hier bei vereinzelt vorkommenden Lücken in der Werteliniensegmentation, aber der Großteil war dem Achsenerkennungsalgorithmus verschuldet. Abbildungen 5.7 und 5.8 zeigen Beispiele von zu Fehler führenden Diagrammen. Bei 5.7 wird die Y-

Achse nach Erreichen des Werts 100 auf 95 zurückgesetzt um zwei weitere Wertelinien einzubringen, welches Fehler in der linearen Y-Wert Interpolation hervorruft. Bei 5.8 dagegen schlägt die Erkennung der X-Achsen fehl, da diese sich unüblicherweise nicht mehr im unteren Rand des Bildes befindet.

Kapitel 6

Zusammenfassung

Literaturverzeichnis

- [1] Minghao Li, Yiheng Xu, Lei Cui, Shaohan Huang, Furu Wei, Zhoujun Li, and Ming Zhou. Docbank: A benchmark dataset for document layout analysis. *arXiv preprint arXiv:2006.01038*, 2020.
- [2] CVAT.ai Corporation. Computer Vision Annotation Tool (CVAT), November 2023.
- [3] Ivan Kuckir. Photopea - online photo editor, 2024. <https://www.photopea.com/>, Accessed: 2024-08-11.
- [4] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics YOLO, January 2023.
- [5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.
- [6] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.
- [7] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.
- [8] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [9] Brad Dwyer, Joseph Nelson, Trevor Hansen, et al. Roboflow (version 1.0) [software], 2024. Computer vision, <https://roboflow.com>, Accessed: 2024-08-13.
- [10] François Chollet et al. Keras. <https://keras.io>, 2015.
- [11] 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.
- [12] Karol Źak. Keras u-net collection, 2024. GitHub repository, <https://github.com/karolzak/keras-unet>, Accessed: 2024-08-14.

- [13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [14] Dimden GD. Chrome lens ocr, 2024. GitHub repository, <https://github.com/dimdenGD/chrome-lens-ocr>, Accessed: 2024-08-14.
- [15] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.
- [16] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.
- [17] DamianoP. Confusion matrix generator, 2024. GitHub repository, <https://github.com/DamianoP/confusionMatrixGenerator>, Accessed: 2024-08-17.
- [18] Shivasankaran V P, Muhammad Yusuf Hassan, and Mayank Singh. Lineex: Data extraction from scientific line charts. *2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 6202–6210, 2023.
- [19] Jaewoong Lee, Wonseok Lee, and Jihan Kim. Matgd: Materials graph digitizer, 2023.
- [20] Daekyoung Jung, Wonjae Kim, Hyunjoo Song, Jeong-in Hwang, Bongshin Lee, Bohyoung Kim, and Jinwook Seo. Chartsense: Interactive data extraction from chart images. pages 6706–6717, 05 2017.
- [21] Junyu Luo, Zekun Li, Jinpeng Wang, and Chin-Yew Lin. Chartocr: Data extraction from charts images via a deep hybrid framework. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1916–1924, 2021.
- [22] Mateusz Kozinski and Renaud Marlet. Image parsing with graph grammars and markov random fields applied to facade analysis. pages 729–736, 03 2014.

Selbstständigkeitserklärung

Hiermit erkläre ich, die vorliegende Bachelorarbeit selbstständig und ohne unerlaubte fremde Hilfe angefertigt zu haben. Alle verwendeten Quellen und Hilfsmittel sind vollständig angegeben. Ich versichere, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Würzburg, den 18. August 2024

Luzian Uihlein

