

Bachelorarbeit

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

Luzian Uihlein

Würzburg, 15. 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	5
2 Literaturübersicht	6
3 Methodik	8
3.1 Extraktion von Diagrammen aus Texten	8
3.1.1 Datensatz DocBank zur Objekterkennung	8
3.1.2 Datensatz historischer Wirtschaftsscans zur Objekterkennung . . .	10
3.2 Schwierigkeitsklassifizierung von Liniendiagrammen	11
3.3 Auswertung von Liniendiagrammen	12
3.3.1 Datensatz von synthetischen Liniendiagrammen zur Segmentation .	13
3.3.2 Datensatz von historischen Liniendiagrammen zur Segmentation .	14
4 Implementation	15
4.1 Ultralytics YOLO	15
4.1.1 Objekterkennung zur Extraktion von Diagrammen aus Texten . . .	16
4.1.2 Klassifizierung zur Schwierigkeitsbestimmung von Liniendiagrammen	17
4.1.3 Instanzsegmentation von Wertelinien in Liniendiagrammen	18
4.2 U-Net: Semantische Segmentation von Wertelinien	19
4.3 Numerische Auswertung von Liniendiagrammen in Tabellenform	21
4.3.1 Extraktion der segmentierten Wertelinien	21
4.3.2 Achsenerkennung durch optischer Schriftzeichenerkennung	23
5 Experimente	26
6 Zusammenfassung	29

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 [15, 16] 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 [17] 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 [18] 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 [19] 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 Diagrammen aus den historischen Textscans, welche dann im folgenden Teil in eine gewünschte Form ausgewertet werden können.

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

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 wissenschaftliche Publikation mit computergenerierten Grafiken zusammengesetzt, weshalb DocBanks Dokumentenseiten lediglich zum Vortrainieren des Detektionsmodells gedacht sind. Beabsichtigt wurde dieser Prozess des Vortrainierens um das System schneller und algemeingültiger, also mit besseren Voraussagen, trainieren zu können. Spätere Experimente untersuchen diese Annahme.

An die Vorkommenshäufigkeit bei den historischen Scans angepasst, wurde die Differenzierung in fünf Objektklassen beschlossen: Linien (line), Balken (bar), Histogramm (histogram), Sonstige (other) und Gemischt (mixture). Aufgrund von Verwechslungen des Modells im Verlauf der Experimente zwischen Balkendiagrammen und Histogrammen

wurden die Datensätze auf vier Klassen reduziert, indem Balkendiagramme und Histogramme vereinigt wurden.

Die Schwierigkeit zwischen Balkendiagrammen und Histogrammen zu unterscheiden beruht darauf, dass Balkendiagramme kategorische Datenvergleiche anschaulich machen, bei denen die Balkenanordnung irrelevant ist, während Histogramme kontinuierliche, numerische Daten darstellen. Die Differenz liegt 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 allerdings nicht ausschlaggebend zur Bestimmung des Diagrammtyps.

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

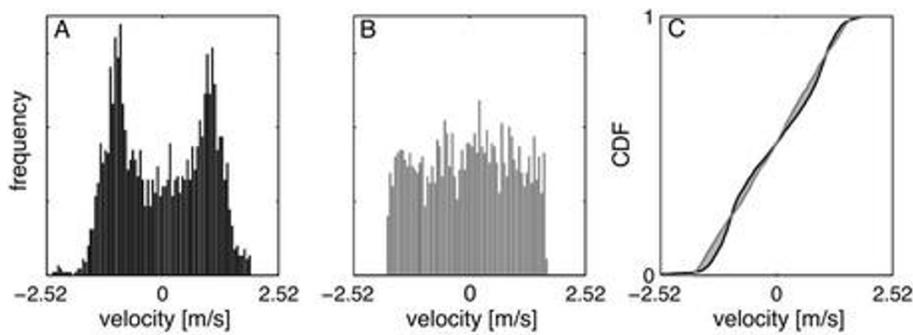


Figure 5: Panel A shows a velocity profile - histogram (h_A ; black) of the

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 wissenschaftlichen 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 darunterliegenden Abbildungsunterschrift gehalten.

Insgesamt wurden 321 Seiten annotiert, beinhaltend aus 105 Liniendiagrammen, 115 Balkendiagrammen (vereinigt mit Histogrammen), 79 sonstige und 66 gemischte Diagrammen.

3.1.2 Datensatz historischer Wirtschaftsscans zur Objekterkennung

Die Scans der geschichtlichen Wirtschaftsmagazine wurden mit ähnlichen Überlegungen annotiert. Hier befinden sich ebenfalls Diagrammgruppen, teils auch mit mehreren verschiedenen Diagrammtypen, siehe Abbildung 3.2, welche alle wieder als gesamte Gruppe annotiert wurden. Bis auf sehr wenigen Ausnahmen, befinden sich alle Abbildungen in den Scans visuell eingerahmt. Da die Ausrichtung derer 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 Landeskarten 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 ergeben.

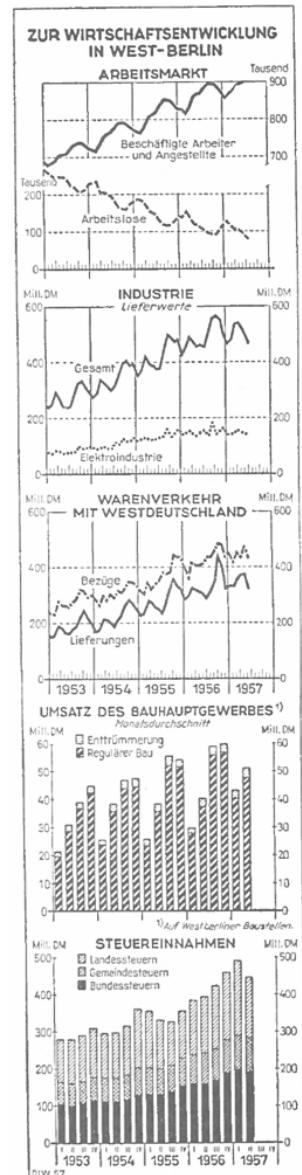


Abbildung 3.2: Diagrammbeispiel historischer Scans

3.2 Schwierigkeitsklassifizierung von Liniendiagrammen

Aufgrund der überwiegenden Liniendiagrammen in den historischen Wirtschaftsscans, wurde sich im Folgenden primär auf die Auswertung der Liniendiagrammen 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.

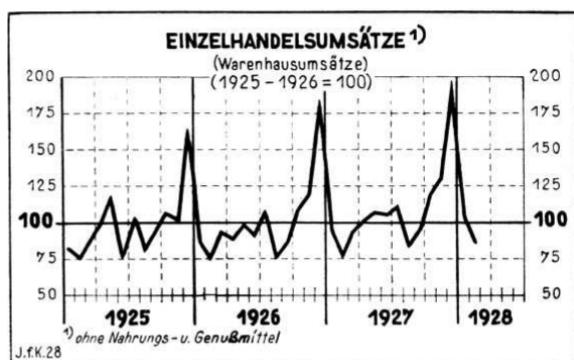


Abbildung 3.3: Liniendiagramm mit einer Wertelinie

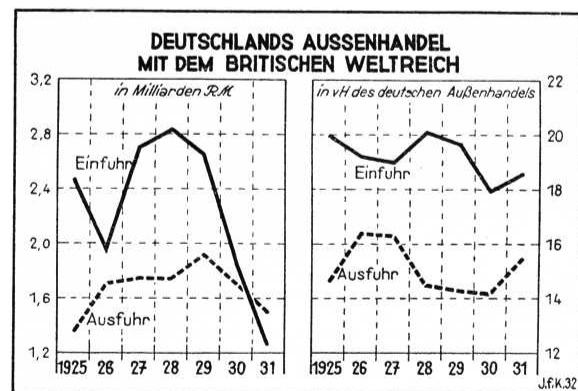


Abbildung 3.4: Zusammengesetzte Liniendiagrammsgruppe

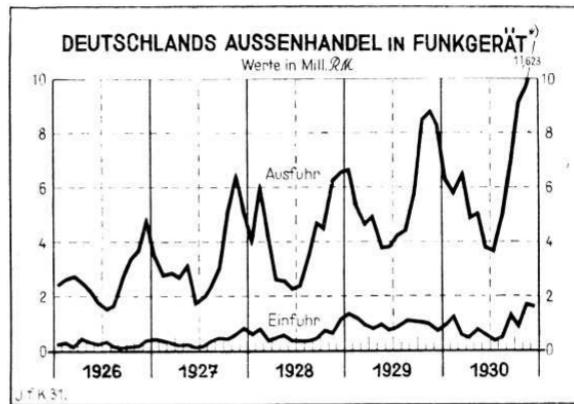


Abbildung 3.5: Liniendiagramm mit sich nicht überlappenden Wertelinie

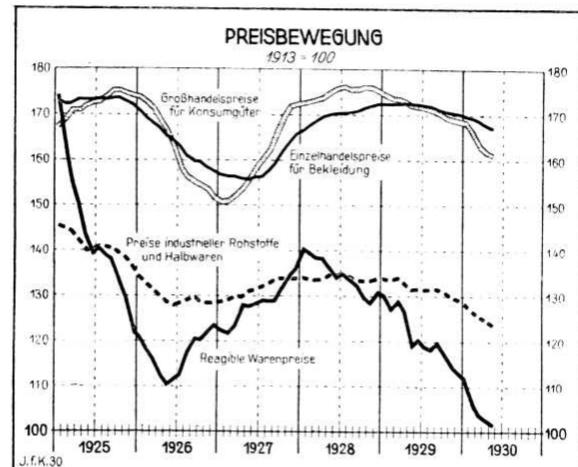


Abbildung 3.6: Liniendiagramm mit sich überlappenden Wertelinie

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 diesem wurde aufgrund späteren Klassifizierungsproblemen von sich nicht überlappenden mit überlappenden Wertelinien, beide Klassen in die gemeinsame Differenzierung der Liniendiagrammen 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 impraktibel ist. Zum einen führen die nicht genau senkrecht und waagerecht verlaufenden Gitterlinien die korrekte Erkennung dieser zu einem nichttrivialen Erkennungsproblem und zum andern 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 semantischer 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

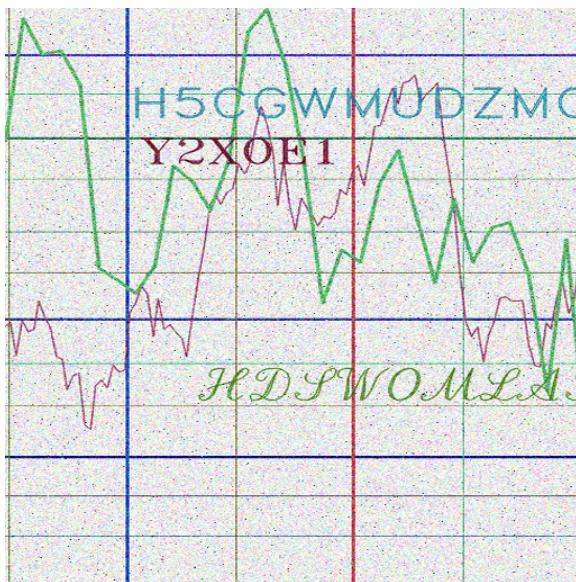


Abbildung 3.7: Synthetisch erstelltes Liniendiagramm

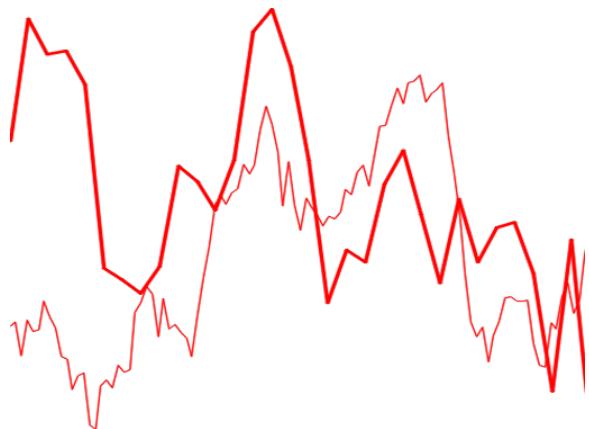


Abbildung 3.8: Zugehörige generierte Binärmaske der Wertelinien

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 überlappenden, Textbeschriftungen vielfältiger Schrifgrößen und Schriftarten. Da beim Generierungsprozess alle Diagrammwerte natürlicherweise bekannt sind, können diese einfach auf einem zweiten, leeren Bild übertragen werden, um so die zugehörige Wertelinienbinärmaske 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 Annotation im Format von Polygonumzeichnungen. Ist dies der Fall, können die getrennten Wertelinienbinärmasken unter anderem mit Hilfe von Konturerkennung 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 heranzukommen.

3.3.2 Datensatz von historischen Liniendiagrammen zur Segmentation



Abbildung 3.9: Historisches Liniendiagramm

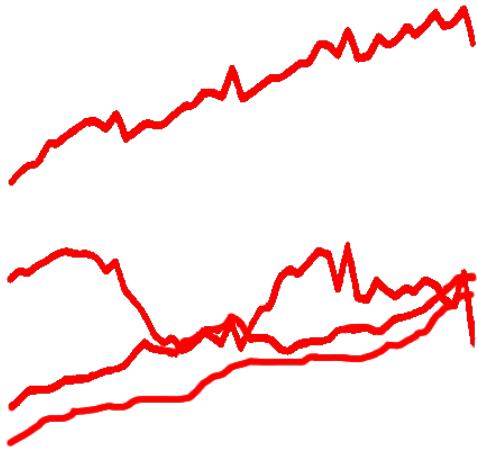


Abbildung 3.10: Zugehörige manuell erstellte Binärmasken der Wertelinien

Die manuelle Erstellung der Wertelinienbinärmasken der historischen Liniendiagrammen 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 beschreiben, durch Konturerkennung von der Binärmask 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 erstmal 2015 [6] veröffentlicht wurde, und seit dem 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 der automatischen Datenagumentation während des Trainingsvorgangs und der Evaluation verschiedener Metriken des trainierenden 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 eingeteilt wurde auf 80 Objektklassen.

Diese werden außerdem in verschiedene Modellgrößen angeboten, sodass die Möglichkeit besteht zwischen Invarianzgeschwindigkeit, benötigte Gleitkommaoperationsleistungsfähigkeiten und verwendbaren Grafikkartenspeicher abwegegen 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 Diagrammen 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 gesehenen 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 Parametern 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 universelle 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 Liniendiagrammen 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 Epochengrenzung. 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 zu besseren Ergebnissen, weshalb sie mit einbezogen wurden. Die restlichen Augmentierungen wurden für die breitere Diagrammsvarianz gewählt.

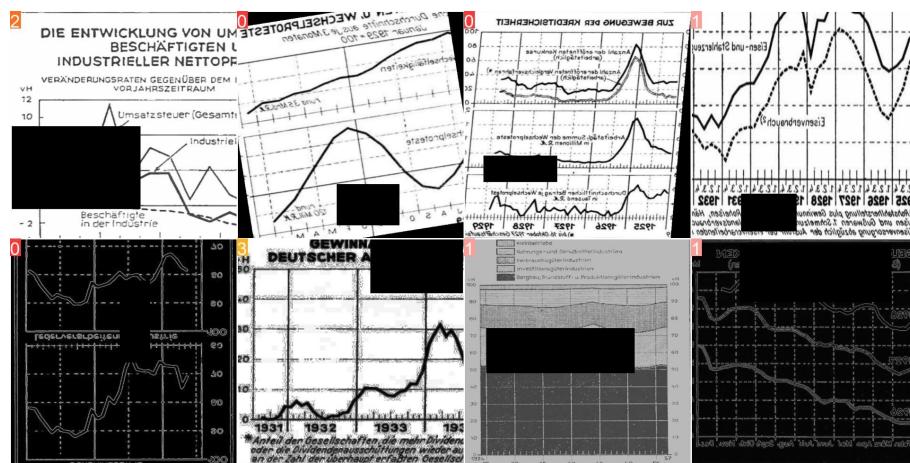


Abbildung 4.1: Augmentierter Trainingsbatch der Klassifizierung

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, von dem 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 den von Ultralytics erstellten Trainingsbatches, als auch durch Auswertung mithilfe der Annotationswebseite Roboflow [9] überprüft werden. Im Anschluss konnte das YOLOv8x-seg Model 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)

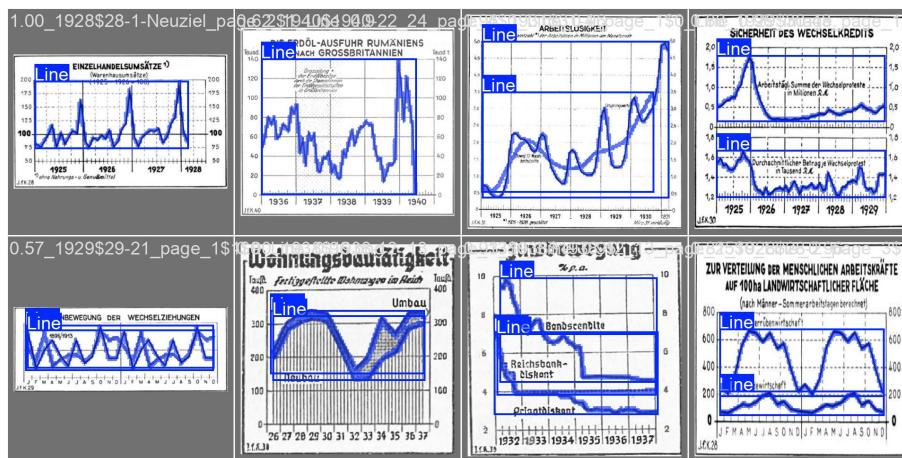


Abbildung 4.2: Annotierte Wertelinien der Liniendiagrammen zur Instanzsegmentation

Hier wurden stärkere Augmentierungstechniken gewählt, da Wertelinien unabhängig von Positionskontexten ausfindig gemacht werden müssen, weshalb additionell 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 medizinisch 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 Datenagumentation 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.

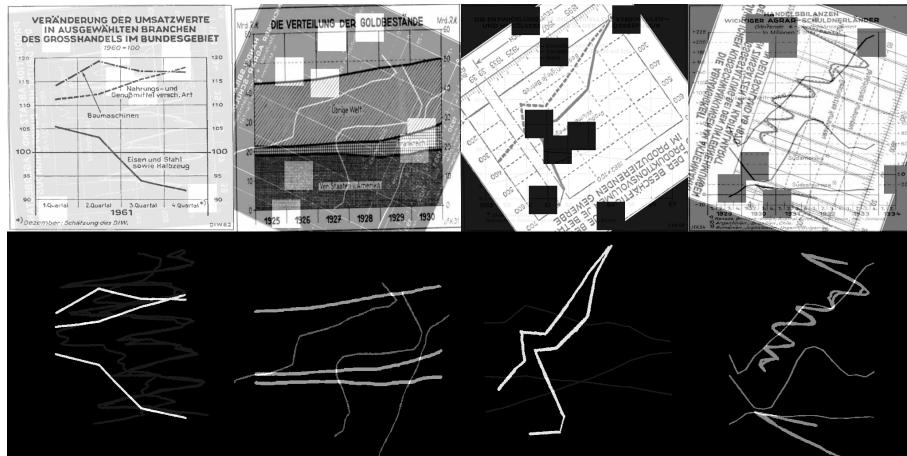


Abbildung 4.3: Augmentierter Trainingsbatch inklusive der zugehörigen Binärmasken

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 Learnrate (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 möglichgemachte, Wertelinienerkennung und einer optischen Schriftzeichenerkennung (optical character recognition; OCR), aus welcher unter anderem Achsenbeschriftungen und deren numerischen Kontext extrahiert werden können. Umgesetzt wurde dieser Algorithmus ebenfalls der Programmiersprache Python. Im Folgenden wird die Funktionsweise des zweigeteilten Algorithmus im Nähen beschreiben.

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 auswerten-
des Liniendiagramm

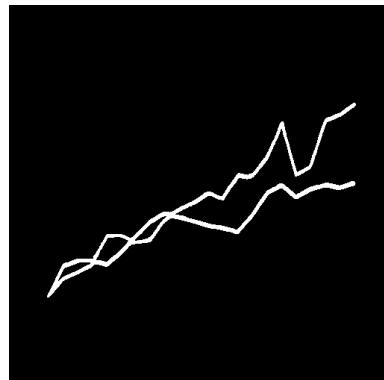


Abbildung 4.5: Semantische
U-Net Vorhersage

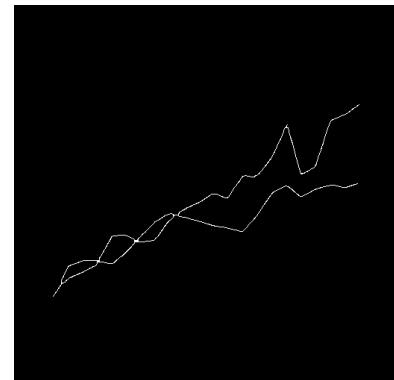


Abbildung 4.6: Skelletierte Bi-
närmaske

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 Pixel gezählt. Da im Fall von vertikal verlaufenden Wertelinien mehrere von diesen aufeinanderfolgen können muss eine Gruppe an sequentiellen nicht-schwarzen Pixel 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 Pixel der *i-ten disjunkten Wertelinienbinärmaske* zugeordnet. Sollte dies allerdings nicht der Fall sein, wird die gesamte Pixelspalte in eine *gemeinsam genutzte Binärmaske* übertragen.

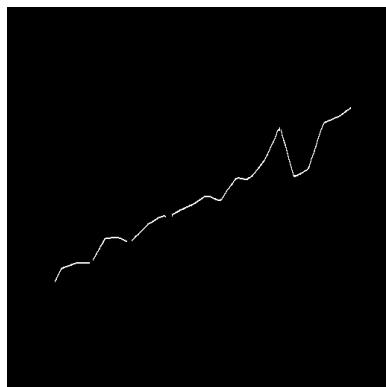


Abbildung 4.7: Disjunkte Binärmasken der ersten Linie

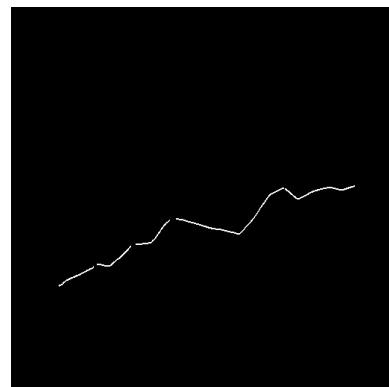


Abbildung 4.8: Disjunkte Binärmasken der zweiten Linie

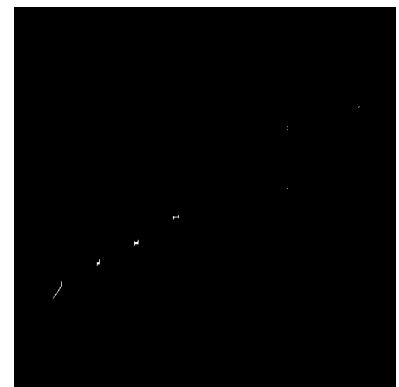


Abbildung 4.9: Gemeinsam genutzte Binärmasken

Somit können die Wertelinien anhand ihrer Position differenziert werden. Um nun die einzelnen kontinuierlichen Linienwerte zu extrahieren, müssen die disjunktiven 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 zu entgegenwirken wird nach Gaußscher Weichzeichnung (gaussian blur) mit einem Kern von 5x5 Pixel die Positionssequenz jeder getrennten Wertelinie im Diagramm ermittelt.

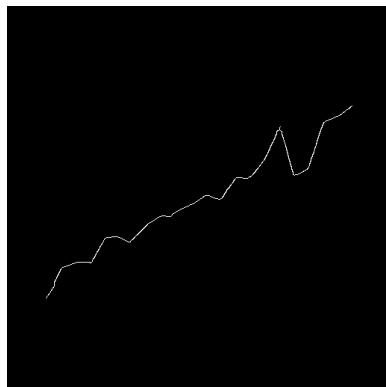


Abbildung 4.10: Vereinigte Binärmasken der ersten Linie

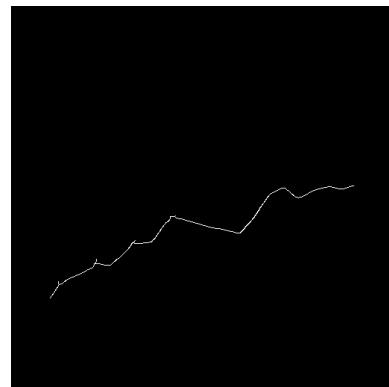


Abbildung 4.11: Vereinigte Binärmasken der zweiten Linie

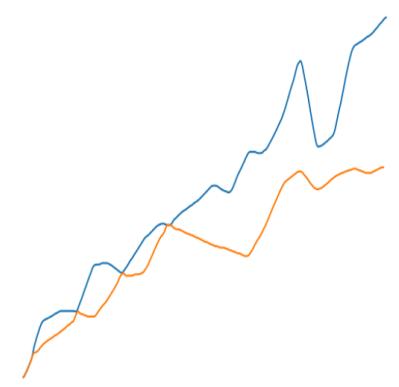


Abbildung 4.12: Getrennte Wertelinien

4.3.2 Achsenerkennung durch optischer Schriftzeichenerkennung

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

Bis auf limitierte Ausnahmen befinden sich so alle Datenpunkte, beispielweise bei Diagramme 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. 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.

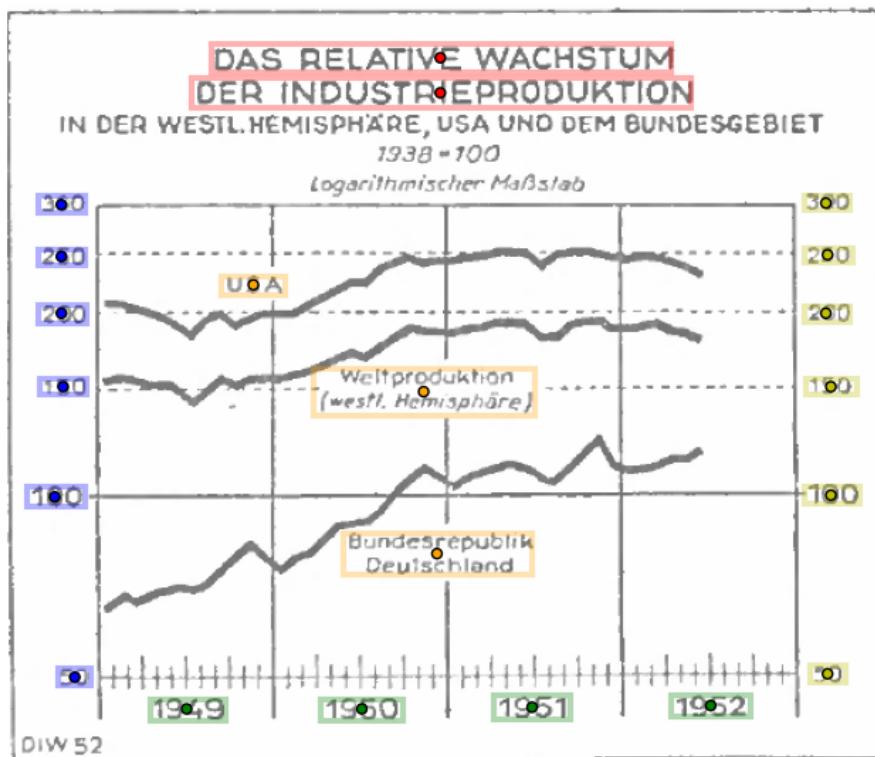


Abbildung 4.13: buh

Nach dem 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. Für die drei Achsenbeschriftungen werden nur jene Labels untersucht, welche sich auf dem äußeren Rand von 15% des Bildes befinden. Die horizontale und vertikale Mittelwertposition 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, Punkte 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.

-> titel und rest

Kapitel 5

Experimente

Die verwendeten Metriken bei der Objekterkennung fassen sich aus Präzision (precision), Erinnerung (recall), mAP50 und mAP50-95 zusammen. Zum Berechnen dieser wird das Aufkommen der Richtig Positiven (TP), Falsch Positiven (FP) und Falsch Negativen (FN) 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 und FN gibt Auskunft über Objekte die in der Realität vorhanden sind, das Modell sie allerdings nicht erkannt hat.

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.

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

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.

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

mAP50 (mittlere durchschnittliche Präzision bei IoU=0.5) ist eine Metrik, die die Genauigkeit des Modells bei der Objekterkennung über alle Klassen hinweg misst. Dabei wird ein Intersection over Union (IoU) Schwellenwert von 0,5 verwendet. Eine Erkennung gilt als korrekt (TP), wenn die IoU zwischen der vorhergesagten und der tatsächlichen

Bounding Box größer als 0,5 ist. Ein hoher mAP50-Wert zeigt, dass das Modell einfache Objekte verschiedener Klassen zuverlässig erkennt und lokalsiert.

$$mAP50 = \frac{1}{n} \sum_{i=1}^n AP_i$$

wobei n die Anzahl der Klassen ist und AP_i die durchschnittliche Präzision für die i -te Klasse bei $IoU=0.5$.

mAP50-95 (mittlere durchschnittliche Präzision bei $IoU=0.5:0.95$) ist eine umfassendere Metrik, die die Leistung des Modells über verschiedene IoU-Schwellenwerte hinweg misst. Sie berechnet den Durchschnitt der mAP-Werte für verschiedene IoU-Schwellenwerte von 0,5 bis 0,95 in Schritten von 0,05. Dies gibt einen robusteren Überblick über die Modellleistung, da es verschiedene Grade der Überlappungsgenauigkeit berücksichtigt. Ein hoher mAP50-95-Wert zeigt, dass das Modell sowohl bei einfachen als auch bei schwereren zuverlässigen Vorhersagen trifft.

$$mAP50-95 = \frac{1}{10} \sum_{t=0.5}^{0.95} mAP_t$$

wobei t die IoU-Schwellenwerte von 0,5 bis 0,95 in Schritten von 0,05 durchläuft.

Im Kapitel Experimente Ihrer Bachelorarbeit sollten Sie die durchgeföhrten Versuche, deren Ergebnisse und Ihre Analyse darlegen. Hier ist eine Übersicht der wichtigsten Punkte, die in diesem Kapitel enthalten sein sollten:

Versuchsaufbau:

Beschreibung der verwendeten Datensätze (Trainings-, Validierungs- und Testdaten) Erklärung der Evaluierungsmaßen (z.B. mAP für YOLO, IoU für U-Net) Definition der Baseline oder Vergleichsmodelle

Durchgeföhrte Experimente:

Detaillierte Beschreibung jedes einzelnen Experiments Begründung für die Wahl der Experimente Variationen in Hyperparametern, Modellarchitekturen oder Trainingsmethoden Ergebnisse:

Präsentation der quantitativen Ergebnisse (in Tabellen oder Grafiken) Qualitative Ergebnisse (z.B. Beispielbilder von Vorhersagen) Vergleich der Leistung von YOLO und U-Net für Ihre spezifische Aufgabe

Analyse:

Interpretation der Ergebnisse Diskussion von Stärken und Schwächen der Modelle Vergleich mit dem aktuellen Stand der Technik oder anderen relevanten Arbeiten

Ablationstudie:

Untersuchung des Einflusses verschiedener Komponenten oder Hyperparameter auf die Modellleistung

Fehleranalyse:

Identifikation von häufigen Fehlertypen Diskussion möglicher Gründe für diese Fehler

Laufzeitanalyse und Ressourcenverbrauch:

Vergleich der Inferenzzeiten von YOLO und U-Net Analyse des Speicher- und Rechenbedarfs

Diskussion der Limitationen:

Grenzen der durchgeführten Experimente Mögliche Verzerrungen in den Daten oder der Evaluation

Zukünftige Arbeiten:

Vorschläge für weitere Experimente oder Verbesserungen basierend auf Ihren Ergebnissen

Dieses Kapitel sollte eine objektive Darstellung Ihrer experimentellen Arbeit sein, die es dem Leser ermöglicht, die Leistung und Eignung von YOLO und U-Net für Ihre spezifische Anwendung zu verstehen und zu bewerten.

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] 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.
- [16] Jaewoong Lee, Wonseok Lee, and Jihan Kim. Matgd: Materials graph digitizer, 2023.
- [17] Daekyoung Jung, Wonjae Kim, Hyunjoo Song, Jeong-in Hwang, Bongshin Lee, Bo-hyoun Kim, and Jinwook Seo. Chartsense: Interactive data extraction from chart images. pages 6706–6717, 05 2017.
- [18] 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.
- [19] 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 15. August 2024

Luzian Uihlein